TAL TECH

DOCTORAL THESIS

Methods for Reliability Assessment and Enhancement of Deep Neural Networks Hardware Accelerators

Mahdi Taheri

TALLINNA TEHNIKAÜLIKOOL
TALLINN UNIVERSITY OF TECHNOLOGY
TALLINN 2025

TALLINN UNIVERSITY OF TECHNOLOGY
DOCTORAL THESIS
4/2025

Methods for Reliability Assessment and Enhancement of Deep Neural Networks Hardware Accelerators

MAHDI TAHERI

TAL TECH PRESS

TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies
Department of Computer Systems

The dissertation was accepted for the defense of the degree of Doctor of Philosophy in Information and Communication Technologies on November 19, 2024

Supervisor: Prof. Dr. Maksim Jenihhin,
Department of Computer Systems
Tallinn University of Technology
Tallinn, Estonia

Adjunct Prof. Dr. Masoud Daneshtalab,
Department of Computer Systems
Tallinn University of Technology
Tallinn, Estonia

Opponents: Prof. Cristiana Bolchini,
Politecnico di Milano,
Milan, Italy

Dr. Leticia Bolzani Pöhls,
IHP - Leibniz Institute for High-Performance Microelectronics,
Frankfurt Oder, Germany

Defence of the thesis: January 24, 2025, Tallinn

Declaration:
Hereby I declare that this doctoral thesis, my original investigation and achievement, submitted for the doctoral degree at Tallinn University of Technology, has not been submitted for any academic degree elsewhere.

Mahdi Taheri

_____
Signature

European Union
European Regional
Development Fund

Investing
in your future

ISSN 2585–6898(publication)
ISBN 978-9916-80-250-2 (publication)
ISSN 2585–6901 (PDF)
ISBN 978-9916-80-251-9 (PDF)
DOI https://doi.org/10.23658/taltech.4/2025
Printed by Koopia Niini & Rauam

Taheri, M. (2025). *Methods for Reliability Assessment and Enhancement of Deep Neural Networks Hardware Accelerators* [TalTech Press]. https://doi.org/10.23658/taltech.4/2025

TALLINNA TEHNIKAÜLIKOOL
DOKTORITÖÖ
4/2025

Süvanärvivõrkude riistvara kiirendite töökindluse hindamine ja täiustamine

MAHDI TAHERI

TAL TECH
KIRJASTUS

Contents

List of Publications

The present Ph.D. thesis is based on the following publications that are referred to in the text by Roman numbers.

I M. Taheri, N. Cherezova, M. S. Ansari, M. Jenihhin, A. Mahani, M. Daneshtalab, and J. Raik, "Exploration of Activation Fault Reliability in Quantized Systolic Array-Based DNN Accelerators," in *2024 25th International Symposium on Quality Electronic Design (ISQED)*, 2024.

II M. Taheri, N. Cherezova, S. Nazari, A. Rafiq, A. Azarpeyvand, T. Ghasempouri, M. Daneshtalab, J. Raik, and M. Jenihhin, "AdAM: Adaptive fault-tolerant approximate multiplier for edge DNN accelerators," in *2024 IEEE European Test Symposium (ETS)*, 2024.

III M. Taheri, N. Cherezova, S. Nazari, A. Azarpeyvand, T. Ghasempouri, M. Daneshtalab, J. Raik, and M. Jenihhin, "AdAM: Adaptive Approximate Multiplier for Fault Tolerance in DNN Accelerators," *in IEEE Transactions on Device and Materials Reliability, doi: 10.1109/TDMR.2024.3523386*.

IV M. Taheri, M. Daneshtalab, J. Raik, M. Jenihhin, S. Pappalardo, P. Jimenez, B. Deveautour, and A. Bosio, "Saffira: a framework for assessing the reliability of systolic-array-based dnn accelerators," in *2024 27th International Symposium on Design & Diagnostics of Electronic Circuits & Systems (DDECS)*, pp. 19–24, 2024.

V M. Taheri, et al., "Appraiser: Dnn fault resilience analysis employing approximation errors," in *DDECS*, pp. 124–127, 2023.

VI M. Taheri, M. Riazati, M. H. Ahmadilivani, M. Jenihhin, M. Daneshtalab, J. Raik, M. Sjödin, and B. Lisper, "Deepaxe: A framework for exploration of approximation and reliability trade-offs in dnn accelerators," in *2023 24th International Symposium on Quality Electronic Design (ISQED)*, pp. 1–8, 2023.

VII S. Nazari, M. Taheri, A. Azarpeyvand, M. Jenihhin, M. Daneshtalab, et. al. "FORTUNE: A Negative Memory Overhead Hardware-Agnostic Fault TOleRance TechniqUe in DNNs," In *2024 33th IEEE Asian Test Symposium (ATS 2024)*.

VIII M. H. Ahmadilivani, M. Taheri, J. Raik, M. Daneshtalab, and M. Jenihhin, "DeepVigor: Vulnerability value ranges and factors for DNNs' reliability assessment," in *2023 IEEE European Test Symposium (ETS)*, pp. 1–6, 2023.

IX M. H. Ahmadilivani, M. Taheri, J. Raik, M. Daneshtalab, and M. Jenihhin, "A systematic literature review on hardware reliability assessment methods for deep neural networks," *ACM Computing Surveys*, vol. 56, no. 6, pp. 1–39, 2024.

X M. H. Ahmadilivani, M. Taheri, J. Raik, M. Daneshtalab, and M. Jenihhin, "Enhancing Fault Resilience of QNNs by Selective Neuron Splitting," in *2023 IEEE 5th International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, pp. 1–5, 2023.

XI M. H. Ahmadilivani, M. Barbareschi, S. Barone, A. Bosio, M. Daneshtalab, S. Della Torca, G. Gavarini, M. Jenihhin, J. Raik, A. Ruospo, et al., "Special Session: Approximation and Fault Resiliency of DNN Accelerators," in *2023 IEEE 41st VLSI Test Symposium (VTS)*, pp. 1–10, 2023.

XII M. H. Ahmadilivani, A. Bosio, B. Deveautour, F. F. Dos Santos, J. D. Guerrero-Balaguera, M. Jenihhin, A. Kritikakou, R. L. Sierra, S. Pappalardo, J. Raik, et al., "Special session: Reliability assessment recipes for dnn accelerators," in *2024 IEEE 42nd VLSI Test Symposium (VTS)*, pp. 1–11, 2024.

XIII M. Jenihhin, M. Taheri, N. Cherezova, M. H. Ahmadilivani, H. Selg, A. Jutman, K. Shibin, A. Tsertov, S. Devadze, R. M. Kodamanchili, et al., "Keynote: Cost-Efficient Reliability for Edge-AI Chips," in *2024 IEEE 25th Latin American Test Symposium (LATS)*, pp. 1–2, 2024.

XIV N. Cherezova, S. Pappalardo, M. Taheri, M. H. Ahmadilivani, B. Deveautour, A. Bosio, J. Raik, and M. Jenihhin, "Heterogeneous Approximation of DNN HW Accelerators based on Channels Vulnerability," in *IEEE International Conference on Very Large Scale Integration (VLSI-SOC)*, 2024.

Other related publications

XV M. Taheri, "DNN Hardware Reliability Assessment and Enhancement," *27th IEEE European Test Symposium (ETS)*, May 2022.

XVI A. Rezaei, M. Taheri, A. Mahani, and S. Magierowski, "LRDB: LSTM Raw data DNA Base-caller based on long-short term models in an active learning environment," *arXiv preprint arXiv:2303.08915*, 2023.

XVII M. Taheri, M. Taheri, and A. Hadjahmadi, "Noise-tolerance gpu-based age estimation using resnet-50," *arXiv preprint arXiv:2305.00848*, 2023.

XVIII M. Taheri, S. Sheikhpour, A. Mahani, M. Jenihhin, "A novel Fault-Tolerant Logic Style with Self-Checking Capability," *2022 IEEE 28th International Symposium on On-Line Testing and Robust System Design (IOLTS)*.

Author's Contributions to the Publications

I In Publication I, I initiated the original idea of creating a methodology for exploring quantization and reliability trade-offs in systolic-array implementations. As the main author, I wrote the simulation programs, carried out the simulations and the analysis of the results, prepared the figures, and wrote the manuscript.

II In Publication II, which is an extension to the conference version (III), I proposed the original idea of creating an adaptive fault-tolerant approximate multiplier (AdAM) tailored for ASIC-based DNN accelerators. I synthesized and tested the designs that were used to carry out experiments, and also was responsible for the preparation of the manuscript.

III In Publication IV, I pioneered the development of a hierarchical methodology for hardware-accurate reliability assessment of systolic array (SA) based DNN accelerators. As the primary contributor, I conceptualized the methodology, developed the simulation framework, conducted extensive evaluations using diverse benchmarks, and authored the manuscript.

IV In Publication V, I introduced the novel resiliency assessment method called APPRAISER, aimed at enhancing the reliability evaluation of Deep Neural Networks (DNNs) in safety-critical applications. As the principal author, I conceived the foundational concepts, developed the methodology, conducted comprehensive evaluations against state-of-the-art fault injection techniques, and authored the manuscript.

V In Publication VI, I proposed and developed the DeepAxe framework, aimed at enhancing the reliability and efficiency of FPGA-based Deep Neural Network (DNN) implementations crucial for safety-critical applications. As the first author, I led the conceptualization, development, and validation of DeepAxe, contributing significantly to the manuscript by outlining theoretical foundations, running experiments, analyzing performance metrics, and interpreting results to underscore the framework's efficacy and versatility in real-world applications.

VI In Publication VII, I proposed and developed the FORTUNE, a hardware-agnostic fault tolerance technique for DNNs that leverages quantization to enhance reliability without significant performance overhead. During this work, I led the conceptualization, development, and validation of the idea, contributing significantly to the manuscript, including running experiments, analyzing performance metrics, and interpreting results to underscore the technique's efficacy and versatility in real-world applications.

VII In Publicationss VIII, and X I was the first co-author, contributing to the original idea of the papers, participating in weekly brainstorming meetings, providing codes for benchmarks, and helping with writing the papers.

VIII In Publication IX, I was the first co-author, contributing to Reviewing the literature for the reliability assessment methods of DNNs, providing the trends of published papers over different years and characterizing and categorizing the reliability assessment methods for DNNs and also helping with writing the manuscript.

IX In Publications XI, and XII that were joint papers with two other research groups, I was the main author of the TalTech research team and contributed

by initiating the original ideas of the papers, writing the simulation programs, carrying out the simulations and the analysis of the results, prepared the figures, and wrote the manuscripts.

X In Publication XIII, that was a keynote presented by my supervisor, Maksim Jenihhin, I was the main author of 7 out of 13 papers that the keynote was based on.

XI In Publication XIV which is a heterogeneous approximation of DNN hardware sccelerators based on channels Vulnerability, I was the first co-author from TalTech, contributing to the original idea of the paper, participating in weekly brainstorming meetings, providing codes for benchmarks, leading the project, and helping with writing the paper.

XII In Publication XV, I presented my early research results at the European Test Symposium conference. My main contributions were preparation of the manuscript, organizing the results, and paper presentation.

XIII In Publication XVI, I was the first co-author, contributing to the original idea of the paper, providing codes, developing novel LSTM training process, and helping with writing the paper.

XIV In Publication XVII, which is the development of a high-accuracy network for age estimation application, I was contributing to the original idea of the paper, preparing simulations, and writing the papers.

XV In Publications XVIII, I developed and validated the design of a novel logic style to enhance hardware reliability at the logic level, particularly focusing on cryptographic algorithm implementations. As the primary author, I played a pivotal role in conceptualizing the novel logic style, designing experimental frameworks, conducting simulations, analyzing results, and documenting findings.

Abbreviations

AI	Artificial Intelligence
ASIC	Application-specific Integrated Circuit
AxC	Approximate Computing
BER	Bit Error Rate
CMOS	Complementary Metal Oxide Semiconductor
DCIS	Design of Circuits and Integrated Systems
DSE	Design Space Exploration
DHA	DNN Hardware Accelerator
DNN	Deep Neural Networks
FE	Fault Emulation
FOM	Figure-of-Merit
FPGA	Field-Programmable Gate Array
FF	Flip-Flop
FI	Fault Injection
FIT	Failure In Time
FS	Fault Simulation
HDL	Hardware Description Language
LOD	Leading One Detection
LSTM	Long Short-Term Memory
MAC	Multiply and Accumulate
ML	Machine Learning
MNIST	Modified National Institute of Standards and Technology
PE	Processing Element
PAP	Power-Area-Performance
RTL	Register-Transfer Level
QNN	Quantized Neural Network
SA	Systolic Array
SDC	Silent Data Corruption
SER	Soft Error Rate
TSMC	Taiwan Semiconductor Manufacturing Company
URE	Uniform Recurrent Equations

1 Introduction

This PhD thesis addresses the reliability assessment and enhancement of DNNs for safety-critical applications.

This introductory chapter presents the motivation behind this thesis, formulates the area's problems, lists a summary of the main contributions, and sketches the thesis structure.

1.1 Motivation

Deep Neural Networks (DNNs) are nowadays extensively applied to various applications due to their impressive ability to approximate complex functions (e.g., classification and regression tasks) via learning. Since powerful processing systems have evolved in the recent decade, DNNs have emerged to be deeper and more efficient and employed in an ever broader extent of domains. Meanwhile, using DNN Hardware Accelerators (DHAs) in safety-critical applications, including autonomous driving, raises reliability concerns [1], [2]. In compliance with ISO 26262 functional safety standard for road vehicles, the evaluated FIT (Failures In Time) rates of hardware components must be less than 10 (meaning 10 failures in 1 billion hours) to pass the highest reliability level [3] which requires diligent design.

DNNs are deployed in their target application by different DHA platforms, including Field-Programmable Gate Arrays (FPGAs), Application-Specific Integrated Circuits (ASICs), and Graphics Processing Units (GPUs) [4]. Depending on the DHA and the application's environment, different fault types may threaten the component's reliability [5]. Fig. 1 demonstrates different possible fault locations in an AI accelerator and their negative effect on the object detection task. In the example, a pedestrian has been identified as a bird, and a red light is misclassified as a green one leading to a potentially disastrous situation. Faults are originated from hardware, however, they can also be modeled at software platforms for ease of study. Accordingly, the reliability of DNNs is tightly coupled with the reliability of DHAs as faults are coming from hardware. It is worth highlighting that the reliability in this thesis does not relate to the reliability in software engineering or security issues e.g., adversarial attacks.

Throughout the literature, various methods of DNN reliability assessment and enhancement are presented. Some review papers have been published on the topic of DNNs reliability enhancement methods [4], [5], [6], [7], [8], [9]. These works aim to formulate the reliability problem in DNNs, categorize available reliability improvement methods in this domain, and overview the fault injection methods for reliability assessment. Reliability assessment of DNNs is a process for evaluating the reliability of a DNN that is being executed either as a software model or by a hardware platform [IX]. The analysis in [9] reviews the subject of fault tolerance in DNNs and describes different fault models and reliability improvement methods in DNNs. Subsequent works such as [5], [4], [6] provide extensive reviews on the reliability improvement methods for DNNs and characterize taxonomies of different methods. Nevertheless, they do not consider the assessment and

evaluation methods of the reliability of DNNs. Other surveys [7], [8] have reviewed fault injection methods for DNNs reliability assessment, with the former work focused merely on fault criticality assessment and the latter included only a few papers in the survey. In [IX], the first Systematic Literature Review (SLR) is presented and dedicated to all methods of reliability assessment of DNNs.

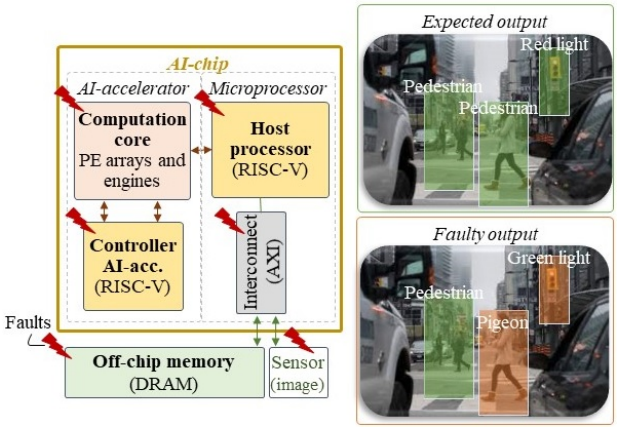


Figure 1: Reliability threats in an example DHA and their possible impact on the output [I].

1.2 Problem Formulation

It has been shown in several studies that the functionality of DNNs in terms of accuracy is remarkably degraded in the presence of faults [10], [11], [12], [13], [14]. As there exist no commonly accepted reliability assessment metrics for DNNs, they require more study to make them applicable for safety-critical applications. Existing state-of-the-art fault-tolerant solutions rely on redundant DNNs with implementation diversity. However, this solution is both costly and does not contribute to facilitating reliability assessment for the overall system.

Thus, the main problem this thesis addresses is the reliable operation of DNN hardware accelerators under fault conditions. The aim is to develop novel assessment frameworks that accurately evaluate the impact of faults on DNNs and propose enhancement techniques that mitigate these effects while maintaining computational efficiency. Specifically, this thesis answers the following problems:

- Reliability concerns in DNN hardware accelerators: Hardware accelerators like FPGAs and ASICs, commonly used for DNN applications, are prone to faults due to process variations, aging, and environmental factors, which can degrade system reliability.

- Lack of comprehensive fault assessment frameworks: There are insufficient tools and methodologies to comprehensively assess the reliability of DNNs on hardware accelerators across different fault types and abstraction levels.

- High resource cost of current fault-tolerant methods: Existing fault-tolerant techniques, such as hardware redundancy (e.g., Triple Modular Redundancy), are resource-intensive and not scalable for low-power or embedded DNN applications.

- Absence of standardized reliability metrics: There is no commonly accepted set of metrics for validating DNN hardware accelerators for safety-critical applications, hindering their deployment in domains like autonomous driving and healthcare.

- Under-explored trade-offs between reliability, performance, and energy efficiency: The balance between fault tolerance, hardware performance, and energy consumption in DNN accelerators remains insufficiently studied, leading to inefficient system designs.

- Limited integration of approximation techniques: Approximate computing, which could reduce resource overhead while maintaining reliability, is underutilized in existing fault-tolerant designs for DNN accelerators.

- Challenges in fault injection scalability: Current fault injection techniques for assessing DNN reliability are computationally expensive and do not scale well for large-scale systems, making it difficult to apply them in real-time or high-performance environments.

By tackling these challenges, this research contributes to the broader goal of ensuring the reliable deployment of DNN hardware accelerators in critical real-world applications.

1.3 Research Objectives

Considering the above problems, this PhD thesis aims to address them as follows:

- Develop and introduce innovative techniques and frameworks for assessing and enhancing the reliability of DNNs.

- Conduct an in-depth analysis of various reliability and hardware optimization strategies, such as quantization and approximation, to evaluate their collective impact on DNN accuracy, reliability, and hardware performance.

- Develop and deploy mitigation techniques for DNNs at both the architectural and hardware accelerator levels, including fault-tolerant designs and redundancy methods.

1.4 Contributions

The contributions of this PhD thesis in tackling the problems explained in section 1.2 consist of:

- Providing a comprehensive overview of essential concepts related to DNNs, reliability assessment, and enhancement, including DNN accelerators and optimization techniques for edge AI. [IX]

- Proposing methodologies for assessing the reliability of DNNs at different abstraction levels, ranging from hardware-accurate, simulation-based fault injection approaches to emulation-based methodologies on actual hardware. [I,IV,V,VI,VIII,X,XI,XII,VI]

- Introducing various mitigation techniques for DNNs based on the reliability assessment results. These techniques, categorized into DNN architecture and DNN hardware accelerator levels, include fault-tolerant designs and algorithmic-level redundancy methods. [I, II, III, VII, X, X]

- Summarizing the key findings of the conducted research and suggesting future directions for applying the studied techniques to other accelerators and further research.

1.5 Thesis Organization

This thesis consists of six main chapters. Each chapter is a novel contribution to this thesis, and the corresponding details of the chapters after the introduction are as follows.

- **Chapter 2** provides a comprehensive overview of the essential concepts required to understand the topics discussed in subsequent chapters. It begins with an introduction to DNNs, followed by key terminologies related to reliability assessment and enhancement. The chapter also introduces various DNN accelerators used for deploying DNNs on hardware. Additionally, it covers Approximate Computing and Quantization as two widely adopted optimization techniques for edge AI.

- **Chapter 3** presents a combination of various proposed methods for assessing the reliability of DNNs like Fault Simulation (FS), Fault Emulation (FE) and Hybrid methods. In general, these methods range from hardware-accurate, simulation-based fault injection approaches to emulation-based methods on actual hardware. The chapter offers a holistic exploration of reliability and hardware optimization techniques, such as quantization and approximation, and assesses their trilateral impact on DNN accuracy, reliability, and hardware performance.

- **Chapter 4** focuses on mitigation techniques suitable for DNNs based on the results from the previous chapter. These techniques are categorized as follows: DNN architecture and DNN hardware accelerator. They include a negative overhead fault-tolerant approximate multiplier for MAC arrays of DNN accelerators and a negative memory overhead technique to store the most critical bits in the same memory element of each critical parameter benefitting from quantization of the DNN model.

- **Chapter 5** concludes the thesis by analysing the key findings and providing future directions. It suggests that the techniques studied in this thesis can be applied to other accelerators and proposes further studies to continue the exploration of the current methods.

At the end, the research papers mentioned in the context of this PhD are attached as appendices.

2 Background

2.1 Deep Neural Networks

Deep Learning (DL) is a sub-domain of Machine Learning (ML) which is the study of making computers learn to solve problems without being directly programmed [15]. Regarding the impressive ability of DNNs in learning, they are applicable in various domains like image and video processing, data mining, robotics, autonomous cars, gaming, etc.

DNNs are inspired by the human brain, and they have two major phases: training and inference. In the training phase, which is an iterative process and performed once, the hyper-parameters (e.g., weights, and biases) of the neural network are updated on a determined dataset. A loss function is adopted in the training phase that measures the difference between the expected and the estimated output of DNN to achieve higher accuracy. Accuracy expresses the proportion of the DNN outputs coinciding with the expected output. On the other hand, in the inference phase, representing the DNN deployment, the network is run several times with the parameters obtained during the training phase [15].

DNNs are constructed ofAfter neurons. Each neuron receives some activation inputs and multiplies them by the corresponding weights. Then, it conveys the summation of the weighted activations to its output. A set of neurons builds up a layer that may have other additional functions, e.g., activation function (ReLu, sigmoid, etc.), batch normalization, (max or average) pooling, etc. [15]. Equation (1) represents the function of the *i*-th neuron in layer *l* (denoted as N_i^l) with input activations from the previous layer *l-1* with *n* outputs (denoted as X^{l-1}), where *W* and *b* represent weights and bias, respectively.

$$N_i^l = \phi(\sum_{j=0}^{n} X_j^{l-1} \times W_{ij}^l + b^l) \tag{1}$$

An abstract view of a neuron and a neural network is depicted in Fig. 2. As shown, inputs are fed into the network through the input layer. The middle layers, called hidden layers, determine the depth of the network and conduct the function of the DNN. The output layer is where the network decides. It produces some probabilities of the possible outputs, i.e., output confidence score, and the class with the highest value is the top-ranked output.

DNNs have various architectures each suitable for specific applications. Nevertheless, it is worth mentioning some terms which are used in this paper. Convolutional Neural Networks (CNNs) are extensively used in classification, object detection and semantic segmentation tasks and consist of multiple convolutional (CONV) and fully-connected (FC) layers. CONV layers have a set of two-dimensional (2D) weights, called filters, that extract a specific feature from the input of the layer. A channel is a set of input feature maps (ifmap) that is convolved with filters resulting in the output feature maps (ofmap) [15].

In the research area of CNNs, there are some models of networks that are most frequently used. For instance, LeNet-5 [16], AlexNet [17], GoogLeNet [18], VGG [19], and ResNet [20] are introduced for image classification, and YOLO

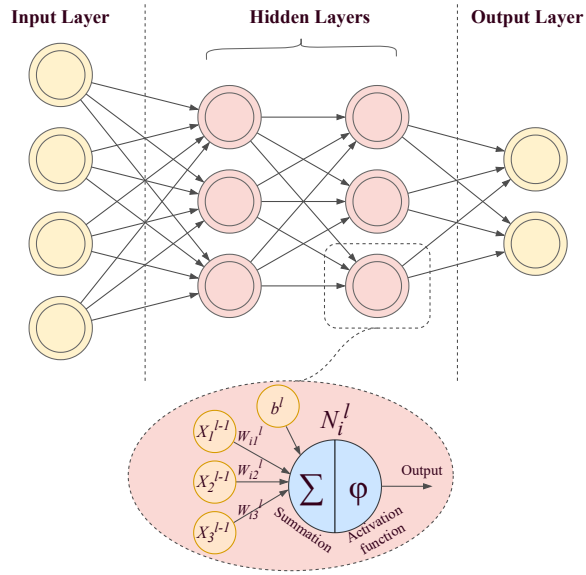


Figure 2: Abstract view of a simple neural network with the detail of a neuron

[21] is designed for object detection. In addition, prominent datasets that are mostly used for training networks on image classification tasks are MNIST [22], CIFAR [23], and ImageNet [24], and on object detection are KITTI [25], and PASCAL VOC [26].

In addition, due to the large number of parameters and calculations in DNNs, Quantized Neural Networks (QNNs) [27] and Binarized Neural Networks (BNNs) [28] are introduced to reduce the complexity, memory usage, and energy consumption of DNNs. These DNNs are the quantized versions of existing DNNs that reduce the bit-width of their parameters and calculations with an acceptable accuracy loss.

2.2 DNN Platforms

2.2.1 Software Frameworks

DNN software frameworks and libraries in high-level programming languages have been developed to ease the process of designing, training, and testing DNNs. These frameworks are widely used due to their high abstraction level of modeling and short design time. Some of well-known software frameworks that are being used for training the DNNs are: TensorFlow [29], Keras [30], PyTorch [31], DarkNet [32], and Tiny-DNN [33]. All these frameworks are capable of using both CPU and GPU to accelerate the training process.

2.2.2 DNN Hardware Accelerators

DHAs are used for the training as well as the inference phase of DNNs. They are called accelerators due to their dedicated design employing parallelism for reducing the execution time of the DNN, either in training or inference. DHAs can

be generally categorized into four classes: FPGAs, ASICs, GPUs, and multi-core processors [34], [35].

According to the literature review of DHAs in [35], FPGAs are used more frequently than other DHA platforms in terms of implementing DNNs, due to their availability and design flexibility for different applications [36]. FPGAs are programmed via their configuration bits that determine the functionality of the FPGA. The system of FPGA-based DNN accelerators usually consists of a host CPU and an FPGA part with corresponding interconnections between them. In this design model, the DNN is implemented on the FPGA part and the CPU controls the accelerator with software, while each part is integrated with memories [36]. A typical structure of an FPGA-based DNN accelerator is depicted in Fig. 3, which is based on HW/SW co-design, that means separating the implementation of DNNs on the integrated CPU (the software) and FPGA (the hardware) that are communicating with one another [37]. High-Level Synthesis (HLS) tools which can synthesize high-level programming languages to RTL are also used for developing FPGA-based DNN accelerators [36].

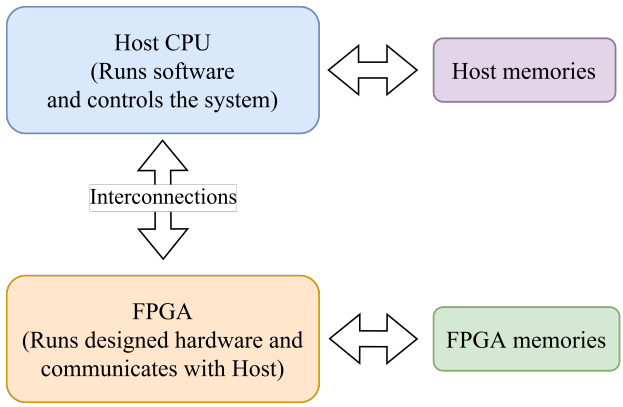


Figure 3: Typical structure of an FPGA-based DNN accelerator [36]

ASIC-based DNN accelerators are more efficient than FPGAs in terms of performance and power consumption but less flexible in terms of applications and require a long design time [38]. There are two general types of architectures for ASIC-based DHA platforms: spatial and temporal [15]. Fig. 4 depicts an example of a spatial architecture model that is constructed of 2D arrays of Processing Elements (PEs) flowing data horizontally and vertically from input/weight buffers to output buffers. PEs perform Multiply-Accumulate (MAC) operations on inputs and weights representing a neuron operation in the DNN. Off-chip memories are required to store the parameters of DNNs and save the intermediate results from PEs. Tensor Processing Unit (TPU) produced by Google, one of the most applicable ASIC-based DNN accelerators, is based on this type of architecture [39].

GPUs are a powerful platform for training and inferring deep networks and are vastly used in safety-critical applications [41]. GPUs include up to thousands of parallel cores, which make them efficient for DNN algorithms, especially in the training phase [38]. GPUs are designed to run several threads of a program and are also exploited to accelerate running DNNs [35]. The general architecture

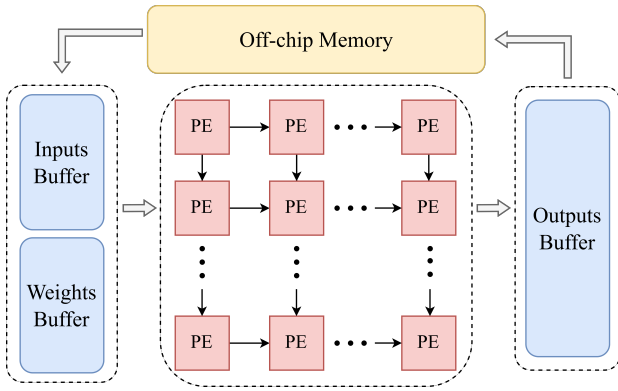


Figure 4: An example of spatial architecture for ASIC-based DNN accelerators [40]

of GPUs is depicted in Fig. 5. There are numerous Streaming Multiprocessors (SMs) in the GPU, each having several cores with a shared register file and caches, while a scheduler and dispatchers control the tasks among and within SMs and cores [42].

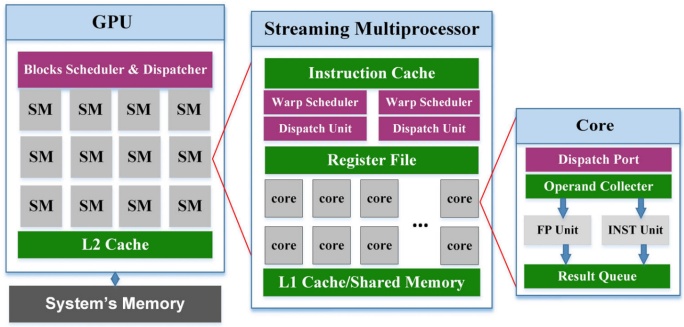


Figure 5: General architecture of CUDA-based GPUs [42]

Multi-core processors, e.g., ARM processors, deploy DNNs mostly for edge processing and Internet of Things (IoT) applications [43], [44], [45]. They facilitate DNNs with parallel computing and low power consumption and provide a wider range of applications for DNNs.

2.3 Reliability, Threats, Fault Models, and Evaluation

Terms of robustness, reliability, and resilience are mostly used in the research pertaining to the reliability of DNNs. These terms are often used interchangeably and ambiguously. In the following, we present the definitions of these three terms as applied in the current thesis:

- **Reliability** concerns DNN accelerators' ability to perform correctly in the presence of faults, which may occur during the deployment caused by

physical effects either from the environment (e.g. soft errors, electromagnetic effects) or from within the device (e.g. manufacturing defects, aging effects, process variations).

- **Robustness** refers to the property of DNNs expressing that the network is able to continue functioning with high integrity despite the alteration of inputs or parameters due to noise or malicious intent.

- **Resilience** is the feature of DNN to tolerate faults in terms of output accuracy.

In this work, we are concerned about the reliability of DNNs, which refers to the ability of accelerators to continue functioning correctly in a specified period of time with the presence of faults. Reliability in this thesis does not relate to the reliability and test in software engineering or security issues e.g., adversarial attacks in which an attacker perturbs the inputs or parameters.

Faults are the sources of threatening the reliability of DNN accelerators (see Fig. 1) that can be caused by several reasons, e.g., soft errors, aging, process variation, etc. [1]. Soft errors are transient faults induced by radiation that are caused by striking charged particles to transistors [46]. Aging is the time-dependent effect of the increasing threshold voltage of transistors due to physical phenomena that will lead to timing errors and permanent faults [47]. Process variations are alterations of transistor's attributes in the process of chip fabrication. As a consequence, voltage scaling may result in faults at the outputs of transistors during their operation [48].

Faults as reliability threats are generally modeled as *permanent* and *transient* faults [6], [5], [9]. Permanent faults result from process variations, manufacturing defects, aging, etc., and they stay constant and stable during the run-time. On the other hand, transient faults are caused by soft errors, electromagnetic effects, voltage and temperature variations, etc., and they show up for a short period of time. Nevertheless, once a faulty value from a component is read by another component and the propagated value does not coincide with the expected one, an *error* happens. Therefore, a fault is an erroneous state of hardware or software, and an error is a manifestation of it at the output. *Failure* or system malfunction is the corruption or abnormal operation of the system which is caused by errors [9], [49], [50].

Faults may have different impacts on the output of DNNs and can be classified based on their effects. A fault may be masked or corrected if detected or result in different outputs compared to the fault-free execution (golden model), in which case the fault is propagated and observed at the output. Faults observed at the output of the system can be classified in two categories: Silent Data Corruption (SDC) and Detected Unrecoverable Errors (DUE), depending on whether a fault is undetected (SDC) or detected (DUE) [6], [51]. Fig. 6 illustrates this general fault classification scheme regarding the output of systems adopted from [49].

2.3.1 Reliability Assessment

Is the process in which the target system or platform is modeled or presented, and by means of simulations, experiments, or analysis, the reliability is measured and

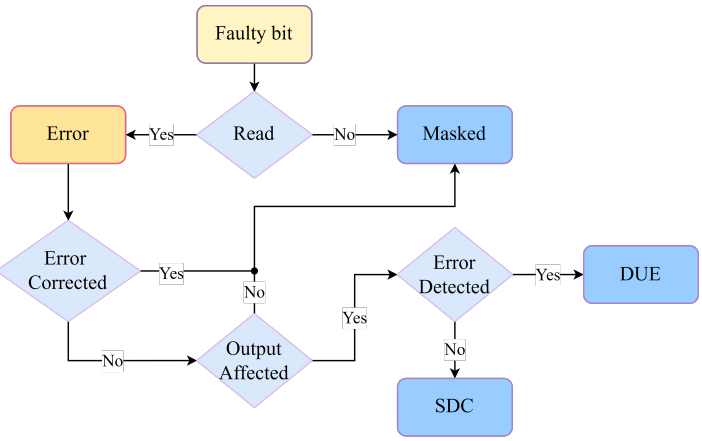


Figure 6: The adopted fault classification based on the output point of view, as in [49]

evaluated. Reliability assessment is a challenging process and several methods can be adopted for modeling and evaluating reliability. In general, evaluating the reliability of a system can be performed by three approaches: Fault Injection (FI) methods, analytical methods, and hybrid methods [52]. FI methods are exploited to inject a model of faults into the system implemented either in software or hardware, while the system is in simulation or being executed. Analytical methods attempt to model the function of the system and its reliability with mathematical equations depending on the target architecture. In hybrid methods, an analytical model is adopted alongside an FI to evaluate the reliability. Generally, FI methods are more realistic than analytical and hybrid methods; however, FI is a time-consuming process with a high computational complexity [53].

In the reliability assessment using FI, it is necessary to determine the target platform, potential fault locations (logic or memory), and the fault type (transient or permanent). Transient faults in logic show up in one clock cycle, while in the memory, they flip a bit that will remain until the end of the execution. Permanent faults are modeled as stuck-at-0 (sa-0), or stuck-at-1 (sa-1), and they exist during the whole execution. According to the selected fault model, perturbation of the model is performed, the system is run, and the outputs are gathered. The output of faulty execution should be compared with the one of the golden-model to measure the impact of faults on the system.

FI allows calculating reliability metrics, e.g., Failures-In-Time (FIT), Architectural Vulnerability Factor (AVF), SDC rate, Soft Error Rate (SER), cross-section, etc. FIT is the number of failures in 10^9 hours, AVF is the probability of fault propagation from a component to other components in a design, SDC rate refers to the ratio of the outputs affected by faults, SER refers to the ratio of soft error occurrence and cross-section is the proportion of observed errors over all collided particles. These quantitative evaluation metrics are usually tightly coupled to each other, yet follow a different purpose to express the reliability of a system.

Exhaustive fault injection into all bits of a platform at every clock cycle requires an extensive simulation. Therefore, to determine how many faults could

be injected into the system in order to be representative statistically, a confidence level with an error margin is presented [54]. It provides a fault rate or Bit Error Rate (BER) for an FI experiment. The number of FI experiments' repetitions regarding the number of possible bit and clock cycle combinations to support the number of injected faults determines the execution space for the FI task.

2.3.2 Reliability Enhancement

refers to the strategies implemented to address faults in DNNs. The literature introduces numerous techniques impacting reliability enhancement across different abstraction levels, from software to hardware. The main categories can be summarized as follows [55]:

- **Model Compression**: This technique aims to reduce the model size, allowing deployment on low-power, resource-constrained devices without significantly compromising accuracy. Recent model compression methods include parameter pruning, quantization, knowledge distillation, low-rank factorization, and compact convolutional filters, among others [IX].
- **Clipped Activations**: Traditional activation functions have unbounded outputs, which can lead to propagation of extreme values when faults occur. By replacing these with bounded activation functions, the output is constrained to a specific threshold, mitigating the impact of faults [IX].
- **Fault Aware Training (FAT)**: This approach treats resiliency as a learning problem, allowing the neural network to learn the effects of faults during the training phase. Faults are introduced into the network during training to enhance its fault tolerance [56].
- **Fault Aware Pruning and Fault Aware Pruning + Retraining (FAP and FAP+T)**: To address faults in DNN accelerators like the Google TPU, this method uses two fault-tolerant techniques: Fault Aware Pruning (FAP) and Fault Aware Pruning + Retraining (FAP+T). Pruning eliminates unimportant connections, thereby bypassing faulty MACs. FAP+T includes an additional retraining step to recover accuracy loss due to the pruned MAC units, allowing the model to adapt to changes and maintain its baseline classification accuracy [57].
- **Selective Hardening**: Traditional hardware redundancy methods, such as Dual Modular Redundancy (DMR) and Triple Modular Redundancy (TMR), involve full hardware replication. TMR, commonly used in industry, provides the required resiliency but at the cost of increased power consumption and significant area overhead. Alternative methods to full TMR have been developed for various safety-critical applications to mitigate these costs [55].
- **Ensemble Learning Based Robustness**: Initially proposed to reduce overfitting and improve generalization, ensemble learning trains multiple models on the same dataset and combines their outputs to determine the best prediction during inference. This method enhances reliability by averaging the predictions of individual models [58].

- **Knowledge Distillation Based Redundancy**: This training-based solution reduces model size by transferring knowledge from a larger, more complex teacher model to a simpler student model. This approach uses a task DNN (teacher) and a checker DNN (student), leveraging knowledge distillation and architecture compression to reduce the size of the checker model without significant performance loss [59].

- **Algorithm-based Fault Tolerance (ABFT)**: ABFT is designed to make matrix multiplications, a core operation in neural networks, fault-tolerant. Originally proposed in [60], ABFT can both detect and correct errors, offering an efficient alternative to traditional TMR methods with lower area overhead. This technique extends error correction codes (ECC) to numeric structures like vectors and matrices.

- **Arithmetic Error Codes**: Falling under the ECC category, arithmetic error codes detect and correct errors during arithmetic operations. These codes have been utilized in various safety-critical applications to enhance system reliability [61].

- **Inter Frame Spatio-Temporal Correlation**: CNNs process each input image (frame) independently, predicting output for each. However, frames and their predictions often correlate. By using both input and output correlation, errors can be detected during processing. If output predictions deviate from expected correlations, it indicates either a valid change in input frames or erroneous predictions. This method helps detect and correct errors by leveraging the temporal and spatial correlations between frames [62].

2.4 Approximate Computing

The most prevalent arithmetic circuits in use are multipliers, adders, and dividers. Dividers are less commonly used compared to multipliers and adders, and adders contribute less to the resource efficiency of DNN hardware accelerators relative to multipliers [63]. Consequently, this section focuses solely on approximate multipliers.

Since multiplication consumes more resources and power than addition, the corresponding works presented in this thesis primarily concentrate on the design of approximate multipliers. It is important to note that approximate MAC units are typically constructed using approximate multipliers; therefore, they are not discussed separately in this section.

Approximate multipliers are categorized into two main groups: (1) conventional approximate multipliers and (2) logarithmic approximate multipliers.

2.4.1 Conventional Multipliers

Conventional approximate multipliers can be classified into two main types: (1) deliberately designed approximate multipliers and (2) CGP-based approximate multipliers. These two categories are detailed below.

Deliberately-designed approximate multipliers:

Deliberately designed approximate multipliers are created by intentionally simplifying the truth table of the exact multiplier. Generally, there are three methods for generating approximate multipliers [63, 64]: (1) approximation in generating the partial products, such as the udm [65]; (2) approximation in the partial product tree, such as the bam [66] and the etm [67]; and (3) approximation in the accumulation of the partial products, such as the icm [68], the acm [69], the approximate multiplier (AM) [70], and the tam [71].

The design of deliberately designed approximate multipliers is reviewed here.

The **under-designed multiplier (UDM)** [65] is created using an approximate 2×2 multiplier. This multiplier produces "111_2" instead of "1001_2" when both inputs are "11_2" to save one output bit.

The **broken-array multiplier (BAM)** [66] eliminates carry-save adders for the lsb in an array multiplier, both horizontally and vertically. This effectively truncates the LSBs, allowing the use of a smaller multiplier for the remaining bits.

The **error tolerant multiplier (ETM)** [67] splits the inputs into LSB and msb sections, which may have different widths. For the LSB part, each bit position is checked from left to right, and if either operand is '1', all remaining bits from that point are set to '1'. Normal multiplication is performed for the MSB part.

The **imprecise compressor multiplier (ICM)** [68] utilizes an approximate (4:2) counter to build approximate multipliers, starting with a 4-bit multiplier to construct larger ones.

The **approximate compressor-based multiplier (ACM)** [69] is built using approximate 4:2 compressors. The two proposed compressors (AC1 and AC2) are implemented in a Dadda multiplier with four different schemes.

The **approximate multiplier (AM)** [70] employs a novel approximate adder that produces a sum bit and an error bit. The error is mitigated using the error bits. The truncated version of this multiplier is known as the TAM [71].

From these primary designs, variants are derived by modifying the configurable parameters in each design, creating a set of 100 deliberately-designed approximate multipliers. For instance, different designs of BAM can be obtained by removing various carry-save adders, and the width of the MSB and LSB parts in the ETM can be adjusted to produce different multipliers.

CGP-based approximate multipliers:

In contrast to the manually designed approximate multipliers, CGP-based designs are automatically generated using Cartesian Genetic Programming [72]. Although various heuristic methods exist for approximating digital circuits, CGP is used here due to its intrinsic multi-objective nature and its success in generating other high-quality approximate circuits [73].

A candidate circuit in CGP is modeled as a two-dimensional array of programmable nodes, each representing a 2-input Boolean function such as AND, OR, XOR, etc. The initial population of CGP circuits includes designs of exact multipliers and circuits generated by mutating accurate designs. Single mutations, by randomly altering the gate function, gate input connections, and/or primary output connections, are used to create more candidate solutions. Further details are provided in [72] and [73].

2.4.2 Logarithmic Multipliers

Let $Z = Z_n Z_{n-1} ... Z_1 Z_0$ be the n-bit binary representation of a positive integer N. Without loss of generality, let Z_k, where $k \leq n$, be the most significant '1':in Z. Hence, N can be represented as:

$$N = 2^k(1+x), \tag{2}$$

where : $0 \leq x < 1$.

Let A and B be the multiplicand and the multiplier, respectively. Following (2), once the base-2 logarithms of input operands A and B are calculated as:

$$\log_2 A = k_1 + \log_2(1+x_1), \tag{3}$$

$$\log_2 B = k_2 + \log_2(1+x_2), \tag{4}$$

their product can be obtained by:

$$A \times B = 2^{k_1+k_2}(1+x_1)(1+x_2). \tag{5}$$

Depending on the computation process, different values for $log_2 A$ and $log_2 B$ and, consequently, different approximate products can be obtained. For example, the Mitchell algorithm uses the following approximation [74]:

$$A \times B \approx \begin{cases} 2^{k_1+k_2}(1+x_1+x_2), & x_1+x_2 < 1, \ 2^{k_1+k_2+1}(x_1+x_2), \\ x_1+x_2 \geq 1. \end{cases} \tag{6}$$

It was found in [75] that the average error for given k_1, k_2, $x_1 \in [0,1)$, and $x_2 \in [0,1)$ for the Mitchell algorithm can be expressed as:

$$E_A = -0.08333 \times 2^{k_1+k_2}. \tag{7}$$

Hence, an error correction term c can be added to the Mitchell algorithm to reduce the average error [75]:

$$A \times B \approx \begin{cases} 2^{k_1+k_2}(1+x_1+x_2+c), & x_1+x_2 < 1, \ 2^{k_1+k_2+1}(x_1+x_2+\frac{c}{2}), \\ x_1+x_2 \geq 1. \end{cases} \tag{8}$$

However, this modified technique increases the area and power consumption compared to the Mitchell algorithm [75].

The approximate LM in [76] uses a so-called soa (ALM-SOA). The set-one-adder (SOA) with k approximation bits (SOA-k) places '1' on the k LSBs, thus overestimating the actual product. Since the Mitchell multiplier tends to underestimate the actual product, the SOA compensates for this accuracy loss. This technique is used in [76] to enhance the accuracy of the Mitchell multiplier with minimal hardware cost.

A low-power version of the Mitchell multiplier is suggested in [77]. As an extended work, a parameter w is introduced in [78] for a customizable LM that considers only the most significant w bits of the operands. Subsequently, truncation is performed after the approximate logarithms of the operands are

calculated using the Mitchell algorithm. This differs from truncating the input operands before computing their logarithm, making this multiplier more hardware-efficient but less accurate than the Mitchell multiplier in terms of both mean and worst-case errors.

Mitchell's method can have relatively large approximation errors depending on the application [79]. To improve accuracy, several Mitchell-based multipliers have been proposed. These usually divide the power-of-two intervals into multiple regions and apply piecewise linear approximation within each region. Different designs vary in the number of regions and the piecewise linear approximation functions used in each region [80].

3 Reliability Assessment of DNN Hardware Accelerators

In this chapter, we explain the reliability assessment methods and frameworks that were introduced in [I, IV, V, VI, VIII, X, XI, and XII].

3.1 Introduction

Reliability assessment of DNNs, are categorized into three main methods: Fault Injection, Analytical, and Hybrid.

3.1.1 Fault Injection Methods

The works based on FI evaluate the reliability of DNNs by fault injection campaign. There exist several taxonomies for the fault injection approaches in the hardware reliability domain [7], [52], [53], [81], [82].In general, FI methods are categorized into three approaches of fault injection as follows:

- **Fault Simulation (FS):** DNNs are implemented either in software by high-level programming languages or Hardware Description Languages (HDL) and faults are injected into the model of the DNN. In the former case, some works consider a DHA model in their software implementations while others do not. We divide works on this approach into hardware-independent, hardware-aware, and RTL model platforms. RTL models represent ASIC-based DHAs.
- **Fault Emulation in Hardware (FE):** Research works on this approach implement and run DNNs on a DHA (i.e., FPGA, GPU, or processor) and inject the faults into the components of the accelerator by a software function, FI framework, etc.
- **Irradiation:** DNN is implemented on a DHA (i.e., FPGA, GPU, or TPU) placed under an irradiating facility to inject beams onto it.

Most of the works on DNNs' reliability assessment use FI methods [IX].

3.1.2 Analytical Methods

Works relying on an analytical method for estimating DNNs' reliability attempt to determine how parameters and neurons of a DNN affect the output based on the connections of neurons and layers. Therefore, they analyze the structure of DNNs and provide a model for the impact of faults on the outputs to find more critical and sensitive components in the DNN. Hence, they can evaluate the reliability of DNNs by means of vulnerability analysis derived by analyses, and eliminate the complexity of simulating/emulating the faults in reliability assessment.

3.1.3 Hybrid Methods

Both, fault injection and analytical methods are used in this category of works to take advantage of both. In this regard, analytical methods can provide some mathematical models in addition to a straight-forward fault injection into the

system for reliability evaluation, so that metrics of reliability evaluation can be obtained with less complexity than extensive FI experiments and more realistic than analytical methods.

In this thesis, we categorized the publications presented in this chapter into two main groups: Fault injection and Hybrid methods. In FI based category, the subsections are grouped under Hardware-Aware Fault Simulation and FPGA-based Fault Emulation methods. The works will be presented accordingly.

3.2 FS method 1: Fault Injection-based Reliability Assessment Framework in Quantized DNN Accelerators

This section is based on the following paper: [I]

3.2.1 Introduction

The reliability of DNN accelerators expresses their ability to produce correct outputs in the presence of hardware faults originating from various phenomena, e.g., radiation-induced soft errors in memory or logic [XI]. DNNs are known to be inherently fault-resilient due to the high number of learning process iterations and several parallel neurons with multiple computation units. Nevertheless, faults may impact the output accuracy of DNNs drastically [XVII], and in the case of resource-constrained critical applications, the reliability of DNNs is required to be evaluated and guaranteed [83]. The complexity of such evaluation motivates an *automated toolchain* with quantization and reliability analysis to support *Design Space Exploration (DSE)* for DNN accelerators already at the early design stage, i.e. starting from a high-level description, followed by providing an FPGA prototype for the selected design.

While the protection of weights stored in ROM can be ensured through error correction codes (ECC) or similar protection techniques, the dynamic nature of activations, which are stored for a short period of time in usually unprotected memories, poses a critical concern. Thus, it is crucial to thoroughly investigate the consequences of faults in the network's activations.

This work presents a framework containing a fully automated toolchain to perform a study on the impact of quantization on network accuracy, hardware performance, and reliability drop in the presence of activation faults (Fig. 1) in systolic-array-based FPGA accelerators. To the best of our knowledge, this is the first framework that holistically considers those parameters. A novel lightweight mitigation technique is proposed and integrated into the framework to study potential trade-offs of compensating the reliability drops.

The proposed methodology enables the analysis both at the level of the network model and at the level of individual layers of the network.

This framework is empowered by techniques for quantizing the networks and restricting the activation ranges to be limited to a certain level throughout the whole network execution by applying an extra scaling function in the network inference. This framework uses the high-level description of a DNN as an input and is capable of providing a transient-fault-resilient systolic-array-based FPGA implementation of the network utilizing the design parameters selected by the DSE. The main contributions in this work are as follows:

- A methodology for holistic exploration of quantization and reliability trade-offs in systolic-array implementation that enables assessing the trilateral impact of quantization on accuracy, activation fault reliability, and hardware performance.
- A fully-automated framework that is capable of applying quantization-aware training, post-training quantization, range-restriction, fault simulation, and implementing the whole methodology down to hardware implementation to measure actual hardware parameters like area, latency, etc.
- A lightweight and effective protection technique is developed and adopted in the framework toolchain to provide the final reliable systolic-array-based FPGA implementation of the network
- Demonstration and analysis of the results on the impact of quantization on reliability, hardware performance, and accuracy of the neural networks due to the transient faults in the activations for two well-known benchmarks.

The rest of this subsection is organized as follows. Related works are discussed in 3.2.2, the methodology and framework are presented in 3.2.3, the experimental setup and results are provided in 3.2.4, and finally, the work is concluded in 3.2.5.

3.2.2 Related Works

The related works are categorized in this subsection. **3.2.2.1 DNN reliability and quantization studies**

Several works examine the impact of different fault models on the basis of a number of layers in DNNs and different data types [84]. Investigation into the effects of data precision is done in [85], where authors conducted a comparison of the resilience of FP16, FP32, and FP64 in the context of Matrix Multiplication. Their findings indicated that the reduction of precision not only enhances GPU performance and efficiency but also contributes to its overall resilience.

Another study [86] involved the deployment of MNIST Convolutional Neural Networks (CNNs) on FPGAs utilizing FP32, FP16. The results of the experiment demonstrated that decreasing the data precision in CNNs can lead to a substantial enhancement in overall resilience. This improvement was attributed to the reduced memory usage. Furthermore, [87] noted that the application of binary quantization to weights in convolutional layers results in decreased vulnerability factors, although it does increase the criticality of faults. [88] showed that the impact of faults is higher in most significant bits (MSBs) and, with more aggressive compression, the most significant bits are more probable to be exposed to faults. The aforementioned works show that quantization from higher data representations like FP32 down to INT16 has a positive impact on the performance and overall resilience, though *on the lower quantization ranks this matter should be studied and is not always impacting positively on the resilience*

In [89], it is shown that in some cases, the impact of the faults in the weight memories of a DNN can be negligible. Even though in the above-mentioned works, *impact of faults (soft errors modeled as bit flips) in the weights of a DNN during inference is examined*, to further enhance our comprehension of the impact of quantization on the reliability of DNNs in systolic-array-based DNN

accelerators, this work is enriched with an FI engine capable of injecting faults into the activations of the DNNs in the systolic architecture.

3.2.2.2 Fault mitigation techniques The process of quantization and outlier regularization offers the potential to restrict the numerical range within a DNN, thereby eliminating the possibility of generating excessively large values due to faults [90], [XVIII].

Hoang et al. analyzed how various boundary values affect the network's accuracy. They have found that the best boundary values for each layer are not necessarily the maximum values of the layers' activations [91]. Hence, they propose an interval search algorithm to find appropriate boundary values for the ReLU activation function at each layer, named FT-ClipAct. The proposed clipped activation function maps their outputs to 0 if activations exceed the boundaries. Although these methods can decrease the effect of faults in DNNs, they remove a significant portion of non-zero activations by replacing them with zero, leading to an accuracy drop in high error rates. It is also noteworthy that the mentioned methods do not consider low integer quantization and are mostly working with FP32 and FP16. In this work, we introduce a novel *lightweight range-checking* circuit that, despite the other works, can consider the maximum values of the layers' activations and replace the out-ranged values with either lower- or upper-bound to avoid fault propagation and also avoid removing a significant portion of non-zero activations by replacing them with only zero. This protection technique is employed in the DNN accelerator hardware generation step of the framework to provide the user with a prototype of the reliable accelerator.

3.2.2.3 DNN hardware accelerator frameworks

The advantages of implementing and deploying DNNs on FPGAs are advocated in several recent works. The existing FPGA-based toolchains to map CNNs are presented in the surveys [36, 92–94]. The FINN framework [95] is released by Xilinx for the exploration of quantized CNNs' inference on FPGAs that also provide customized data-flow architectures for each network.

Heterogeneous systems are another design strategy in the automated toolchains that propose hardware/software co-design [96–98]. In these designs, computational units, e.g., adders and multipliers, are mainly implemented on Programmable Logic (PL) that is controlled by a control unit in a CPU using a dedicated framework, e.g., OpenCL [99]. In this work, we introduce a hardware generation step as part of the framework, to explore DNN inference on an FPGA-based accelerator with a customizable systolic array. It seamlessly integrates with the PYNQ framework [100], leveraging the original PYNQ bootable image. This integration enhances versatility and compatibility, enabling users to implement their network on different FPGA devices supporting PYNQ. Furthermore, the reconfigurable systolic array implementation introduced in this step provides flexibility and scalability. Users can customize this step to meet their specific network requirements by providing trained parameters and network architectures, resulting in efficient and high-performance DNN inference.

To the best of our knowledge, none of the previous works explored the impact of using different levels of full quantization (weights, activations and biases) of a DNN in the presence of transient faults in the activations on the reliability, accuracy, and delay/resource utilization of the target DNN accelerator.

The approach proposed in this work goes beyond the state of the art by establishing a fully automated tool for enabling efficient quantization in FPGA-based DNN accelerators aimed at safety-critical applications. The proposed framework contains a high-level simulator to study the impact of quantization on the reliability and accuracy of the network by considering the hardware architecture, with and without protection techniques, followed by an efficient and user-friendly heterogeneous FPGA implementation of the selected DNN configuration.

3.2.3 Proposed Methodology

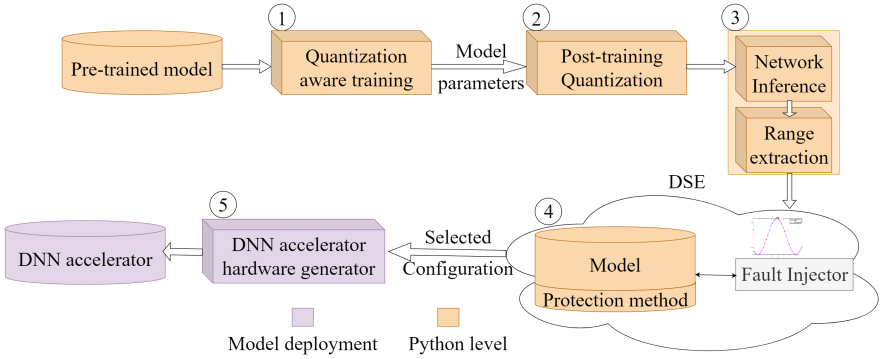


Figure 7: DeepAxe proposed methodology flow

Fig. 7 illustrates the methodology flow established in the toolchain for reliability and hardware performance analysis of quantized DNN hardware accelerators. This framework takes the DNNs' *Pre-trained model* description as the input. The design, training, and testing of the DNNs are performed in Python. *Quantization-aware training* and the *Post-training quantization*, *Range extraction* and *DSE* steps are seamlessly integrated into the same environment and are responsible for extracting the required data for the hardware generation step. This step is responsible for the hardware implementation of the selected configuration to measure actual hardware parameters like area, latency, etc.

3.2.3.1: Quantization-aware training. For this purpose, a full quantization is implemented, targeting all activations, weights, and biases. The framework first takes the description of the network provided by the user and then uses the TFlite library for quantization-aware training. The user can replace their preferred quantization library with the toolchain for this step. The main output of this step is the quantized network's parameters (weights and biases) and network architecture.

3.2.3.2: Post-training quantization. In the post-training quantization step, the user can define any further quantization that can be applied to the network with a negligible accuracy loss depending on the level of the quantization. This framework supports quantizing the network down to 4-bit INT. The output accuracy of the generated network is also provided at this step and is kept as a baseline for the further steps of the methodology. For this step, the following

algorithm is applied to the network parameters:

The mapping equation is defined as:

$$\tilde{x} = \text{clamp}\left(\left\lfloor \frac{x}{S} \right\rfloor + Z;\, q_{\text{min}}, q_{\text{max}}\right) \tag{9}$$

$$S = \frac{x_{\text{max}} - x_{\text{min}}}{2^b - 1} \tag{10}$$

Where Z is the offset defined as zero-point, x_{max} and x_{min} represent the maximum and the minimum value in the vector. The quantization range $[q_{\text{min}}, q_{\text{max}}]$ is determined by the bit-width. We focus solely on uniform unsigned symmetric quantization, as it is the most commonly employed quantization setup. Hence, q_{min} is equal to 0, and q_{max} is equal to $2^b - 1$, where b denotes the bit-width, determining the number of integer grids.

3.2.3.3: Inference and range extraction. In this step, after running the inference, the ranges of the activations are extracted for evaluation and reliability study. The ranges are extracted based on the set of validation data, and then the framework extracts the next set of ranges for each layer based on the test data and validates the extracted data correspondingly.

3.2.3.4: Design Space Exploration.

3.2.3.4-A: Fault simulation. Reliability analysis relies on a Fault Injection (FI) in a *systolic-array-based simulation* of the network in Python, assuming the single bit-flip faults in the activations. While the multiple-bit fault model is more accurate, it requires a prohibitively large number of fault combinations to be considered. Fortunately, it has been shown that high fault coverage obtained using the single-bit model results in a high fault coverage of multiple-bit faults [101]. Therefore, a vast majority of practical FI and test methods are based on the single-bit fault assumption. However, this framework is capable of applying multiple-bit-flips as a fault model depending on the user demand.

The reliability analysis step applies the accuracy drop comparison of the network-under-test as one of the assessment metrics. In addition, the framework assesses the reliability of the DNN by comparing the output probability vector of the golden run (i.e. the DNN that behaves as expected, without faults) and the faulty run (i.e. the DNN that includes the fault). These metrics involve the SDC (Silent Data Corruption) rate. Specifically, one of the two metrics is "absolute", and the other one is "relative". The SDC rate is defined as the proportion of faults that caused misclassification in comparison with the golden model.

- **SDC-1:** Fault caused a misclassification in the top-ranked output class.
- **SDC-5:** Fault caused the top-ranked element not to exist in the top-5 predicted output classes.
- **SDC-10%:** Fault caused a variation in the output confidence score of the top-ranked output class more than 10% compared to the golden model.

After choosing the preferred quantization in **3.2.3.2**, the designer can go through the systolic-array-based fault injector provided for the reliability evaluation of the Quantized DNN (QDNN). The final design is fed to the next step *hardware*

generator for the DNN hardware accelerator generation and hardware performance evaluation process.

3.2.3.4-B: Fault mitigation. Analyzing the output values of the network's intermediary layers post-training reveals identifiable upper and lower bounds for the neuron's output values. Leveraging this characteristic, we can ensure that any out-of-range outputs are reassigned to the respective upper or lower-bound values. This approach can be effectively implemented using specialized hardware units, as outlined below.

Out-range Error Detection: If the neuron's output value exceeds the predetermined upper or lower bound, it indicates a fault in the neuron's input values. To address this, a comparison is made between the neuron's output value and the two pre-established threshold values. For effective error detection, this work introduces the following strategy.

For each layer, we store two values of upper bound and lower bound as the reference threshold for the out-ranged values. The output of the MAC (Multiply-Accumulate) unit is compared with the threshold values using two subtractors (negative values indicate that the output is beyond the threshold). The result of this comparison defines the final output (Fig. 8). The general overhead of this mitigation technique is two stored values for each layer, and two subtractors to compare the MAC output value with the range threshold values and provide the select signal for the MUX to make the decision.

Three variations of this protection technique were implemented in the software to provide users with insights into the reliability enhancements this framework offers:

1. **Method 3.2-1:** When out-of-range value is detected it is replaced by the lower bound (min value).
2. **Method 3.2-2:** When out-of-range value is detected it is replaced by the upper bound (max value).
3. **Method 3.2-3:** When out-of-range value is detected it is replaced by either lower or upper bound depending on the sign of the MAC output.

This protection technique is designed for easy replacement with any other protection methods (i.e. FT-ClipAct [91]) within this framework toolchain without compromising the overall versatility of the framework.

3.2.3.5: Hardware generation.

At this step, a systolic-array-based QNN accelerator for FPGA SoC is generated based on the parameters of the quantized network provided by **3.2.3.4** to assess hardware utilization and requirements.

The following tasks are executed at this step:

1. Network parameters are analyzed to determine the size of the systolic array, bit precision, and AXI bus bandwidth for data transfer. This analysis takes into account the number of kernels and feature map sizes. The goal is to optimize hardware accelerator performance for the generated network and improve overall efficiency.

2. The board is configured with the PYNQ bootable image. PYNQ provides Python and Jupyter Notebook support to AMD-Xilinx embedded devices. Included Python APIs allow to control both processing system and programmable

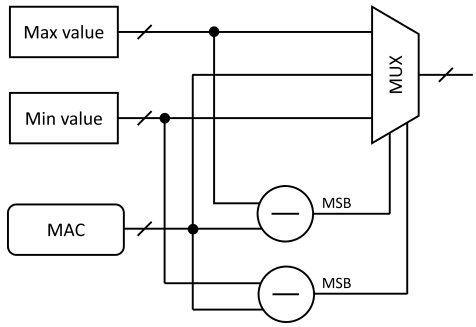


Figure 8: Proposed lightweight mitigation technique

Table 1: Lenet-5 layer-level reports of fault criticality (%) based on FI for different quantized networks

% of critical faults	Unprotected					Protected with Method 1				
Lenet-5	8 bit	7 bit	6 bit	5 bit	4 bit	8 bit	7 bit	6 bit	5 bit	4 bit
conv1	0.31	0.52	1.37	3.27	9.12	0.01	0	0.49	2.82	9.06
conv2	0.29	0.46	1.33	3.62	9.38	0.07	0.08	0.84	3.42	8.49
fc1	1.67	2.03	5.65	14.88	21.15	1.04	0.9	2.14	6.67	11.21
fc2	1.6	2.41	5.88	16.31	25.5	1.24	1.23	1.98	4.79	13.68
% of critical faults	Protected with Method 2					Protected with Method 3				
Lenet-5	8 bit	7 bit	6 bit	5 bit	4 bit	8 bit	7 bit	6 bit	5 bit	4 bit
conv1	0.3	0.6	1.45	1.69	3.76	0	0	0.37	1.46	3.49
conv2	0.21	0.57	1.27	2.45	4.71	0.07	0.08	0.51	1.71	4.08
fc1	1.72	1.78	4.91	9.23	11.13	0.82	1.18	1.82	3.2	4.53
fc2	1.59	2.22	5.94	17.42	19.41	0.97	1.26	2.24	3.09	5.07

logic (FPGA). PYNQ setup was selected to provide the users with a familiar interactive Python environment.

3. Network weights and biases are loaded on the board as NumPy array files. The network is described using a provided Python package that interfaces with the accelerator.

4. FPGA is configured from the Jupyter Notebook with the generated accelerator. Then, inference can be run using the provided input data.

3.2.4 Experimental Results

Experimental results are reported in this subsection as follows: **3.2.4.1 Experimental setup**

Two networks are studied in this work: Lenet-5 and AlexNet. Lenet-5 is trained on the MNIST dataset, and AlexNet is trained on the CIFAR-10 dataset. Both networks are trained according to the **3.2.3.1** methodology using quantization-aware training. Lenet-5 is trained using 16-bit INT data type, AlexNet is trained using 8-bit INT. For the study, different levels of quantization are applied in **3.2.3.2** using post-training quantization.

Simulations are performed on 2 $\times$ Intel Xeon Gold 6148 2.40 GHz (40 cores,

Table 2: AlexNet layer-level reports of fault criticality (%) based on FI for different quantized networks

% of critical faults	Unprotected					Protected with Method 3				
AlexNet	8 bit	7 bit	6 bit	5 bit	4 bit	8 bit	7 bit	6 bit	5 bit	4 bit
conv1	0.5	0.79	1.76	4.03	8.81	0.05	0.06	0.52	1.87	3.56
conv2	0.58	1.05	1.35	1.66	4.11	0.03	0.03	1.035	1.39	3.31
conv3	1.46	1.47	5.11	11.48	23.91	0.07	0.08	1.14	1.29	4.38
conv4	0.99	1.63	2.46	7.13	14.26	0.03	0.04	1.30	4.13	5.17
conv5	0.90	2.10	3.69	7.82	14.31	0.04	0.09	1.61	3.44	5.17
fc1	3.02	4.95	8.15	16.38	31.19	0.14	0.20	1.90	5.11	8.66

Table 3: SDC report for two unprotected Lenet-5 examples with different quantization levels

Metric (%)	16-bit	8-bit
SDC-1	3.18	5.24
SDC-5	28.04	37.26
SDC-10%	14.30	17.65

80 threads per node) with 96 GB RAM. To speed up the simulation process, the framework supports multi-thread parallelism.

To show the hardware characteristics of the output QDNN, studied networks are implemented on the Zynq UltraScale+ ZCU104 Evaluation Board (xczu7ev-ffvc1156-2-e).

3.2.4.2 Fault simulator The fault simulator that is used in **3.2.3.4** calculates the sufficient number of faults required for the reliability analysis. QDNNs generated by **3.2.3.2** are validated by means of fault injection over the test set.

Random fault injection. According to the adopted fault model, a random single bit-flip is injected into a random activation in a random layer of the network, and the whole test set is fed to the network to obtain the accuracy of the network. This process is repeated several times to reach an acceptable confidence level, which depends on the number of neurons and data representation bit length based on [54]. This work provides an equation to reach 95% confidence level and 1% error margin. The framework adopts the formula presented in this work and provides a sufficient number of repetitions required for reliability analysis.

3.2.4.3 Validation results

The accuracy results for the quantized networks are reported in Table 4. Further, fault injection is applied on each network automatically as part of the defined configuration of the framework, and reliability drop and fault criticality are reported in Fig. 9 and Table 1 for the Lenet-5 and in Fig. 10 and Table 2 for AlexNet. *Reliability drop* is defined as the percentage of accuracy loss in the presence of the faults in the activations in a systolic-array-based simulation model of the network. *Fault criticality* is defined as percentages of the faults that show a negative impact on the network accuracy and lead to misclassification. In Fig. 9 and Table 1, the results for all versions of the proposed protection technique are documented for Lenet-5. Table 2, only the network protected with Method 3 is compared with the unprotected network for AlexNet, and in Fig.

Table 4: Model-level design space exploration results for Lenet-5 and AlexNet

Network	BP	GIOPS	Resource utilization			Accu-	Reliability	HW utilization (LUT)			Fault criticality improvement, %		
			LUT	FF	DSP	racy, %	improvement, %	M1	M2	M3	M1	M2	M3
Lenet-5	16	0.058	5298	12,892	9	95.41	—	144	144	576	—	—	—
	8	0.079	3475	7003	9	94.02	64.33	72	72	288	57.78	8.75	65.61
	7	—	—	—	—	93.93	67.95	68	68	135	71.24	14.95	67.74
	6	—	—	—	—	93.52	71.90	63	63	99	57.25	6.16	65.91
	5	—	—	—	—	92.49	81.17	68	68	81	36.30	31.34	66.86
	4	0.087	2114	3865	9	89.65	81.17	36	36	63	25.85	44.92	69.20
AlexNet	16	0.338	16,654	35,503	64	—	—	1024	1024	2048	—	—	—
	8	0.465	12,138	20,539	64	73.03	92.96	512	512	1024	—	—	94.27
	7	—	—	—	—	72.26	89.79	480	480	960	—	—	95.19
	6	—	—	—	—	72.11	73.72	448	448	704	—	—	58.59
	5	—	—	—	—	70.69	66.32	480	480	576	—	—	54.13
	4	0.562	6428	10,067	64	69.15	78.07	256	256	448	—	—	60.08

10 the reports the reliability drop without the protection techniques to show the impact of faults in activations, on different quantization level and layers of an AlexNet network. Fig. 11 shows the reliability drop of different quantized versions of AlexNet and LeNe-5 in the presence of different protection techniques.

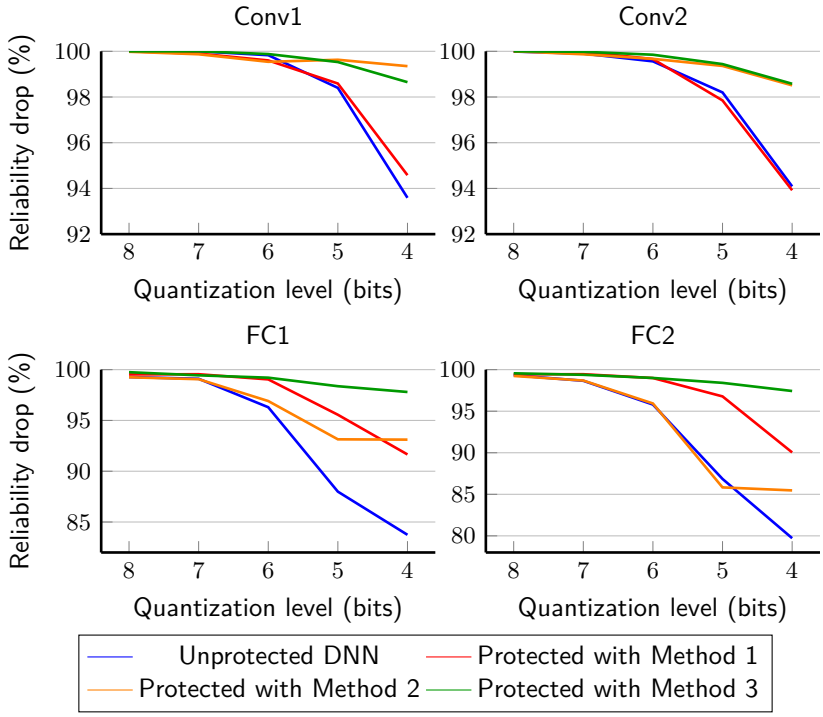


Figure 9: Lenet-5 layer-level reports of reliability drop (based on FI for different quantized networks)

From the previous works [86], it is evident that the reduction in memory size and quantization can lead to enhanced resilience and mitigate the impact of weight faults due to a reduced memory footprint. However, according to the presented charts, quantization may simultaneously heighten the network's vulnerability to faults in activations and logic. This is particularly crucial in lower precision networks, where even minor bit alterations can have significant ramifications. That is why reliability studies in the DNNs should be done for each QDNN to ensure the impact of quantization on the network's reliability.

Fig. 9 shows that protection Method 3 is capable of improving the reliability of the network in the presence of a fault for more than 34.23% in the worst case for Lenet-5. These numbers are calculated based on the following equation:

$$\%\ \text{of Improvement} = \left(\frac{\text{New Value} - \text{Old Value}}{\text{Old Value}}\right) \times 100$$

The same results are reported for AlexNet in Table 4, which shows an improvement

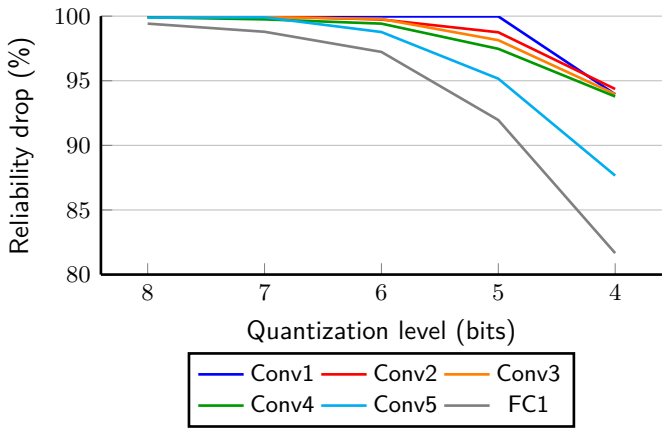


Figure 10: AlexNet layer-level reports of reliability drop (%) based on different quantization levels (unprotected design)

of more than 51.79% in the worst case. Improvements in fault criticality for both networks at the model level are also reported in Table 4, which demonstrates the positive impact of the protection technique on reducing the criticality of faults in both networks. These data also showcase the increasing fault criticality in different networks by increasing the level of quantization. Based on the results reported in Table 4, protection Method 3, which shows the best results for improving reliability among all of the proposed protection techniques, introduces less than 10% overhead compared to the LUTs required for the unprotected network implementation. Meanwhile, full protection of the network with TMR (Triple Module Redundancy) introduces more than 200% hardware overhead.

The fault injection procedure is performed for different quantizations and different versions of the proposed protection technique, and the accuracy drop, due to quantization and fault injection, is profiled. Further, in Table 3, SDC metrics of two examples of quantized Lenet-5 are reported. It can be seen that these two networks are susceptible to injected faults. Specifically, the SDC-10% and SDC-5 are very high: on average, about 3.18% of the time the faulty inference misclassified the input in the 16-bit network and 5.24% in the 8-bit network; furthermore, in 28.04% cases for the 16-bit network and 37.26% cases for the 8-bit network, the expected class is not even in the TOP-5 predictions. In addition, it can be observed that the 16-bit quantized network shows better performance in the presence of faults compared to the 8-bit network. In general, these results show that the DNNs used in this experiment are not suitable for a safety-critical application.

Hardware resource utilization and inference latency in GIOPS (Giga Integer Operations Per Second) for different quantization levels are reported in Table 4 alongside accuracy, reliability improvement due to the quantization, and hardware overhead and fault criticality improvement for fault mitigation techniques. These results of model-level design space exploration are provided for the user to understand the trade-off between reliability, accuracy, and required computational

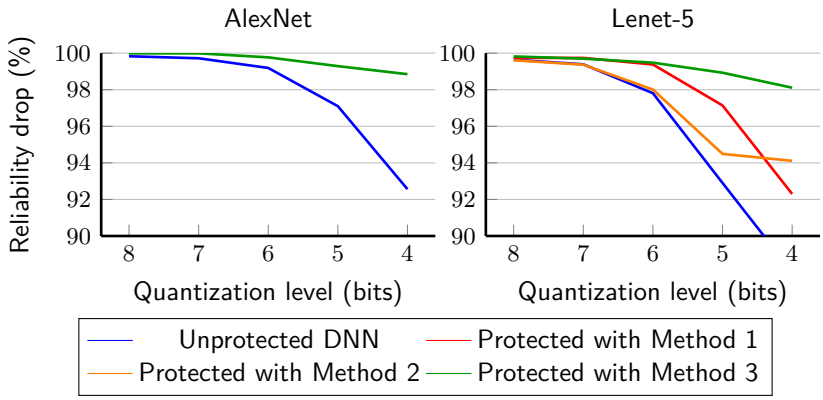


Figure 11: Model-level reports of reliability drop (%) based on different quantization degrees for AlexNet (left) and LeNet-5 (right)

resources.

3.2.5 Conclusion

This subsection presents a comprehensive methodology for exploring and enabling a holistic assessment of the trilateral impact of quantization on model accuracy, activation fault reliability, and hardware efficiency. A fully automated framework is introduced that is capable of applying various quantization techniques, fault injection, and hardware implementation, thus enabling the measurement of crucial hardware parameters like area and latency. Moreover, this subsection proposes a novel lightweight protection technique integrated within the framework to ensure the dependable deployment of the final systolic-array-based FPGA implementation. The experiments on established benchmarks demonstrate the analysis flow and the profound implications of quantization on reliability, hardware performance, and network accuracy, particularly concerning the transient faults in the network's activations.

3.3 FS method 2: SAFFIRA - Software Level Systolic-Array Simulator for Reliability Assessment of DNN Accelerators

This section is based on the following paper: [IV]

3.3.1 Introduction

DNN hardware-accelerator simulation for FI is computationally expensive and typically demands a substantial amount of time to complete a single inference [XI]. This subsection introduces a novel simulation flow and FI tailored to significantly accelerate the injection process on systolic-array-based DNN hardware accelerators. The systolic-array core of the DNN accelerators is modeled using the Uniform Recurrent Equations (URE) system. The proposed injection flow has been implemented as an open-source tool named **SAFFIRA**, which stands for **Systolic Array simulator Framework for Fault Injection based Reliability**

Assessment. Simulation-based FI is usually done either without considering the underlying hardware or through RTL (Register-Transfer Level) simulations known for their resource-intensive computations and time-consuming nature. SAFFIRA is based on a systolic Array (SA) simulator, thus offering the advantage of being more precise than a hardware-agnostic tool, but much faster than traditional RTL-level simulations. Experimental results show a reduction of the fault injection time up to 3× compared to the state-of-the-art hybrid (software/hardware) hardware-aware fault injection frameworks and up to 2000× compared to RT-level fault injection frameworks — without compromising accuracy.

The key contributions of this work are the following:

- introducing a hierarchical **methodology** for the hardware-accurate reliability assessment of SA using a novel simulation-based fault injection approach by modeling the systolic-arrays using Uniform Recurrent Equations (URE) system;
- presenting an **open-source tool** implementing the aforementioned methodology;
- introducing a new **metric** called **faulty distance** for reliability assessments of DNNs;
- evaluating the performance of the framework on state-of-the-art DNN benchmarks

The rest of this subsection is organized as follows. Subsection 3.3.2 presents the related works. Subsection 3.3.3 presents the proposed fault injection flow for SA. Subsection 3.3.4 shows the experimental setup and results. Subsection 3.3.5 concludes the work.

3.3.2 Related Works

This subsection discusses previous works targeting DNN reliability assessment by using simulation-based FI.

3.3.2.1: Hardware-Agnostic FI Tools

Tools in this category perform fault injection without taking into account the underlying hardware. Some of these are capable of performing FI directly in the DNN models. In this category, PyTorchFI [102] and TensorFI [103] can inject faults into DNN models respectively implemented in PyTorch, Tensorflow, and Keras. All of these open-source frameworks can inject both permanent and transient faults into weights as well as activations given specific error rates such that it is possible to evaluate the accuracy loss.

Moreover, to further enhance the efficiency, additional FI tools have been introduced. For example, BinFI [104] is an extension of TensorFI that aims at identifying critical bits in DNN. Another tool, namely LLTFI [105], is able to inject transient faults into specific instructions of DNN models in either PyTorch or TensorFlow.

3.3.2.2: Hardware-Aware FI Tools

These tools can perform FI in software, taking into account the relying hardware using some abstract models of the ‘DNN hardware accelerator.

In [106], the authors used an RTL model of a SA to perform their experiments. Reference [107] maps a DNN into the RTL implementation of the accelerator. They study the effect of transient faults in memory and datapath accurately. In these studies, FI is performed in software while all of its parameters are integrated with the corresponding hardware components. Authors in [108] implemented their DNN and the fault injector in software, inspired by an FPGA-based DNN accelerator. Moreover, in [13], DNN and FI are implemented in Keras, and the architecture of a SA accelerator is considered for a fault-tolerant design. Similarly, authors in [109] evaluate their proposed reliability improvement technique on memories in TensorFlow while injecting transient faults into the weights. PyTorch is used in [110] to implement the DNN, and transient faults are injected into activations (datapath or MAC units) and weights (memory) regarding the SA accelerator model. Reference [111] also uses PyTorch and injects faults by a custom framework called TorchFI to inject faults into the outputs of CONV and FC layers of the network.

The effect of permanent faults at PEs' outputs is studied in [112] where the model of the accelerator is adopted from implementing the DNN in an N2D2 framework [113]. Furthermore, authors in [114] use PyTorch and study permanent faults in MAC units of an accelerator while training to improve the reliability at inference. Authors in [115] developed a Keras-based accelerator simulator to study the effect of permanent faults on the on-chip memory of accelerators by injecting permanent faults into activations and weights. Weight remapping strategy in memory to decrease the effect of permanent faults is evaluated in [116] using Ares. SCALE-Sim [117], a systolic CNN accelerator simulator, is adopted in [118] to study permanent faults in PEs and computing arrays in systolic array-based accelerators.

Similar to the Hardware-Independent platform, faults are injected based on BER, or fault rate, and experiments are repeated to reach 95% confidence level and 1% error margin [13]. In general, the main drawbacks in the existing reliability assessment methods for DNN can be summarized as follows:

- There is no software FI framework in hardware-aware platforms. Hence, there is a potential for DNN accelerator simulators to be exploited or developed for the reliability assessment of DNNs;
- Several FI research works carry out accuracy loss and fault classification as an evaluation of reliability. Also, some works considered FIT (Failure In Time) [84]. However, there is still an urgent need to present DNN-specific metrics for reliability evaluation. In this work, we are introducing a new metric called faulty distance to provide a better understanding of the network resilience.

3.3.3 Proposed Methodology

The proposed methodology for the SAFFIRA framework is illustrated in Fig. 12. After providing the trained network parameters and architecture, in step one, the fault list is generated. Possible fault locations can be defined by the user or can be a random fault list generated based on the network parameters by the framework. Faults can be selected as transient or permanent faults targeting

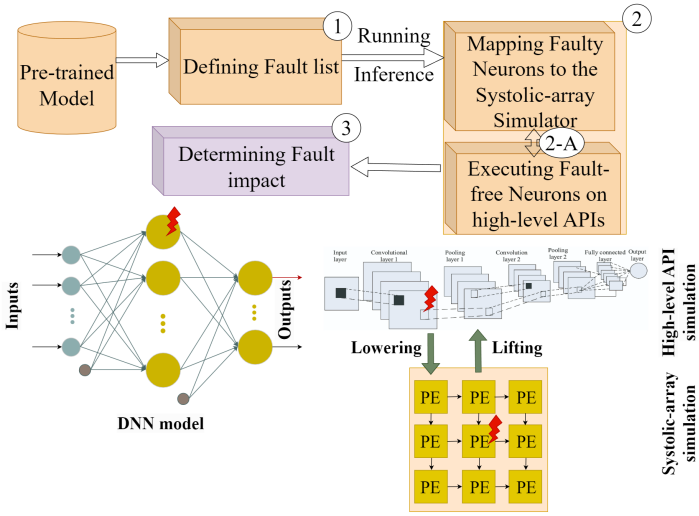


Figure 12: SAFFIRA methodology

different activations of the DNN. Then, in step two, the fault injection campaign is performed at the systolic-array simulation environment in Python, and the rest of the network is executed at the high-level API (e.g. Pytorch) to speed up the process. In this step, switching between high-level API and systolic-array simulator (2-A) is done by a method called LoLif, which is described in below in this subsection. Finally, the reliability of the network and the impact of the faults are reported at step three by different metrics.

SAFFIRA supports various data representations, including fixed-point, integer, and floating-point formats. This framework also supports various relevant mapping to systolic-array architecture scenarios (e.g. output stationary, weight stationary, etc.). These flexibilities allow researchers to adapt the framework to different applications and tailor the reliability assessment to specific hardware requirements.

3.3.3.1 Hardware simulations:

SAFFIRA is a SA model based on the URE system. As described by [119], it is possible to generate a SA that solves the problem described by a URE system. In the case of SAFFIRA, the URE system is the one associated with matrix multiplication since it is the operation deployed on the SA for DNN execution. The following subsection presents the formal details needed to perform a simulation followed by performing fault injection in such a context, and finally, the strategies strictly related to DNN are covered.

3.3.3.1-A Mathematical formalism

A URE system is defined on top of an integer lattice L_n of points p in the n-dimensional Euclidean space E_n. The goal is to solve a system of equations associated with the variables $x_1(p), x_2(p), \cdots, x_m(p)$ for all points $p \in R$, where $R \subseteq L_n$ [119]. This system can be either uni-variate or multi-variate. Here, only uni-variate case is considered, thus the system would have the following form:

$$x_1(p) = f[x_1(p-w_1), \cdots, x_m(p-w_p)], \tag{11}$$
$$x_2(p) = x_2(p-w_2), \tag{12}$$
$$\vdots \tag{13}$$
$$x_m(p) = x_m(p-w_p). \tag{14}$$

The points $p - w_{i_k}$ belong to L_n. The vectors w_k are constants independent of p and this is why they are said to have *uniform dependence*. Each equation $x_i(p)$ depends on the points $p - w_{i_k}$.

The authors of [119] showed a strategy to model a SA starting from the problem to solve. Specifically, the authors explain three steps:

1. find a URE system for the problem to solve,
2. find a timing function compatible with the dependencies of the URE system,
3. find an allocation function to map the URE onto a finite architecture.

The main idea is to project the space E_n twice: the first time, the resulting points will correspond to the spatial arrangement of each PE. The second projection determines iso-temporal planes, identifying operations that are computed during the same clock cycle but on different PE; each plane corresponds to a different clock cycle. The space-projection matrix P and the temporal dimension vector π are used later.

3.3.3.1-B Convolutions The strategy explained above opens the possibility to implement a variety of algorithms as a systolic array. Based on the literature, it is possible to perform a convolution as a matrix multiplication [120]. The experiments shown below are performed using a systolic array to perform the matrix multiplication $C = A \times B$. The associated URE is the following:

$$c(i,j,k) = c(i,j,k-1) + a(i,j-1,k) \times b(i-1,j,k) \tag{15}$$
$$a(i,j,k) = a(i,j-1,k) \tag{16}$$
$$b(i,j,k) = b(i-1,j,k) \tag{17}$$
initial conditions (18)
$$a(i,0,k) = a_{ik}, \quad \forall i,k \tag{19}$$
$$b(0,j,k) = b_{kj}, \quad \forall k,j \tag{20}$$
$$c(i,j,0) = 0, \quad \forall i,j \tag{21}$$
$$\tag{22}$$

where $p = (i,j,k) \in R \subseteq L_n$, in which i, j and k assume values between 1 and $N1$, $N2$, $N3$ respectively. $N1$, $N2$ and $N3$ are problem parameters such that $A \in \mathbb{R}^{N_1,N_3}, B \in \mathbb{R}^{N_3,N_2}, C \in \mathbb{R}^{N_1,N_2}$.

When it comes to performing a convolution, the input matrices must be *reshaped* such that the result of the SA is a convolution. In this work, this concept is called LoLif, which stands for Lowering and Lifting strategies. This idea is explained in [120]. If computing a convolution $C = A * B$ is needed, it can be implemented as a transformation lif of the matrix multiplication of transformed matrices $low_a(A) \times low_b(B)$. In formulas:

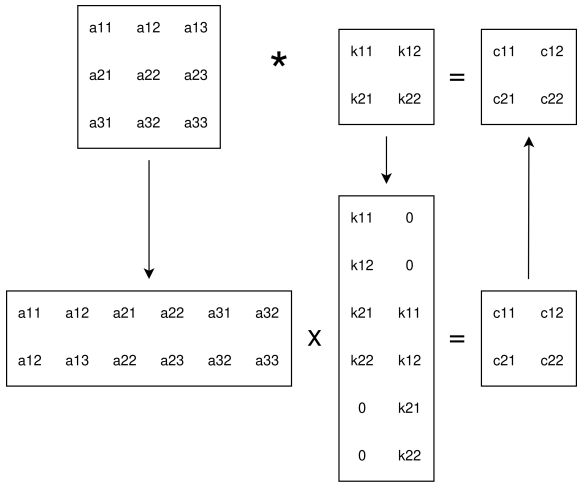


Figure 13: LoLif example. Applied transformations are similar to im2col and im2row.

$$C = lif(low_a(A) \times low_b(B)), \tag{23}$$

where Lif, Low_a and Low_b are corresponding transformations, as shown in the example of Fig. 13

3.3.3.1-C Injections

In order to perform the simulation, it is sufficient to solve the system shown above. Nevertheless, this method gives the possibility of injecting faults in the values in a hardware-aware fashion. To achieve the injection, it is sufficient to change the values $a(p)$, $b(p)$, $c(p)$ for specific points p. The faulty values must then be propagated to the following PE. Given that each point p is projected to the physical space $r = (x, y)$ using the physical space-projection matrix, $r = Pp$, it can be inferred that how the injected values are propagated through the different PE. Specifically, for some dependence vector d for the different labeled variables a, b, c. Looking at the system above, the following can be observed: $d_a = (0,1,0)$, $d_b = (1,0,0)$ and $d_c = (0,0,1)$. Afterwards, the propagation direction can be found using the same relationship shown before: $\delta x_i = Pd_i, i = \{a, b, c\}$. This means that the value of a in some PE in position s will be propagated to the PE in position $s + \delta x_a$. The same reasoning can be done for the time, supposing that a fault is propagated not only in space but also in time. We can compute $\delta t_i = \pi d_i$. For simplicity, $\pi = (1,1,1)$ is fixed to reduce the exploration space. In this case, the time dependency δt_i will always be 1: $\delta t_i = 1$.

Figure 14 shows an example. In this case, an injection in the element $s = (x, y, t)$ on the generic line i is done between times 0 and ∞. The injected elements are visible in the figure. Specifically, the fault will propagate in time, thus injecting also $s + \delta t_i$ and $s + 2\delta t_i$. In the same way, this fault will propagate in space, to the element cascading from s. Note that the value propagation only happens after each clock cycle. This means that the next injected element will be displaced also in time, thus injecting element $s + \delta x_i + \delta t_i$. In the same way, the latter will propagate to the following element on the following clock cycle, thus injecting element $s + 2\delta x_i + 2\delta t_i$ and so on.

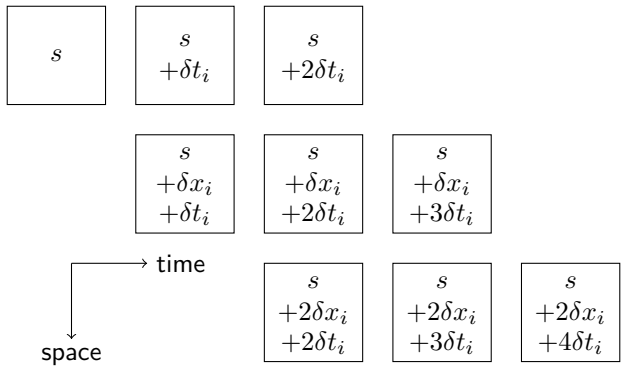


Figure 14: When injecting element s, the fault is propagated in time (thus affecting elements $s+\delta t_i$ and $s+2\delta t_i$) and in space (forwarding the faulty value to neighboring elements $s+\delta x_i+\delta t_i$, $s+2\delta x_i+\delta t_i$ and so on).

The set of points belonging to the injection can be transposed back into the iteration space E_n using the pseudo-inverse P^{-1}. The set of points identified with this strategy will be subject to injection. Formally, injection is as a function h applied to a variable:

$$a(p) = h(a(p - w_a)) \tag{24}$$

3.3.4 Experiments Results

Two different sets of experiments are performed using SAFFIRA. First, a fault injection based on the permanent-fault model is performed on two different quantization versions of the LeNet-5 network (8-bit and 16-bits integers). The second set of experience is performing fault injection based on the transient fault model in the three different benchmarks (AlexNet, VGG-16 and ResNet-18). All networks are fully quantized to INT data type, including all activations, weights, and biases. The base accuracies are reported in the table 5

Table 5: Base accuracy of networks under test

DNN	accuracy (%)
8-bit LeNet-5 (MNIST)	93.8
16-bit LeNet-5 (MNIST)	95.4
AlexNet (CIFAR-10)	78.0
VGG-16 (CIFAR-10)	93.4
ResNet-18 (CIFAR-10)	93.8

The SA model for these experiments is output stationary. This means that its physical-space projection matrix P is as follows:

$$P = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$$

Such a matrix corresponds to a rectangular SA with $N1 \times N2$ PE. Please

note that with this projection, the variable c (i.e. the partial sum) is a *stationary variable* since it is always available on the same PE regardless of the iteration. Whether a variable is stationary or not depends on the employed projection.

In all experiments, fault injection is repeated several times to reach an acceptable confidence level, based on [54]. This work provides an equation to reach 95% confidence level and 1% error margin.

A: Fault Classification:

The DNN resilience is evaluated by comparing the output probability vector of the golden run (i.e. the DNN that behaves as expected, without faults) and the faulty run (i.e. the DNN that includes the fault). The SDC rate is defined as the proportion of faults that caused misclassification in comparison with the golden model [121].

In addition, the targeted hardware reliability can be calculated by differentiating SDC rates of injected transient faults into defined classes and calculating FIT for the accelerator (*accel*) by its components (*comp*) with (25) in which FIT_{raw} is provided by the manufacturer, $Size_{comp}$ is the total number of the component bits, and SDC_{comp} is obtained by FI.

$$FIT_{accel} = \sum_{comp} FIT_{raw} \times Size_{comp} \times SDC_{comp} \tag{25}$$

Finally, **faulty distance** is proposed. This metric can be used to evaluate the resilience of classifications DNN. Supposing the golden probability vector is G, the faulty probability vector is F and the function $ag(\cdot)$ corresponds to the argmax function, then the faulty distance function d_f is defined as follows.

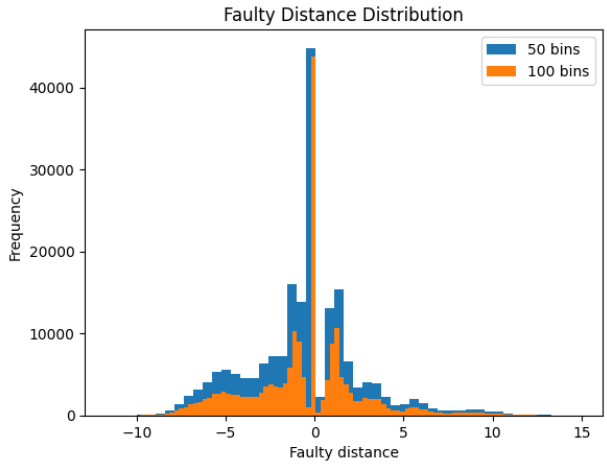
$$d_f = (1 - \frac{G \cdot F}{G \cdot F}) \cdot (ag(F) - ag(G)) \tag{26}$$

In this metric, cosine similarity is being used $cos\theta = \frac{G \cdot F}{G \cdot F}$. Cosine similarity serves as a metric for assessing the resemblance between two non-zero vectors within an inner product space. Representing the cosine of the angle between the vectors, this measure calculates similarity by normalizing their dot product. In our study, we utilize cosine similarity to evaluate the entirety of generated probabilities across various classes in both faulty and golden modes. The cosine similarity metric yields values within the range of -1 to 1. Proximity to 1 signifies a high degree of similarity between vectors. Therefore, the faulty distance metric gives 0 when the faulty output corresponds to the correct classification. The bigger the metric, the worse the misclassification is.

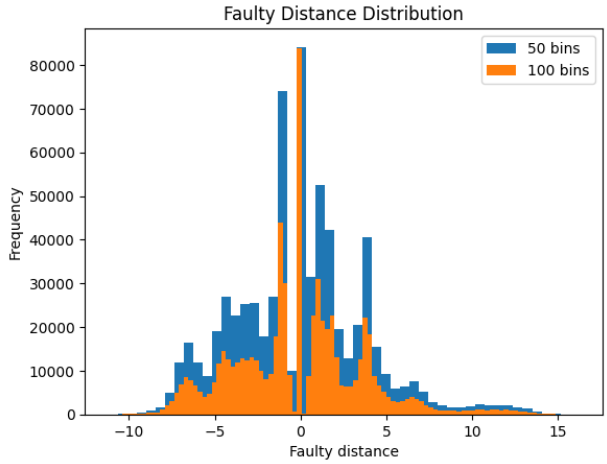
B: Results:

Table 6 shows the results of the FI for permanent fault injection experiments on LeNet-5 with the different metrics. It can be seen that this network was highly susceptible to the injected permanent faults. Specifically, the SDC-1 and SDC-5 are very high: on average, about 82% of the time, the faulty inference misclassified the input; furthermore, about 93.5% of the inputs were completely missed since the correct label was below the fifth position. The SDC-10% and SDC-20% rates are very high as well: more than 95% of the inputs had the correct class with a probability much too low than expected. Average Faulty Distance (AFD) is also reported that shows the 16-bit network in this particular

case, is more reliable compared to the 8-bit network in the presence of permanent faults in the systolic architecture.



(a) Faulty Distance on the 8bit network



(b) Faulty Distance on the 16bit network

Figure 15: Histogram plot of the Faulty distance values

These results show that the DNN used was not usable in a safety-critical environment. This result was expected since the network was not trained to withstand stuck-at faults like the ones injected.

For the second experiment, only SDC and AFD are reported in table 7.

Faulty Distance: In the previous subsection, only the average faulty distance was shown. Nevertheless, this metric can be looked through with more details when plotted as a histogram. Figure 15 shows the histograms (both with 50 and 100 bins) of the metrics per each experiment. It is possible to see a peak at 0, which corresponds to all the correctly classified inputs. The height of that

Table 6: FI experiments results on two LeNet-5

Metric	16bit	8bit
SDC-1 (%)	77.84	87.70
SDC-5 (%)	93.05	94.49
FIT (failures/10^9 hours)	4.9e-4	5.0e-4
SDC-10% (%)	98.16	98.53
SDC-20% (%)	96.21	96.97
AFD	-0.04	-0.53

Table 7: Reliability analysis of different state-of-the-art DNN benchmarks

DNN	SDC-1%	SDC-5%	SDC-10%	SDC-20%	AFD
AlexNet (CIFAR-10)	4.3	29.1	13.1	9.7	7.1×10^{-2}
VGG-16 (CIFAR-10)	3.0	40.0	46.5	84.5	1.9×10^{-3}
ResNet-18 (CIFAR-10)	1.5	23.0	16.5	82	1.6×10^{-3}

column is precisely the same as the complement of the SDC-1 metric. On top of that, it is possible to see two different, yet similar, trends for the two networks. Figure 15a shows other three peaks: around $+1$, -1 and -5. This means that, although most of the inputs were mis-classified, the difference with the golden vector was not extremely big, in general. On the other hand, figure 15b shows many more peaks, this means that it is more difficult to predict how a fault will propagate in this case.

Computation Time: The experiments were performed on a server using python3 with an Intel Xeon Silver 4210, with a total number of 40 cores. SAFFIRA completes 500 inferences of two convolutional layers, with the same systolic array, in about 10 minutes with minimal optimization. This means a total of about 16.3 simulations per second. For comparison, by utilizing the framework presented in [122] to perform fault injection on the same networks as this work, on average, 5.8 simulations per second are executed. The mentioned framework is the state-of-the-art hybrid (software/hardware codesign) hardware-aware fault injection framework. Therefore, SAFFIRA provides about 2.8× speed up by performing the same analysis. Also, the same fault injection campaign is performed at the RT level using QuestaSim. The results show 0.007 simulation per second, which is 2100× slower than the proposed method in this work.

3.3.5 Conclusion

This work presents a novel hierarchical fault injection strategy for systolic arrays, addressing the time efficiency issue by introducing a novel hierarchical software-based hardware-aware fault injection strategy tailored for systolic array-based DNN implementations. The approach demonstrates a reduction of the fault injection time up to threefold compared to the state-of-the-art hybrid (software/hardware) hardware-aware fault injection frameworks and more than 2000× compared to RT-level fault injection frameworks — without compromising accuracy. Additionally, we propose and evaluate a new reliability metric through experimental assessment. The performance of the framework is studied on state-of-the-art DNN benchmarks.

3.4 FS method 3: DeepAxe - Approximation and Reliability Trade-offs in Dataflow DNN Accelerators

This section is based on the following paper: [VI]

3.4.1 Introduction

deployment of a DNN accelerator for the safety-and mission-critical applications (e.g., autonomous driving) requires addressing the trade-off between different design parameters of *hardware performance*, e.g., area, power, delay, and *reliability*.

A compromise between conflicting requirements can be achieved by simplifying the implementation to sacrifice the precision of results but benefiting from lower resource utilization, energy consumption, and higher system efficiency. *Approximation Computing (AxC)* is one of such concepts in hardware design [123].

Moreover, the assessment of the reliability of DNN accelerators is a challenging issue by itself. Reliability of DNNs concerns DNN accelerators' ability to execute correctly in the presence of faults [4] originating from either the environment (e.g., soft errors, electromagnetic effects, temperature variations) or from inside of the chip (e.g., manufacturing defects, process variations, aging effects) [5].

The ability to tolerate the impact of faults on the output accuracy is called *fault resiliency* and, in practice, it is one of the contributors to the DNN accelerators' reliability [124]. DNNs are known to be inherently fault-resilient due to the high number of learning process iterations and also several parallel neurons with multiple computation units.

Nevertheless, faults may impact the output accuracy of DNNs drastically [125], and in case of resource-constrained critical applications, DNNs' fault resiliency is required to be evaluated and guaranteed [126], [83].

The complexity of such evaluation motivates an *automated tool-chain* with AxC and resiliency analysis to support *Design Space Exploration (DSE)* for DNN accelerators already at the early design stage, i.e. starting from a high-level description.

High-Level Synthesis (HLS) tools bridge high-level programming and hardware implementation and allow overcoming the complexity of the process and reducing the design time. Recently, DNN-tailored HLS tools were proposed, e.g., CNN2gate [127], fpgaConvNet [127] and DeepHLS [128]. Such tools are capable of providing a synthesizable C implementation of DNNs for FPGAs from a high-level description in a language such as e.g., Keras.

This subsection presents a novel framework and a fully automated tool-chain DeepAxe to provide a design space exploration for FPGA-based implementation of DNN accelerators by analyzing approximation and soft-error reliability trade-offs. To the best of our knowledge, this is the first framework that holistically considers both the transient fault resiliency and hardware performance of DNN accelerators as design parameters. DeepAxe is empowered by techniques for quantizing the networks and providing the capability of substituting the exact computing (ExC) units of the network with AxC units and identifying the optimal design points for selective approximation.

DeepAxe uses the Keras description of a DNN as the input and is capable of providing an FPGA-ready approximated and transient-fault-resilient inference implementation of the network based on the design parameters selected based

on the DSE results. The main contributions in this work are as follows:

- A methodology for selective approximation of reliability-critical DNNs providing a set of Pareto-optimal DNN implementation design space points for the target resource utilization requirements.
- A framework DeepAxe for holistic exploration of approximation and reliability trade-offs in DNN accelerator FPGA-based implementation that enables assessing the trilateral impact of approximation on accuracy, reliability, and hardware performance.
- Integration of the fully automated DeepAxe tool-chain into the DeepHLS environment.
- Demonstration and validation of the framework on representative custom and state-of-the-art DNNs and datasets.

The rest of this subsection is organized as follows. Related works are discussed in 3.4.2, the methodology and framework are presented in 3.4.3, the experimental setup and results are provided in 3.4.4, and finally, the work is concluded in 3.4.5.

3.4.2 Related Works

The advantages of implementing and deploying DNNs on FPGAs are advocated in several recent works. The existing FPGA-based tool-chains to map Convolutional Neural Networks (CNNs) are presented in the surveys [36, 92–94]. The FINN framework [95] is released by Xilinx for the exploration of quantized CNNs' inference on FPGAs that also provides customized data-flow architectures for each network. Research works [129] and [130] provide Register-Transfer Level (RTL) models using conventional synthesis tools, e.g., Vivado HLS, where the outputs can be directly synthesized on an FPGA. Heterogeneous systems are also another design strategy in the automated tool-chains that propose hardware-software co-design [97, 98, 130]. In these designs, computational units, e.g., addition, or multiplication, are mainly implemented on Processing Logic (PL) that is controlled by a control unit in a CPU using a dedicated framework, e.g., OpenCL [99].

Using Fixed-point (FxP) data type instead of Floating Point (FP) is becoming more popular due to the lesser resource utilization while keeping the output accuracy degradation at an acceptable level [131–133]. Throughout the literature, comprehensive simulations exist that prove that merely an 8-bit data type for MAC operations in DNN execution is sufficient to provide a practical accuracy along with favorable resource utilization [39, 134]. In this work, we considered 8-bit as the base data type for the simulations and implementations.

A number of works in the literature explore the reliability of the DNNs [135, 136]. Some works examine the impact of different fault models on the basis of a number of layers in DNNs and different data types [84]. Studying the significant impact of transient faults vs permanent faults is also done by [137]. The fault analysis of exact DNNs has drawn a lot of attention in the state-of-the-art research, and only recently, researchers have started to investigate also the reliability of approximated DNN accelerators (AxDNNs) [83]. A somewhat

expected conclusion in [136] is that the error induced by approximation, along with the faults in the DNN structure, are not evenly propagated. The impact of a fault may differ based on different parameters, like fault type, fault location, the approximation error resiliency for each layer, etc. To the best of our knowledge, none of these works explored the impact of using different combinations of approximated layers of a DNN in the presence of transient faults on the reliability, accuracy and delay/resource utilization of the target DNN accelerator.

The approach proposed in this subsection goes beyond the state of the art by establishing a fully automated tool for enabling efficient AxC in FPGA-based DNN accelerators aimed at reliability-critical applications. The proposed DeepAxe framework is integrated into DeepHLS environment [128], which is capable of providing completely synthesizable code for efficient FPGA implementations. In particular, this work extends DeepHLS with fault simulation, resiliency analysis and also the use of AxC. The new features allow providing the designers a guideline to choose optimal configurations based on specific requirements for latency, accuracy, resource utilization, and fault resiliency.

3.4.3 Proposed Methodology

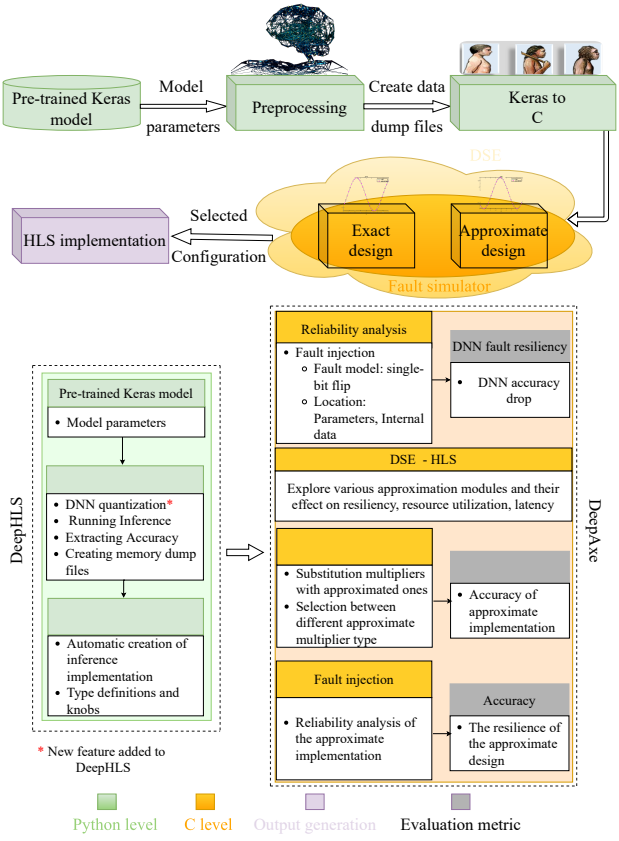


Figure 16: DeepAxe methodology flow

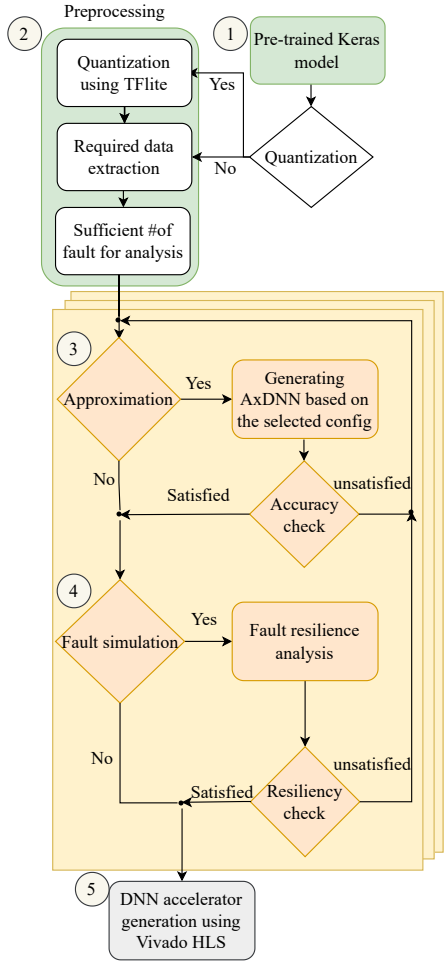


Figure 17: DeepAxe flowchart

Fig. 16 illustrates the methodology flow established in the DeepAxe tool-chain for reliability and hardware performance analysis of approximated DNN hardware accelerators. DeepAxe is a framework taking the DNNs' *Pre-trained Keras model* description as the input. Then, DeepAxe feeds the extracted model parameters through the flow to apply the initialization needed before creating the C code. The design, training and test of the DNNs are performed in Python, the *Preprocessing* step is seamlessly integrated into the same environment and is responsible for extracting the required data for the next step.

DeepAxe also supports quantizing the network down to 8-bit INT as a part of the preprocessing step. For this purpose, a full quantization is implemented, targeting all activations, weights and biases. The framework first takes the description of the network in Keras, and then uses the TFlite library to generate a training-aware quantized network. The user can replace their preferred Keras-based quantization library to the tool-chain for this step. The main output of this

step is the quantized network's parameters (i.e., weight/bias) and also the files containing the memory dump of the test data. Specifically, the *Keras to C* step implies converting all the above-mentioned parameters to multidimensional arrays in C format. The output accuracy of the generated network is also provided at this step and is kept as a baseline for the further steps of the methodology.

Reliability analysis relies on a fault injection (FI) in C assuming the single bit-flip faults in the network's activation layers for resiliency assessment. While the multiple-bit fault model is more accurate, it requires a prohibitively large number of fault combinations to be considered ($3^n - 1$ combinations, where n is the number of bits). Fortunately, it has been shown that high fault coverage obtained using the single-bit model results in a high fault coverage of multiple-bit faults [101]. Therefore, a vast majority of practical FI and test methods are based on the single-bit fault assumption.

The reliability analysis step applies the accuracy loss comparison of the network-under-test as the assessment metric. *Approximate design* (see the yellow region in Fig. 16) refers to the selective approximation of DNNs by layers provided by DeepAxe. It instruments the user with the flexibility of choosing between a) different AxC models provided by any library of approximate computing units, such as AxC multipliers in EvoApproxLib, and b) the subset of layers, for setting up different configurations of the network. As an example, in a network with n computing layers (containing both convolutional and fully connected layers), the user has 2^n combinations for exploring the exact and approximate implementations for each layer individually.

After choosing the preferred approximation configuration, the designer can go through the fault injector provided for the resiliency evaluation of the AxDNN. Eventually, the final design can be fed to the *HLS implementation* step for DNN hardware accelerator generation process by the HLS tool.

To illustrate the DeepAxe methodology, the flowchart provided in Fig. 17 shows the step-by-step process from the beginning to the end of DeepAxe tool-chain. After providing the Keras description of the network in Step 1, the user can decide if they need to quantize the network. Then, preprocessing step can be performed, enabling the user to apply a pre-analysis on the network to extract the sufficient number of faults for the reliability assessment, considering the number of its neurons.

Steps 3 and 4 in Fig. 17 show an iterative process to examine different approximated DNN combinations and, accordingly, their fault resiliency analysis to build the DSE. By enabling the fault simulation process in Step 4, the user can follow the impact of their chosen AxC model and also the approximation configuration on the resiliency of the network compared to the other AxC model/configurations and also to the exact model. Finally, the selected design and its configuration are fed into the HLS tool for implementation.

It is noteworthy that all steps in the yellow box of Fig. 16 can be iterative, and the user can repeat these steps to find the optimal point based on their requirements. For instance, the user might decide to analyze an assumed approximation configuration, i.e. AxC model for the multiplier and also the layers to approximate. If, after applying approximation, the accuracy check does not satisfy the user, they can try another approximation configuration. Once

the requirements are satisfied, it is possible to proceed to the fault vulnerability analysis. If, after applying the fault injection, the resiliency of the network is also satisfying, the next step is generating the DNN accelerator based on the selected configuration.

3.4.4 Experimental Results

Experimental results are reported in this subsection as follows: **3.4.4.1 Experimental Setup**

First, all DNNs are implemented, trained and tested in Keras. The required data for further steps of DeepAxe are also generated in the same environment. In the DeepAxe flowchart (Fig. 17), the green parts, including steps 1 and 2, refer to the steps of the framework implemented in this high-level environment. Both a three-layer MLP and LeNet-5, trained on the MNIST dataset, and AlexNet, trained on the CIFAR-10 dataset, are representative DNNs and efficient to perform the validation of the proposed methodology and framework. All networks use ReLu as an activation function.

All networks are quantized down to 8-bit INT data type, including all activations, weights, and biases, by using the TFlite [138] library in Python. The yellow parts in Fig. 17 are implemented in C. Simulations are performed on 2 x Intel Xeon Gold 6148 2.40 GHz (40 cores, 80 threads per node) with 96GB RAM. To speed up the simulation process, DeepAxe supports multi-thread parallelism, and users can benefit from this feature based on the number of cores their CPU provides.

All implementations in C are synthesizable by DeepHLS. The approximate multipliers in the C implementation of the network (referring to step 3 in Fig. 17) are adopted from the C codes provided by EvoApproxLib library [139]. In this subsection, three 8-bit INT approximate multipliers are picked from EvoApproxLib with different error, area, and power characteristics reported in Table 8. The error parameters reported in this table are as follows:

- MAE - Mean Absolute Error (Mean Error Magnitude)
- WCE - Worst-Case Absolute Error (Error Magnitude / Error Significance)
- MRE - Mean Relative Error (Mean Relative Error Distance)
- EP - Error Probability (Error Rate)

Power (power consumption in mW) and area (area on the chip in μm^2) are also reported as the design parameters in the last two columns of the table. To show the hardware characteristics of the output AxDNN, the Lookup Table (LUT) and Flip Flop (FF) utilization, as well as the number of required clock cycles for a one-time execution of the output AxDNN accelerator, are reported as the results based on the reports produced by Xilinx Vivado HLS tool on a Xilinx Spartan-7 FPGA with part number xc7s100-fgga676-1 and 100 MHz frequency.

3.4.4.2 Fault simulator: The fault simulator that is used in step 4 in Fig. 17 is implemented in the automated tool-flow of DeepAxe in a way that users can select the sufficient number of faults they need for their resiliency analysis. AxDNNs generated by step 3 in Fig. 17 are validated by means of fault injection over the test set.

Table 8: Exact and approximate multipliers used in this subsection and their parameters

Circuit name	MAE	WCE	MRE	EP	Power	Area
Exact multiplier	0.0000	0.0000	0.00	00.00	0.425	729.8
mul8s_1KVP	0.0510	0.2100	2.73	74.80	0.363	635.0
mul8s_1KV9	0.0064	0.0260	0.90	68.75	0.410	685.2
mul8s_1KV8	0.0018	0.0076	0.28	50.00	0.422	711.0

Table 9: Networks trained and quantized down to 8-bit INT for evaluation of this work

Network	Dataset	Accuracy 8-bit quantized network (%)
3-layer MLP	MNIST	80.40
LeNet-5	MNIST	85.80
AlexNet	CIFAR-10	78.50

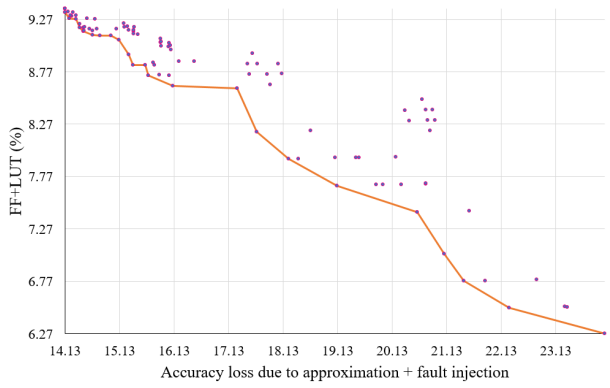
Random Fault Injection. According to the adopted fault model, a random single bit-flip is injected into a random neuron in a random layer of the network, and the whole test set is fed to the network to obtain the accuracy of the network. This process is repeated several times to reach an acceptable confidence level which depends on the number of neurons and data representation bit length based on [54].

To find the required number of repetitions for the fault simulation experiments, [54] provides an equation to reach 95% confidence level and 1% error margin. However, it can pessimistically obtain a larger number, and the execution time of the iterative fault simulation experiments would be very long. Therefore, we have performed a fault simulation for each neural network to find a smaller number of experiments in a way that the difference of the average accuracy is less than 0.1% in comparison with the average accuracy of the network achieved using the statistical fault injection approach [54]. As a result, we have selected for injection 600, 800, and 1000 random single bit-flip faults for 3-layer MLP, LeNet-5, and AlexNet fault simulation, respectively.

3.4.4.3 Validation Results

The proposed methodology is validated on three networks, i.e. a 3-layer MLP, LeNet-5 and AlexNet, trained on two representative datasets MNIST and Cifar-10. Each network is fully quantized down to 8-bit INT as a part of the preprocessing step of the methodology. The accuracy results for the quantized networks are reported in Table 9. Further, all possible combinations of approximate layers in the network are tested for selective approximation. For each experiment, three different multipliers reported in Table 8 are examined separately for efficiency to substitute the original exact multipliers.

The fault injection procedure is performed for all different configurations, and the accuracy drop, due to approximation and fault injection, is profiled. Further, the HLS synthesis results of all configurations are generated, and the resource utilization in the number of FF, LUTs as well as the number of clock cycles required for processing one image for each network, are collected. A Pareto frontier for resource utilization and accuracy drop due to applying FI on different



(a)

	Accuracy loss (%)	#of [FF+LUT] (%)	Pareto points	
Worst error	24.02	6.27	mul8s_1KVP (1-1--111)	Best area
	22.27	6.51	mul8s_1KVP (1-1--011)	
	21.44	6.77	mul8s_1KVP (1-1--010)	
	21.08	7.03	mul8s_1KVP (1-1--000)	
	20.59	7.43	mul8s_1KVP (0-1--111)	
	19.12	7.68	mul8s_1KVP (0-1--011)	
	18.23	7.93	mul8s_1KVP (0-1--001)	
	17.64	8.19	mul8s_1KVP (0-1--000)	
	17.28	8.61	mul8s_1KVP (0-0--111)	
	16.10	8.63	mul8s_1KV9 (0-1--111)	
	15.66	8.73	mul8s_1KV9 (0-1--101)	
	15.60	8.83	mul8s_1KV9 (0-1--100)	
	15.37	8.83	mul8s_1KV9 (0-1--001)	
	15.30	8.93	mul8s_1KV9 (0-1--000)	
	15.12	9.07	mul8s_1KV9 (0-0--111)	
	14.97	9.11	mul8s_1KVP (0-0--010)	
	14.77	9.11	mul8s_1KVP (0-0--001)	
Best error	14.63	9.12	mul8s_1KV8 (0-1--111)	Worst area

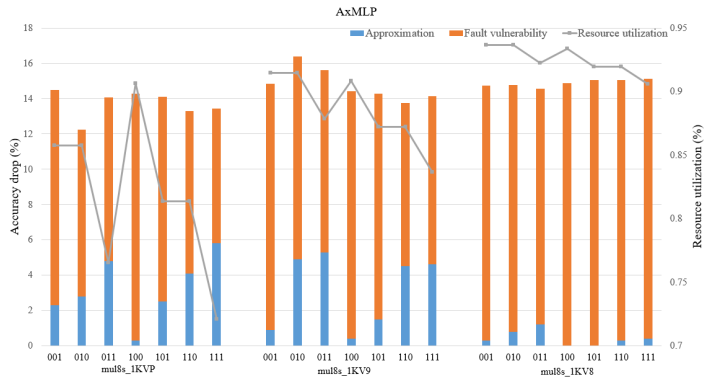
(b)

Figure 18: (a) Resource utilization of the approximate implementation vs. accuracy loss when the approximate implementation is fault-simulated (b) Approximation configuration of each point on the Pareto frontier

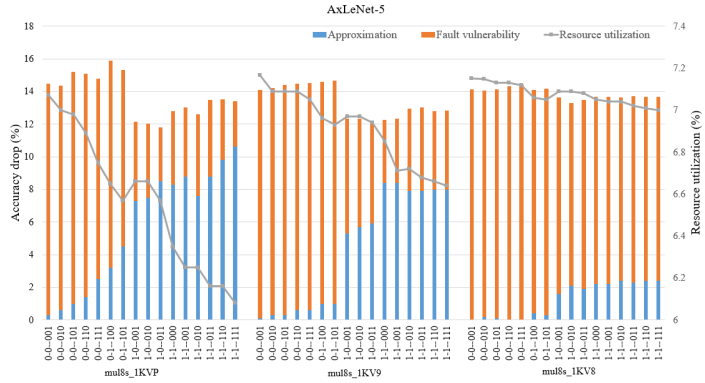
approximation configurations is plotted, and the results for LeNet-5 are reported in Fig. 18(a).

Fig. 18(b) shows the points on the Pareto frontier. The first column is the accuracy drop due to performing fault injection on that particular AxDNN configuration, the second column is resource utilization of the AxDNN in percentage, and finally, the last column is the selected approximate multiplier (AxM) and order of layers in ad-hoc (ones means that particular layer is approximated and dashes represent the non-computational layers like maxpooling). The coloured rows are some extreme and mid-range points of the Pareto chart. The same experiment is repeated for MLP and AlexNet networks, and the results for some extreme and mid-range points of their pareto charts are presented in Table 10.

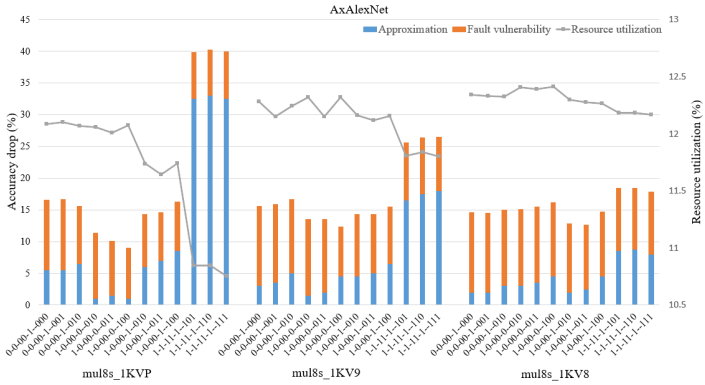
It can be observed from this table that, generally, by approximating more layers, the latency and resource utilization are less. It is also noteworthy that the



(a)



(b)



(c)

Figure 19: Reports of accuracy drop (due to approximation for different configurations), fault vulnerability, and resource utilization of (a) 3-layer MLP network, (b) LeNet-5 and (c) AlexNet

fault vulnerability of the network, which can be defined as the accuracy drop of the AxDNN due to applying FI, also becomes less. Fault vulnerability is opposite

Table 10: The impact of approximation configuration and fault injection for MLP, LeNet-5, and AlexNet.

DNN dataset	Multiplier	Layer configuration	Base accuracy (%)	Accuracy loss (%) [Exact network - AxDNN]	AxDNN accuracy loss (%) [AxDNN - FI on AxDNN]	Latency (#of clk cycles)	Resource utilization (%) #of[FF + LUT] / Total #of[FF + LUT]
MLP MNIST	mul8s_1KVP	111	80.40	5.8	7.62	206644	0.72
	mul8s_1KVP	101		2.5	11.62	272180	0.81
	mul8s_1KV9	101		1.5	12.78	274740	0.87
	mul8s_1KV9	100		0.4	14.03	274740	0.90
	mul8s_1KV8	001		0.3	14.72	285010	0.95
LeNet-5 MNIST	mul8s_1KVP	1-1–111	85.80	10.6	2.82	164864	6.27
	mul8s_1KVP	1-1–011		8.8	4.67	195584	6.51
	mul8s_1KV9	0-1–111		1.7	12.70	206408	7.93
	mul8s_1KV9	0-1–101		1.0	13.66	206504	8.19
	mul8s_1KV8	0-1–111		0.7	13.23	175784	9.12
AlexNet CIFAR-10	mul8s_1KVP	0-0-11-0–011	78.50	16.0	9.12	19933514	11.75
	mul8s_1KVP	0-0-11-0–100		17.0	10.41	20324170	11.84
	mul8s_1KVP	0-0-00-0–001		2.0	11.10	20467530	12.35
	mul8s_1KV9	0-1-11-1–111		18.5	9.58	19799882	11.04
	mul8s_1KV9	0-1-11-1–110		17.5	11.80	19945802	11.93
	mul8s_1KV9	0-0-00-0–001		3.0	12.60	20470090	12.45
	mul8s_1KV8	1-1-11-1–110		6.5	10.90	20470090	12.18
	mul8s_1KV8	0-1-11-1–111		6.0	11.70	20470090	12.19
	mul8s_1KV8	0-1-11-1–110		4.5	12.00	20470090	12.21
	mul8s_1KV8	0-0-11-0–011		3.5	12.00	20470090	12.35
	mul8s_1KV8	0-0-11-0–100		2.5	12.15	20470090	12.33
	mul8s_1KV8	0-0-00-0–001		0.0	12.64	20470090	5

Table 11: Case study: the impact of full approximation on three different MLP architectures

Network MNIST dataset	Exact network accuracy (%)	Normalized resource utilization (%) [exact network]	AxM	Accuracy drop (%)	Fault vulnerability	Normalized latency	Normalized resource utilization (%)
7-layer MLP	98.80	100	mul8s_1KV8	0.2	2.45	1.00	96
			mul8s_1KV9	1.4	1.03	1.00	90
			mul8s_1KVP	0.9	1.33	0.75	76
5-layer MLP	86.30	69	mul8s_1KV8	0.0	3.33	1.00	96
			mul8s_1KV9	1.9	2.12	1.00	89
			mul8s_1KVP	3.1	3.84	0.78	76
3-layer MLP	80.40	36	mul8s_1KV8	0.4	14.14	1.00	95
			mul8s_1KV9	4.6	7.62	1.00	88
			mul8s_1KVP	5.8	9.54	0.76	74

to fault resiliency and means the more the accuracy of an AxDNN drops due to applying FI, the more vulnerable the network is against faults. Generally, by increasing the level of approximation, the network shows better resiliency to faults. Still, there are several configurations that do not follow this trend and a tailored analysis using a framework such as DeepAxe is necessary for higher confidence.

Fig. 19 depicts the impact of different approximation units on the case-study DNNs' accuracy, resource utilization and fault vulnerability. For each network, three approximation units are chosen. For approximating the networks, the same configurations are picked to observe the impact of different AxM on the networks. Then all approximation units are applied, and the accuracy drop, fault vulnerability and resource utilization are reported. The correlation between the AxM error metrics reported in Table 8, their area overhead, and the accuracy drop of the AxDNN impacted by AxMs lead us toward a conclusion that the network accuracy is generally impacted by a) the level of approximation and the configuration of the layers that are substituted by AxM; b) the error metrics of the AxM that is used as a substitution of ExC unit.

3.4.4.4 Approximate multipliers case-study:

As a case study, three MLP networks with different architectures on the basis of a number of layers are selected. The base accuracy for each quantized network is 98.80% for the network with 7 layers, 86.30% for a network containing 5 layers and 80.40% for 3-layer MLP network. The results for full approximation of the MLP networks with each case-study approximate multiplier (AxM) are reported in Table 11.

All the values in the table are normalized to the corresponding values of the ExC networks.

For the 7-layer MLP, it is shown that the multiplier mult8s_KVP is the best option for full approximation, in the sense that the accuracy of the network drops only 0.9%, and yet, latency and resource utilization of the network are better than for the other two multipliers. Therefore, based on the application of the network, if the designer can sacrifice the accuracy for 0.9%, they can gain 25% improvement in network latency and 24% improvement in resource utilization of the implemented network on FPGA.

The situation is different for the 5-layer MLP network. Based on the results of Table 11, the best multiplier can be mult8s_KV9 since the accuracy does not drop dramatically and yet, it gains a better resiliency than the other two multipliers. Similarly, in the 3-layer MLP, the best candidate for full approximation of the network is mult8s_KV9 multiplier since it shows the best resiliency with a little accuracy drop and still, provides 12% improvement in resource utilization compared to the exact design.

In summary, this case study shows the importance of exploring different AxMs for optimal implementation, i.e. not to compromise the accuracy of the network and, at the same time, to improve the network resiliency and hardware performance of the target design.

3.4.5 Conclusion

In this subsection, we proposed a framework DeepAxe for design space exploration for FPGA-based implementation of DNNs by considering the trilateral impact of applying functional approximation on accuracy, reliability and hardware performance. The framework enables selective approximation of reliability-critical DNNs, providing a set of Pareto-optimal DNN implementation design space points for the target resource utilization requirements The design flow starts with a pre-trained network in Keras, uses an innovative high-level synthesis environment DeepHLS and results in a set of Pareto-optimal design space points as a guide for the designer. The framework is demonstrated on a case-study of custom and state-of-the-art DNNs and datasets.

3.5 FE method: APPRAISER - DNN Fault Resilience Analysis Employing Approximation Errors

This section is based on the following papers: [V, and XI]

3.5.1 Introduction

In today's applications, network parameters, e.g., weights, occupy most of the inference accelerator's areal footprint, making them natural targets for soft-

errors-caused disturbances. Unlike other logic structures, DNNs are known to be relatively resilient to transient faults. However, in practice, such faults still may cause a significant accuracy drop in DNNs because of the large area and memory requirements for the state-of-the-art DNNs accelerators. Although numerous techniques have been proposed recently to evaluate the architectural fault resilience of DNNs, they are still rather costly. Throughout the literature, Fault Injection (FI) is the most commonly used method for resilience evaluation of DNNs.

Fault injection by emulation in hardware, usually in FPGAs, is widely adopted by the industry [4] because of its ability to evaluate real-scale DNN accelerator designs with significantly shorter run times compared to software-based simulation.

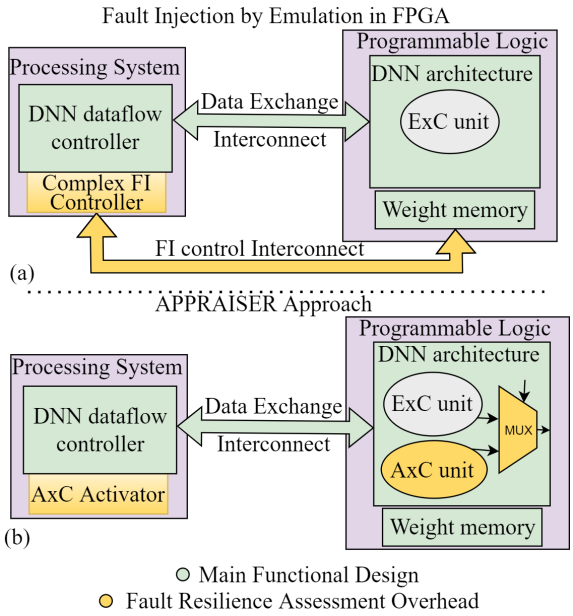


Figure 20: DNN fault resiliency assessment methods: (a) Fault injection by emulation in FPGA; (b) APPRAISER approach using errors by AxC units.

However, the state-of-the-art approaches for fault injection by emulation in hardware imply iterative procedures for each injected fault, including numerous extra memory accesses, which make them time-consuming and imply complex implementation. Fig. 20(a) illustrates the execution overheads of the general flow of FI by emulation in hardware. In particular, such an iterative approach is breaking the pipeline and requires a complex FI Controller and an extra FI control interconnect [12, 140–142].

Fig. 20 (b) illustrates the proposed approach APPRAISER, which allows for reducing the fault resiliency assessment overheads. The ability to tolerate the impact of faults on the output accuracy is called *fault resiliency* and, in practice, it is one of the contributors to the final DNN accelerators' reliability [124].

In this subsection, our contribution is a novel method of fault resiliency

analysis for DNN architectures that applies functional approximation for a non-conventional purpose and harnesses approximate computing errors for its interest. To the best of our knowledge, for the first time, *Approximate Computing* (AxC) units are adopted to improve the processing time-, design-, and control-complexity for DNN fault resiliency analysis process.

APPRAISER provides a rapid exploration of different options of the network architecture, training, dataset, etc., to study the fault resilience of the DNNs. In particular, it enables efficient analysis of subsequent layers' resilience to faults in the weights of a compromised layer.

The new method has the following advantages:

- It eliminates the need for designing and deploying an extra complex controller for the fault injection procedure. A simple approximate units enabling circuitry (AxC Activator) is employed instead.
- The inference pipeline process executes a batch of inputs with no need to break this process.
- The resilience assessment process is performed without an extra interconnect for weight sampling.
- The proposed approach is not iterative for each potential fault location, unlike the traditional fault injection. Thus, the analysis complexity is vastly reduced.

The rest of this subsection is organized as follows. Related works are discussed in 3.5.2, the methodology and framework are presented in 3.5.3, the experimental setup and results are provided in 3.5.4, and finally, the work is concluded in 3.5.5.

3.5.2 Related Works

The extensive growth of the memory footprint size in today's practical DNN inference HW accelerators increases the chances of soft errors' occurrences causing prediction failures. Even a minor change in the DNN architecture may cause a notable difference in the DNNs' architectural fault resiliency. Evaluating the resiliency of DNNs with FI by emulation in hardware is a practical method used today by the industry. There are several works emulating fault injection on FPGAs as a hardware platform.

Fiji-FIN [142] is one of such DNNs' resiliency evaluation frameworks. It considers the model's accuracy degradation as a metric to study the impact of soft errors on the network's parameters, such as weights and activation. Unfortunately, it implies severe effort for designing the fault injection campaigns. For each single fault injection, the execution of the inference should be halted for manipulating the DNN parameters, and it has to be resumed thereafter. It means that the classification time for a batch of inputs should be interrupted to apply fault injection between the classification process of two consecutive inputs.

A similar method is also used in [12, 141]. These works also propose injecting transient faults into on-chip memories of the design implemented on the FPGA. In these works, the bit stream file of the FPGA is obtained by a High-Level Synthesis (HLS) tool and imported to the FPGA. While the system is running, the

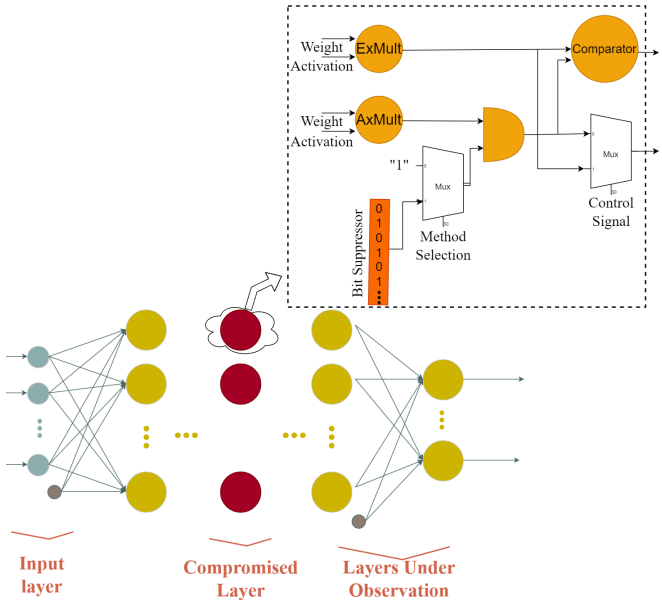


Figure 21: Proposed APPRAISER method evaluation

faults are generated and injected by the embedded processor and the reliability is evaluated in comparison with the golden model.

In contrast to the works mentioned above, this subsection proposes a novel non-iterative fault resilience analysis by exploiting the approximation errors instead of fault injection

It enables keeping the inference pipeline process to be executed on a batch of inputs unbroken.

3.5.3 Proposed Methodology

AxC is commonly used to approximate hardware components to improve compute efficiency while maintaining functional accuracy. However, in practice, the errors induced by approximation can be used to mimic the errors caused by faults in logic circuits. These errors affect the outputs of the corresponding units and propagate to subsequent layers, impacting their activations (Fig. 21). The proposed approach for evaluating DNN's fault resiliency using approximate computing (AxC) units is presented in Fig. 21. To implement our proposed method, an AxMult, or an AxMult + a bit suppression unit (AxMult+) is implemented along with the exact implementation of the multipliers (ExMult) in the network, depending on whether the network is being run in functional or fault resilience assessment mode. The golden inference for the validation dataset is run only once, and the layer outputs are stored and compared with a Comparator unit. The Bit Suppressor unit is meant to increase the probability of more significant bits of the neuron being impacted by faults. The less significant bits of the layer Output Feature Map are already affected by the AxMult with proper randomness depending on the data distribution in the network and layers.

The overall flow of the proposed method is illustrated in Fig. 22. In Step 1, the user initializes the method by selecting the compromised layer in the DNN structure, the validation dataset (i.e., DNN inputs), and the application-specific target fault rate assumed for the analysis. In Step 2, suitable AxC units are selected for Approximate Processing Elements (AxPEs), such as the AxC multipliers from a relevant library, e.g., the EvoApproxLib [139], or their variants with bit suppression. In Step 3, the selected AxMults started executing the compromised layer by enabling corresponding AxPEs along with the Exact Processing Elements (ExPE) in the DNN architecture. The DNN inference is run while keeping the network pipeline intact, and the resulting DNN output accuracy drop is recorded as the primary metric for analyzing DNN fault resilience. A more significant drop in accuracy with induced errors implies a less fault-resilient DNN implementation. At the same time, the outputs of the AxMults are compared with the ExMults outputs to calculate the actual error at each neuron. The rest of the inference is executed by ExMults for both erroneous and exact outputs, and the comparison is performed for all the subsequent neurons of the network.

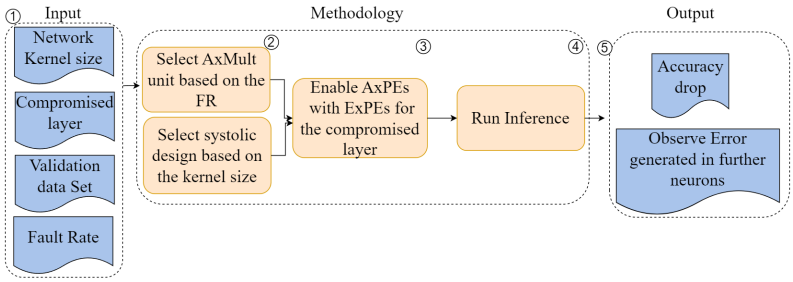


Figure 22: APPRAISER methodology flow

The characteristics of the approximation-induced errors can be evaluated using different metrics such as normalized error, number of flipped bits, and impact on the neural network classification accuracy drop. In this study, we rely on a simple set of metrics that includes:

- Normalized error: the average error on the output of each layer is calculated by subtracting the neurons' outputs of that layer from the golden output and dividing all the error values by the maximum value.
- Network accuracy: calculated by executing the network under different circumstances (faulty, AxMult, AxMult + bit suppressor and bit suppressor) over the test set.
- Bitflips in subsequent layers: calculated by comparing all bits in the next layers' outputs with the golden model and counting the bits that do not match as flipped bits.

Accelerator Model:

Fig. 23 illustrates the accelerator model to perform resilience analysis on FPGA. It consists of two different systolic architecture designs based on the

network under test. The $N \times N$ systolic architecture is used based on the convolution layers' kernel size to perform the most optimum dot matrix. At the same time, all designs have ExPE and AxPE to perform the resilience analysis and benefits of a dual register to store the results of both approximate systolic and exact systolic for further comparisons. An Error Detector (ED) module is also provided to compute the error generated at each neuron's output compared to the exact output and can be used for the neuron's vulnerability evaluation.

This implementation provides us with the following features:

1. Understanding the vulnerability of neurons by computing the error generated through the hardware and further layers by comparing the exact and approximate systolic design outputs;

2. Increasing the controllability for enabling errors in each layer individually and keeping the other layers correct;

3. Eliminating the need for designing and deploying an extra complex controller for the fault injection procedure. A simple approximate unit enabling circuitry is employed instead;

4. The inference pipeline process executes a batch of inputs with no need to break this process;

5. The resilience assessment process is performed without an extra interconnect for weight sampling;

6. The proposed approach is not iterative for each potential fault location (unlike the traditional fault injection). Thus, the analysis complexity is vastly reduced.

Note that the features (3)-(6) are specific for FI emulation in Programmable Logic and generally not available in Processing Logic based methods such as Fiji-FIN.

3.5.4 Experimental Results

Experimental results are reported in this subsection as follows:

3.5.4.1 Evaluation methodology To assess the feasibility of the proposed method, we implemented the same flow as shown in Fig. 21 with fault injection (FI). Using Table I, we narrowed down the list of candidate approximate multipliers from the EvoApproxLib library [139] based on several relevant metrics, with a primary focus on two established features, namely, the Variance of Error Distance (Var-ED) and Root Mean Square (RMS-ED) presented in [143]. These metrics are crucial in determining the approximation-induced errors that affect the performance of an AxC unit in DNNs. We selected mul8s_1L2N for the experiment based on these metrics and results achieved from the high-level experiments on the network through the proposed GPU accelerated framework for CNN approximation in Section II.

For the reference part, we repeated the fault resiliency evaluation on the original network, which was instrumented with a state-of-the-art FI method [142]. In this study, we considered the injection of multiple bitflips at a random location

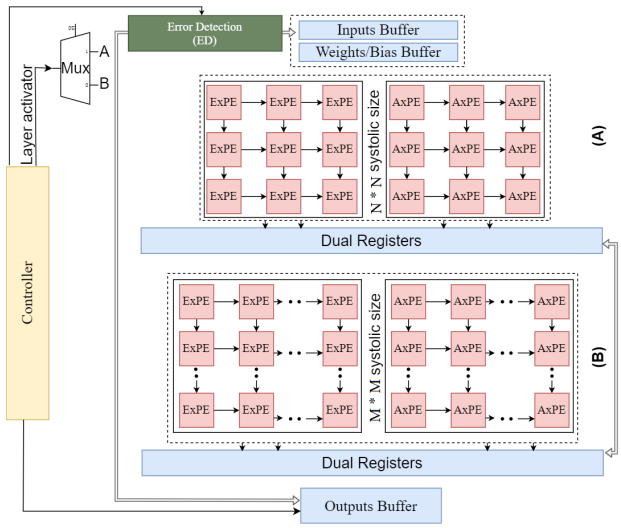


Figure 23: Proposed systolic architecture for our Resiliency assessment DNN accelerator framework

in all OFM' bits of the compromised layer for every input in the DNN validation test set. In this case, we assumed that 10% of the weights' bits were faulty.

To achieve a high FI confidence level using the statistical fault injection approach [54], we repeated the experiment for each fault model with 1000 random faults per image. The average accuracy of all repetitions was then reported.

We evaluated the impact of AxMult, AxMult + Bit Suppression (AXMult+), Bit Suppression alone, and fault injection, along with normalized error and the number of flipped bits, on the DNN accuracy. The results show a drop in DNN accuracy due to these factors. We compared the normalized error and the number of flipped bits for each scenario.

3.5.4.2 Experimental Setup

To evaluate the feasibility of the proposed method, a case-study Convolutional Neural Network (CNN) with two convolutional layers, two max-pooling, and one Fully-Connected (FC) layer was implemented and trained. The simulations were performed on an Intel® Core™ i7-6800K CPU @ 3.40GHz × 12, and the proposed method was implemented with Python 3. The hardware synthesis and implementation results are produced by the Xilinx Vivado HLS tool on a Xilinx Versal VCK190 FPGA (xcvc1902-vsva2197-2MP-e-S) at 166 MHz operational frequency.

The CNN under study is trained on a dataset of 2000 images of animals (cats and dogs) and humans for binary classification. The accuracy of the network over the test set (including 450 images of animals and humans) is 93.34%. Bit truncation quantization is applied in network parameters during training, and data precision is reduced to 8-bit.

3.5.4.3 Evaluation Results

We analyzed the similarity of the fault resiliency analysis results obtained by

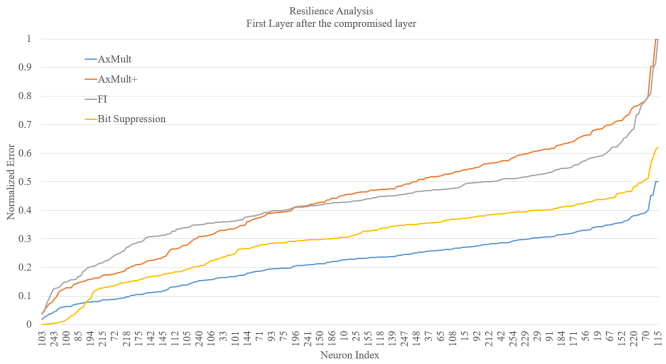


Figure 24: Normalized output error of Pool2: Applying AxMult, AxMult+ , Bit Suppression and FI on the Conv1

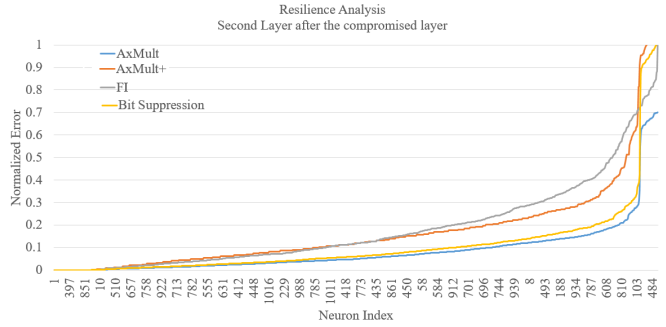


Figure 25: Normalized output error of Conv2: Applying AxMult, AxMult+ , Bit Suppression and FI on the Conv1

fault emulation and our proposed method using the metrics identified in Section 3.5.3.

Fig. 25 shows the distribution of *normalized error* in the output of the second convolutional layer (Conv2) in the presence of 10% random faults in the first convolution layer (grey), errors induced by AxMult (blue), and errors induced by AxMult + bit suppressor (orange) enabled in the first convolution layer, respectively. Fig. 24 reports the impact of applying FI and our proposed method on the same convolutional layer and its effect on the second pooling layer of the network. These results demonstrate the similarity in error propagation trends between the proposed and reference methods.

In practice, by analyzing these charts, users can set a criticality threshold on the output error of the neurons based on their application and determine the number and indices of neurons to be used for any protection techniques. Generally, if we set the threshold at some error value, all methods suggest some neuron indices for mitigation techniques. As it can be concluded, both AxMult + bit suppression and FI show very similar behaviors. However, relying solely on the AxMult or bit suppression techniques is quite inaccurate for high fault ratios like this case study here.

For example, by setting the error threshold to 0.7, FI will recommend the

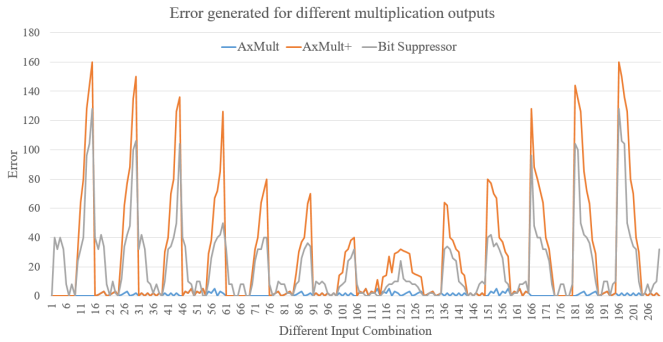


Figure 26: Multiplication output error generated by AxMult, AxMult+ and Bit Suppression

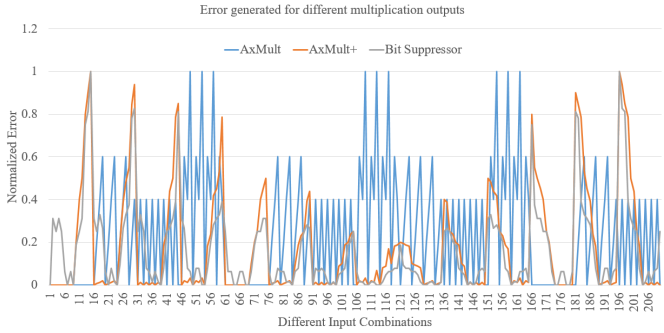


Figure 27: Normalized Multiplication output error generated by AxMult, AxMult+, and Bit Suppression

user to protect 50 out of 1024 neurons of the Conv2 network's second CONV's neurons, while AxMult + bit suppression will recommend 53 out of 1024 neurons, including all the critical neurons recognized by FI. Fig. 26 and Fig. 27 show the error distribution of the three different methods, i.e., AxMult, AxMult + bit suppressor, and bit suppressor on the output of a multiplication operation with all the combinations of two 8-bit inputs. From Fig. 26, it is evident that the error values generated by AxMult + bit suppressor can almost cover a vast range of different values, and Fig. 27 shows that the error is evenly distributed on all different input combinations.

Table. 12 is reporting the number of bitflips and accuracy drop in subsequent layers caused by the compromised first convolution layer. These results also demonstrate the strong similarity of the trends in error propagation by the AxMult and its variants with the reference method. In case of accuracy drop, AxMult + bit suppression shows a strong correlation with the FI method and surpasses the other two methods.

Table 13 reports details of the hardware accelerator implementation. Based on the results, the proposed implementation can be executed on the FPGA at 166 MHz clock frequency, and only by using $\sim$16% of the available LUTs on the board all three mentioned systolic-array size architectures can be implemented to improve the efficiency of the accelerator. The timing comparison of the

Table 12: Bitflips and Accuracy drop induced by our proposed method vs. the reference fault injection method by fault rate 10% in OFM of the first convolution layer

Measured Layer	Bitflips in subsequent layers			
	FI (reference) [%]	AxMult [%]	AxMult+ [%]	Bit suppressor [%]
Conv1	10	10.30	10	10.20
Pool1	9.07	9.20	9.06	9.15
Conv2	16.76	16.80	16.77	16.83
Pool2	16.51	16.66	16.53	16.62
Accuracy drop [%]				
	16.73	9.33	18.33	24.73

Table 13: Hardware implementation of the proposed hardware accelerator

	Resource Utilization (%)				
Conv2D systolic size	**LUT**	**FF**	**BRAM**	**Data Path Delay**	**CLK Frequency**
3*3	0.03	0.00	0.83	Logic: ∼20% Route: ∼80%	166 MHz
5*5	0.09	0.00	0.83		
32*32	15.30	0.91	0.85		

proposed method and the state-of-the-art fault injection method are presented in Table. 14. As it can be concluded, by keeping an acceptable accuracy of FI in identifying the critical neurons, we get thousands of times speed-up in the resilience assessment of the DNNs. (Specifically, it is 5417 times in this example). At the same time, the proposed method does not need extra interconnects to manage the assessment process, and the original controller of the accelerator can take care of the fault resiliency assessment process.

Table 14: Timing overheads of the proposed method vs. the reference fault injection method (Conv1 layer)

Network	Analysis Control Circuitry	Interconnects	DNN execution time in FPGA
Base CNN	N/A	Data Exchange Interconnect	∼120ms
Fault Resilience Assessment			
CNN instrumented with FI	Complex FI Controller	(Data Exchange + FI) Interconnect	∼650,000ms
CNN instrumented with AxMult+	Accelerator Controller	Data Exchange Interconnect	∼120ms

3.5.5 Conclusion

The state-of-the-art methods for fault injection by emulation incur a spectrum of time-, design- and control-complexity problems. To overcome these issues, a novel resiliency assessment method called APPRAISER is proposed that applies functional approximation for a non-conventional purpose and employs approximate computing errors for its interest. By adopting this concept in the resiliency assessment domain, APPRAISER provides thousands of times speed-up in the assessment process, while keeping high accuracy of the analysis. In this subsection, APPRAISER is validated by comparing it with state-of-the-art approaches for fault injection by emulation in FPGA. By this, the feasibility of the idea is demonstrated, and a new perspective in resiliency evaluation for DNNs is opened.

3.6 Hybrid Analytical and Hierarchical FI-based Reliability Assessment for Systolic-Array-Based DNN Accelerators

This section is based on the following papers: [VIII, X, and XII]

3.6.1 Introduction

Simulation-based FI is less expensive in terms of equipment, but at the same time, it implies the most resource-intensive computations and is very time-consuming. On the other hand, analytical and hybrid approaches are proposed to reduce the complexity of exploiting FI for the reliability assessment of DNNs [IX]. Analytical approaches attempt to provide mathematical approaches to estimate reliability, nonetheless, their evaluation accuracy is challenging to address and they are mostly hardware-agnostic. Whereas hybrid FI-analytical approaches can take advantage of both FI and analytical approaches in terms of scalability, accuracy and hardware-based analysis [IX]. To our knowledge, there is no work assessing the reliability of Systolic Arrays (SAs) running DNNs using hybrid methods to accelerate the analysis process.

This section introduces a novel hybrid analytical and hierarchical simulation-based reliability assessment for systolic-array-based DNN accelerators based on FI. This methodology is tailored to significantly accelerate the fault injection process on systolic-array-based DNN hardware accelerators. The systolic-array core of the DNN accelerators is modeled using a URE system [119]. The proposed injection flow has been implemented based on an SA simulator [IV], thus offering the advantage of being more precise than a hardware-agnostic tool, yet much faster than traditional RTL-level simulations. An analytical method [VIII] is used to prune the fault space to further optimize the tool and speed up the reliability assessment process.

3.6.2 Proposed Methodology

The methodology for the proposed framework is illustrated in Fig. 28. After providing the trained network parameters and architecture, in Step 1, the fault list is generated. Possible fault locations can be defined by the user or can be a random fault list generated based on the network parameters by the framework. In this work, we consider random transient faults in the registers of the systolic array's processing elements.

In Step 2, to prune the fault space, we adopt and extend the DeepVigor methodology presented in [VIII] that provides vulnerability analysis for DNNs and QNNs, respectively. To this end, we find error values for each output fmap that misclassifies the network output. Let $\delta_k^l(X_i)$ be an added positive or negative error value to an output fmap by a fault in the k-th neuron at layer l with input data X_i. For each neuron, we find the minimum positive and maximum negative $\delta_k^l(X_i)$ that misclassifies the output from the golden classification. This value is obtained for all input data X and aggregated over them. The aggregation leads to a vulnerability value range for each neuron, as shown in Fig. 29. It is labeled as follows:

- **Vulnerable** (red area): if a fault deviates the output of a neuron as in this range, it will certainly lead to misclassification for any input.

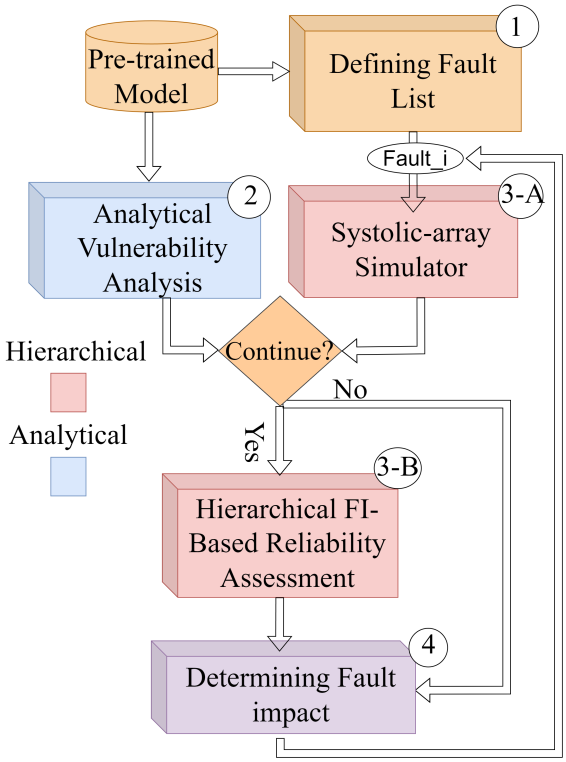


Figure 28: The proposed methodology for hybrid analytical and hierarchical reliability assessment for systolic array DNN-HAs

- **Non-Vulnerable** (green area): if a fault deviates the output of a neuron as in this range, it will not change the output classification for any input.
- **Semi-vulnerable** (grey area): if a fault deviates the output of a neuron as in this range, it might or might not lead to misclassification for any input.

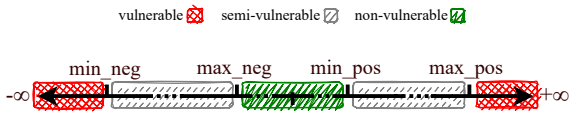


Figure 29: Vulnerability ranges for fault space pruning

This analysis enables pruning the fault space in a way that we map the deviations at the erroneous outputs of neurons induced by fault to the obtained vulnerability ranges for neurons. *If the error corresponds to the red or green areas, we immediately classify them respectively as critical or non-critical and do not continue the fault simulation.* Otherwise, if the error corresponds to a semi-vulnerable range, the fault simulation is required to be performed.

In Step 3-A, the simulation is performed from the beginning of the DNN to the fault's location to obtain the erroneous output of the corresponding neuron. The error is mapped to the vulnerability values of the corresponding neuron (shown in Fig. 29). If the fault is pruned (corresponding to green or red areas), the fault impact is determined (Step 4). Otherwise, the simulation continues (Step 3-B) to obtain the fault impact (Step 4).

In Steps 3-A and 3-B, switching between high-level API and systolic-array simulator is done by solving the URE system mentioned before; this step is described later in this section and Fig. 30. In Step 4, the reliability of the network and the impact of the faults are reported by different metrics. After evaluating the impact of one fault, the next fault is selected for evaluation.

The hardware simulation in Step 3-A is based on a formalization of the problem to solve through a URE system. Such a system can describe the problem to be solved by the architecture through a set of recurrence relations. In our specific case, we are interested in solving the problem of matrix multiplication. We associate the following system to such a task.

$$c(i,j,k) = c(i,j,k-1) + a(i,j-1,k) \times b(i-1,j,k) \tag{27}$$
$$a(i,j,k) = a(i,j-1,k) \tag{28}$$
$$b(i,j,k) = b(i-1,j,k) \tag{29}$$
$$\tag{30}$$

The generalization of this process is described in [119]. This equation system can be associated with actual hardware processing by projecting the iteration space to the physical space. The iteration space contains all the points (i,j,k) of the equation system. The physical space describes where (i.e., which processing element) and when (i.e., at what clock cycle) each computation happens in the real world. To achieve that, the iteration space is projected twice: the

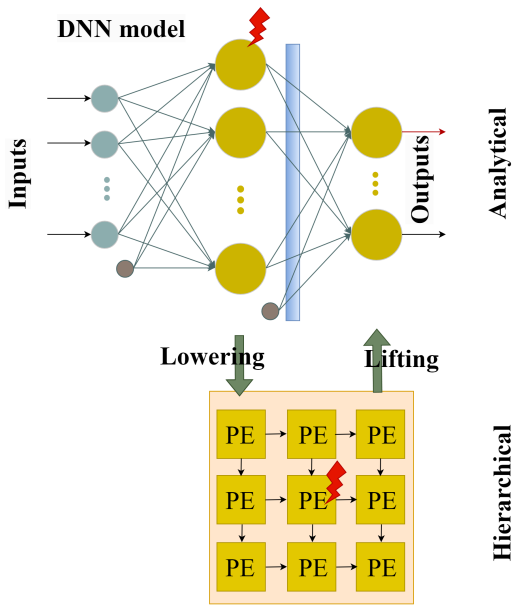


Figure 30: Fault injection across hierarchical and analytical abstractions

first time, the resulting points will correspond to the spatial arrangement of the processing elements; the second projection determines iso-temporal planes, identifying operations that are computed during the same clock cycle but on different processing elements; each plane corresponds to a different clock cycle. The space-projection matrix P and the temporal dimension vector π are used later.

In order to perform the simulation, it is sufficient to solve the system shown above. Nevertheless, this method gives the possibility of injecting faults in the values in a hardware-aware fashion. To achieve the injection, it is sufficient to change the values a, b and c at specific iterations (i,j,k). The faulty values must then be propagated to the following PE. Given the system of equations, the propagation can be computed easily, taking into account the transformation matrix. Figure 31 shows the concept. In this case, an injection in the element $s=(x,y,t)$ on the generic line i is done between times 0 and ∞. The injected elements are visible in the figure. Specifically, the fault will propagate in time, thus injecting also $s+\delta t_i$ and $s+2\delta t_i$. In the same way, this fault will propagate in the space to the element cascading from s. The value propagation only happens after each clock cycle, which means that the next injected element will be also displaced in time, thus injecting element $s+\delta x_i+\delta t_i$. In the same way, the latter will propagate to the following element on the following clock cycle, thus injecting element $s+2\delta x_i+2\delta t_i$ and so on. The injected points can be translated into iteration vectors i,j,k using the inverse transformation. Formally, it is sufficient to consider the injection as a function h applied to one of the three values, e.g. $a(i,j,k)=h(a(i,j-1,k)$

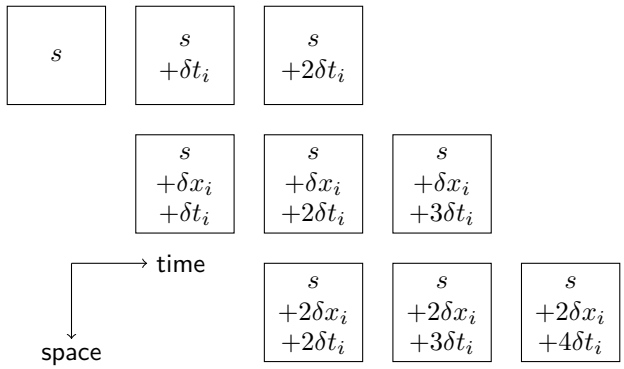


Figure 31: Fault propagation in systolic array. When injecting element s, the fault is propagated in time (thus affecting elements $s+\delta t_i$ and $s+2\delta t_i$) and in space (forwarding the faulty value to neighboring elements $s+\delta x_i+\delta t_i$, $s+2\delta x_i+\delta t_i$ and so on).

3.6.3 Experimental Results

To evaluate the methodology, experiments were performed using a 16-bit quantized LeNet-5 trained on the MNIST dataset. The network was injected with random transient faults (assuming that they are caused by Single-Event Upsets). More specifically, a fault affects a random bit in the weight register of a random processing element. The fault causes an error, which manifests itself by fixing the value of a bit to either 0 or 1. The target architecture is an Output-Stationary Systolic Array. In this experiment, faults are injected only in the first layer of the network. It is simulated using the URE system to obtain its output values. The output is compared against the vulnerability ranges obtained by the analytical approach, which determines whether the injected fault is vulnerable, semi-vulnerable or non-vulnerable (see Fig. 29). If the fault happens to be semi-vulnerable, the simulation must be carried out until the end to determine whether the fault produced misprediction or not; in the other two cases, the output is pre-determined, and there is no need to complete the simulation until the end. It is worth mentioning that the vulnerability ranges are obtained in less than one minute on an NVIDIA 3090 GPU. This methodology allows to accelerate the process of fault injection simulations.

The produced fault list includes 964 random faults, each simulated with 100 random input images.

Table 15 shows the number of simulations in each category. The simulations are grouped by channel for better understanding. The first column indicates the channel in which the faults are injected. The “non-vulnerable", “semi-vulnerable", and “vulnerable" columns indicate the number of simulations with the corresponding vulnerability. The “speed-up" column indicates the percentage of simulations that do not need to be fully performed because of the vulnerability ranges gathered in the previous step. More specifically, speed-up is the percentage of pruned simulations.

As observed in Table 15, no fault was pruned as vulnerable using the analytical approach. This is because the analytical approach did not provide any red area

Table 15: Channel-wise analysis of the fault injection speedup. The last column (su) indicates the percentage of vulnerable and non-vulnerable simulations with respect to the total.

Channel	Non-vulnerable	Semi-vulnerable	Vulnerable	Speed-up (%)
0	13673	3127	0	81.38
1	12606	3594	0	77.81
2	13044	2956	0	81.52
3	10392	5508	0	65.35
4	11811	3589	0	76.69
5	13173	2927	0	81.82

for the first layer of LeNet-5, yet it provides an effective green area in which up to 81.52% of faults are pruned. Table 15 shows that consistently, for each layer, at least 65% of the simulations are predetermined using the analytical approach leading to a remarkable time-saving for simulations.

If we assume that x is the time needed to perform a single hardware simulation (computing a convolution between a kernel and a feature map), we would need a total time t for a single inference as follows:

$$t = x \times \sum K_{i,j,l} \qquad \forall (i,j,l) \in NN_f \subseteq NN \tag{31}$$

where K is always equal to 1 and indicates a single convolution and (i,j,l) is the triple $(fmap_{input}, fmap_{out}, \text{layer})$ in the neural network NN. NN_f represents subset of the convolutions affected by the fault f.

For example, in our LeNet-5, the set NN has 6 elements for the first layer: $(1,1,1),(1,2,1),\dots(1,6,1)$. The second layer has 6×16 elements: every fmap of the output of the first layer has to be convoluted with every kernel of the second layer, so we will have $(1,1,2),(1,2,2),\cdots(1,16,2),\cdots(2,1,2),\cdots(6,16,2)$. Note that the first layer only has one input fmap: the input image itself, this we only have 6 convolutions in that case. The subset NN_f corresponds to all those convolutions whose input fmap is different than the golden and thus needs to be recomputed, taking into account the errors introduced by the fault. In our case, with the type of fault chosen, we would have an error affecting only one value in the first channel. Such an error is propagated to the following layers, although it may be masked. Especially when considering permanent faults, we would be forced to simulate every convolution until the output to determine the criticality of the fault. In our setup, NN_f would have a total of 102, considering that our systolic array only computes convolutional layers. Using the analytical approach made it possible to interrupt the simulation just after the first layer. In the conducted fault injection campaign, about 76% of the total simulations were pre-determined, allowing us to simulate the systolic array only once in 76% of the total inferences. This gave us an average time per simulation $t' = x \times (0.24 \times 102 + 0.76) \approx 0.24 \times t$ and a total 86% of fault injection speed up.

Figure 32 shows the speed-up in percentage concerning the injected bit position. It can be concluded that the most significant bits (0 to 7) are the ones that produce a smaller speed-up in general. Most of these faults fall in

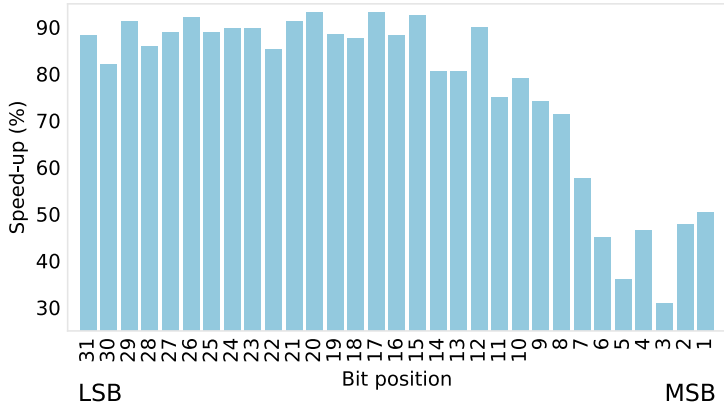


Figure 32: Speedup per injected bit position

the "semi-vulnerable“ category, thus it is necessary to propagate the error to infer the fault criticality. The least significant bits (14 to 31), on the other hand, provide great speed-up since the error is small enough for the simulation to be classified as "non-vulnerable".

Finally, the methodology is applied for SDC-1 (i.e., Silent Data Corruption leading to misclassification) analysis.

Table 16: SDC1 computed over each channel. The second column (misclassified) shows how many “semi-vulnerable" simulations ended up misclassifying the output. The third column (SDC1) shows the metric computed over the whole batch.

Channel	Misclassified	SDC1 (%)
0	48	0.29
1	50	0.31
2	41	0.26
3	93	0.58
4	69	0.45
5	60	0.37

Using the data in table 15, we could compute a lower bound corresponding to the number of “non-vulnerable" simulations. Since there is consistently no “vulnerable" simulation, the lower bound effectively corresponds to the complement of the speed-up. Furthermore, the SDC-1 was calculated for the “semi-vulnerable" faults. Table 16 shows the metric computed for the network under analysis. The second column shows the number of simulations that ended up in misclassification. The third column shows the metric itself. The network shows great resilience to this type of fault. In total, the SDC1 is 0.37%.

3.6.4 Conclusion

The presented approach is capable of significantly reducing the fault injection simulation time. This method will be extended to larger DNNs considering various fault models for obtaining reliability evaluation metrics and combining

the assessment with selective hardening techniques [X].

3.7 Chapter Conclusions

In conclusion, the assessment of DNN reliability is a multifaceted domain categorized primarily into Fault Injection, Analytical, and Hybrid methods. Fault Injection methods, which are the most commonly utilized, simulate various fault scenarios to evaluate the robustness of DNNs under different conditions. This category includes approaches such as Fault Simulation, Emulation in Hardware, and Irradiation, each offering unique insights into the potential vulnerabilities of DNN implementations across different hardware platforms. Analytical methods, on the other hand, focus on understanding the intrinsic structural weaknesses of DNNs by analyzing the connections and interactions between neurons and layers. This approach facilitates a theoretical evaluation of DNN reliability without the extensive overhead of practical fault injection experiments. Hybrid methods combine the strengths of both Fault Injection and Analytical approaches, providing a balanced methodology that enhances reliability assessment through comprehensive analysis and practical fault simulation. The works reviewed in this chapter highlight the advancements in these methodologies, emphasizing their roles in ensuring the dependable operation of DNNs in various applications.

4 Reliability Enhancement of DNN Hardware Accelerators

4.1 Introduction

This chapter proposes techniques aimed at enhancing the reliability of Deep Neural Networks (DNNs) when deployed on hardware accelerators such as FPGAs and ASICs. The chapter presents two primary techniques: AdAM (Adaptive Approximate Multiplier) and FORTUNE (Fault TOleRance Technique). These techniques are designed to mitigate the effects of hardware faults while maintaining performance efficiency. AdAM allows dynamic adjustment of approximation levels to reduce resource usage while maintaining fault tolerance, making it suitable for fault-prone environments. FORTUNE, a hardware-agnostic method, provides fault resilience by optimizing memory overhead and ensuring critical bits are protected, enhancing DNN reliability without adding significant resource consumption. The chapter combines theoretical frameworks with experimental results, demonstrating substantial improvements in power consumption, accuracy, and fault tolerance over existing methods.

4.2 AdAM: Adaptive Approximate Multiplier for Fault Tolerance in DNN Accelerators

This subsection is based on the following papers: [II, and III]

4.2.1 Introduction

In the past decades, Deep Neural Networks (DNNs) demonstrated a significant improvement in accuracy by adopting computationally intense models IX. Consequently, the size of these models has increased drastically, imposing challenges in their deployment on resource-constrained platforms [144].

Different DNN compression techniques such as model quantization and pruning [144] as well as approximate computing (AxC) [145] VI enable the use of DNNs in edge devices. While these techniques decrease the accuracy of DNNs, they bring the benefits of lower resource utilization and energy consumption and higher system efficiency XII. As an example, quantizing DNNs down to 8-bit INT gained popularity in edge AI applications, because of the minimal impact on accuracy drop and a significant reduction in memory footprint [146].

On the other hand, the role of DNNs in a wide range of safety- and mission-critical applications e.g., autonomous driving, is expanding. Therefore, deploying a DNN accelerator requires addressing the trade-off between different design parameters and *reliability* VI. Although DNNs possess certain intrinsic fault-tolerant and error-resilient characteristics, it is insufficient to conclude the reliability of DNNs without considering the characteristics of the corresponding hardware accelerator.

Consider Fig. 33 that demonstrates different possible fault locations in an AI accelerator and their negative effect on the object detection task. In the example, a pedestrian has been identified as a bird, and a red light is misclassified as a green one leading to a potentially disastrous situation. In this work, the faults in the computational core of the AI chip are considered.

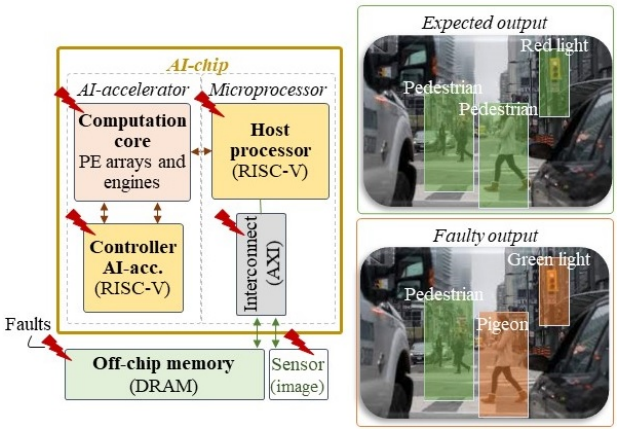


Figure 33: Possible locations of faults in AI chip I

Moreover, with the continuous scaling-down of the process, there is a discernible trend indicating that the Soft Error Rate (SER) of combinational circuits may surpass that of sequential circuits [147], [148]. Therefore, the main focus of this study is introducing a novel reliability technique to mitigate soft errors in the combinational logic of DNN accelerators while keeping resource utilization and energy consumption low.

This work presents an architecture of an adaptive fault-tolerant approximate multiplier (AdAM) tailored for ASIC-based DNN accelerators. The multiplier is based on the logarithmic Mitchell's multiplier that substitutes multiplication with the addition of approximated logarithms of the operands. The proposed multiplier protects higher-order bits of the product based on the maximum position of the leading one bit in the input strands of the multiplier. This multiplier is a negative overhead fault tolerance approximate multiplier compared to the exact multipliers. The contributions of this subsection are as follows:

- The architecture of a novel adaptive fault-tolerant approximate multiplier tailored for DNN accelerators, including an adaptive adder relying on an unconventional use of the leading one position value of the inputs for fault detection through optimizing unutilized adder resources.
- DNN accelerator fault detection and protection methodology with formal definitions for scalability and reproducibility.
- Implementation and validation of the multiplier design. Reliability and hardware performance trade-off assessment and comparison of the proposed multiplier with exact and approximate state-of-the-art multipliers using several state-of-the-art DNN benchmarks.

The proposed multiplier provides a reliability level close to the multipliers protected by Triple Modular Redundancy (TMR) while utilizing 2.74 $\times$ less area and having 39% less power-delay product compared to the unprotected exact multiplier. In fact, it has similar area, delay, and power consumption parameters

compared to the state-of-the-art approximate multipliers with similar accuracy while providing fault detection and mitigation capability.

The remainder of this subsection is organized as follows: 4.2.2 summarizes related works. 4.2.3 presents the proposed method. 4.2.4 presents the experimental setup and discusses the results. Finally, 4.2.5 concludes the work.

4.2.2 Related Works

Multipliers are one of the primary arithmetic building blocks widely used in DNNs. Approximate computing is a promising technique for designing digital circuits with lower area and power consumption while achieving a higher working frequency, particularly when the target application has some error resiliency, such as DNNs [145], XII. Various approximate multipliers are proposed in the literature. These options encompass dynamic segment multiplier (DSM) structure [149], dynamic range unbiased multiplier (DRUM) structure [150], memristor-based multipliers [151], simplified adder-based or truncated multipliers [152], utilization of inexact compressors [153], and approximate booth multipliers [154]. One of the other approximation techniques to further speed up the multiplication is to move to the logarithmic numbering system to compute addition instead of multiplication. The general idea is to obtain the logarithm of the inputs, calculate their sum, and convert the output value to the final result through an antilogarithm operation [155], [156]. The complexity and accuracy of this method come from the logarithm and antilogarithm steps. Truncating the input operand of multiplication and using logarithmic approximation also has a dual effect on the area, speed, and power consumption improvement. The first approximate logarithmic multiplier was proposed by Mitchell, who used binary logarithms to approximate multiplication [156]. There are several studies conducted on improving the performance and accuracy of this method by considering DNN application [157]. Tosam is a scalable approximate multiplier that reduces the number of partial products by truncating each of the input operands based on their leading one-bit position and improves delay, area, and energy consumption up to 41%, 90%, and 98%, respectively, [152]. ScaleTRIM is a scalable approximate unsigned LOD multiplier for DNNs that exploits curve fitting and linearization for fitting input products and a novel error compensation method using lookup tables [155].

The error introduced by the approximation is deterministic, and its impact can be studied comprehensively on the accuracy drop of the network. However, soft errors are unpredictable in contaminated and harsh environments that can lead to DNNs malfunction and accuracy drop drastically [158]. In contrast to the proposed approximate multipliers, AdAM considers reallocating resources saved by approximation for fault tolerance. Recent research investigates the reliability of DNNs alongside approximation XII, VI. In [159], DNNs and approximated DNNs are tested in the presence of stuck-at faults, and the results demonstrated that approximated DNNs are more resilient under special conditions.

Triple Modular Redundancy (TMR) and Gate-Sizing (GS) are two well-established hardening methods widely employed to mitigate soft errors in combinational circuits. Despite achieving 100% fault coverage for a single fault in one module of a combinational circuit, TMR incurs a substantial near 200% area

and power overhead [6]. Therefore, numerous algorithms and frameworks are developed to enhance the efficiency of applying these methods and balance their hardening effects and design costs [160].

A relaxed fault-tolerant (RFT) hardening method that exploits the inherent fault tolerance of different applications to reduce implementation costs was proposed in [161]. However, RFT lacks generality and flexibility. Approximate TMR (ATMR) is a technique that replaces some modules of TMR with approximate ones while ensuring that the majority voter gives the correct output [162]. This technique is investigated for various purposes and platforms, including some that are exclusive to FPGAs and some that work with both FPGAs and ASICs [160]. However, ATMR still requires duplicating the whole combinational circuit, even at the finest level of granularity.

To address the high overhead incurred by traditional fault tolerance methods, this work presents an adaptive approximate multiplier that provides a high level of reliability while using less area and PDP than an exact multiplier.

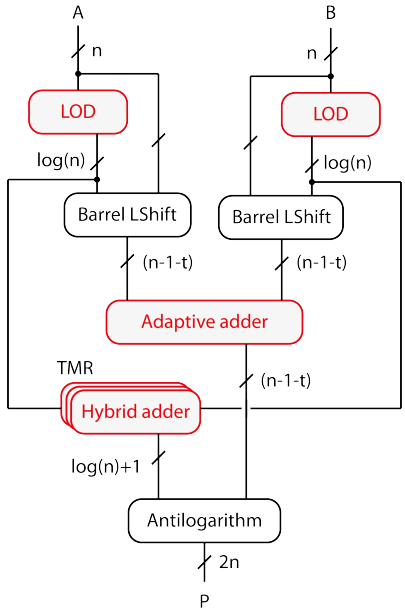


Figure 34: AdAM architecture (the contributions and extensions to the logarithmic Mitchell multiplier are marked with red color)

4.2.3 Proposed Methodology

We propose an architecture for adaptive fault-tolerant approximate multiplier tailored for DNN accelerators. This architecture includes an adaptive adder relying on an unconventional use of input Leading One Detector (LOD) values for fault *detection* and *mitigation* through the optimization of unutilized adder resources. A gate-level optimized LOD design and a lightweight triplicated hybrid adder design are used to enhance further the proposed architecture's reliability, resource utilization, and efficiency. The base for the proposed multiplier is the

classical Mitchell multiplier [156]. However, the methodology can be applied to all logarithmic approximate multipliers. Another level of approximation is introduced in the adaptive adder (Fig. 34) considering the application of this multiplier in DNNs with a proven negligible impact on the network accuracy (see Table 22 in the results section). Mitchell multiplier employs approximate logarithms of the input values. By adding these logarithms, Mitchell's algorithm estimates the product. The final result is obtained by taking the antilogarithm of this sum.

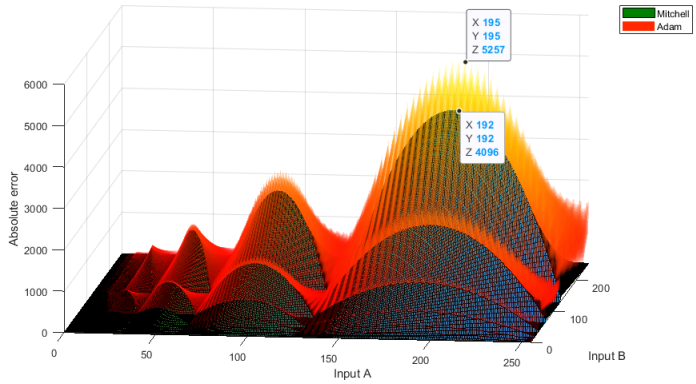


Figure 35: Comparison of absolute error distribution between the original 8-bit Mitchell's multiplier and AdAM: top point shows the maximum absolute error for Adam, lower point shows the maximum absolute error for Mitchell's multiplier

4.2.3.1 Definitions

Consider two n-bit positive integers A and B that can be represented as

$$A = \sum_{i=0}^{k_A} 2^i a_i, B = \sum_{i=0}^{k_B} 2^i b_i, \quad a_i = b_i = \{0,1\} \tag{32}$$

where k is the position of the leading one bit, $0 \leq k < n$. By factoring 2^k we get

$$A = 2^{k_A}\left(1 + \sum_{i=0}^{k_A-1} 2^{i-k_A} a_i\right) = 2^{k_A}(1+X_A) \tag{33}$$

$$B = 2^{k_B}\left(1 + \sum_{i=0}^{k_B-1} 2^{i-k_B} b_i\right) = 2^{k_B}(1+X_B) \tag{34}$$

Since $k \geq 0$, X is in the range $0 \leq X < 1$ and is the fraction term of the number. The logarithms of A and B then can be expressed as

$$log(A) = k_A + log(1+X_A) \tag{35}$$

$$log(B) = k_B + log(1+X_B) \tag{36}$$

Mitchell's method approximates $log(1+X)$ with the value of X. This way, logarithms are approximated as

$$log(A) \approx k_A + X_A \tag{37}$$
$$log(B) \approx k_B + X_B \tag{38}$$

The logarithm of AB can be approximated as

$$log(AB) \approx k_A + X_A + k_B + X_B \tag{39}$$

The antilogarithm of the above expression is the final product:

$$\hat{P} = 2^{k_A + k_B}(1 + X_A + X_B) \tag{40}$$

AdAM architecture introduces a truncation parameter t that defines the level of fault tolerance, i.e. the number of protected higher-order bits of the fractional part. Considering the truncation parameter, fractional part X is defined as

$$X_A^t = \sum_{i=0}^{k_A - 1} 2^{i-k_A} a_i \cdot \left[\text{i-k}_A \geq n - t\right] \tag{41}$$

$$X_B^t = \sum_{i=0}^{k_B - 1} 2^{i-k_B} b_i \cdot \left[\text{i-k}_B \geq n - t\right] \tag{42}$$

where n is the number of bits, t is the truncation parameter, $[\cdot]$ are the Iverson brackets. The final product then is expressed as

$$\hat{P} = 2^{k_A + k_B}(1 + X_A^t + X_B^t) \tag{43}$$

Truncation of the fractional parts introduces additional errors to the result. The absolute error $(P - \hat{P})$ behavior of the original 8-bit Mitchell's multiplier and AdAM is compared in Fig. 35 over the entire input domain. The maximum errors introduced by each method are marked in the figure. There is no additional error when the truncated bits in both operands are '0', as the number of '1' in the truncated bits increases, additional error increases as well.

However, it has been shown that the trained synaptic weights in NN applications do not have a uniform distribution, and they are mostly centered around zero [163]. Therefore, the impact of the proposed multiplier on the accuracy of different networks is reported in the result section.

The level of fault tolerance of the proposed multiplier is defined by two parameters: the aforementioned truncation parameter t and duplication level h. The truncation parameter t defines the minimum number of protected bits, and duplication level h defines the maximum possible number of protected bits. Smaller operands have smaller mantissa values that do not require the whole adder. Therefore, more adder resources can be used for fault tolerance by duplicating the addition of more higher-order bits. Duplication level h does not affect the accuracy of the multiplier, it affects area, power, and delay. A higher h value means a higher level of fault tolerance and higher resource utilization. A

smaller value means a lower level of fault tolerance and lower resource utilization. Different configurations of the proposed multiplier are denoted as AdAM(t,h).

4.2.3.2 Hardware Implementation

The proposed architecture of the multiplier is presented in Fig. 34 (the contributions and extensions to the logarithmic Mitchell multiplier are marked with red color). First, a novel optimized *Leading One Detector (LOD)* circuit (subsection 4.2.3) is used to find the index of the first '1' bit in each operand. This index denoted as k, is the characteristic or integer part of the logarithm and has $log_2(n)$ bits. Then, the operands are shifted left by k bits, aligning the leading one with the Most Significant Bit (MSB). $(n\check{}1)$ bits after the leading one represent the mantissa part denoted as m. The mantissa is truncated to $(n-1-t)$ bits. The truncated operands are passed to the adaptive $(n\check{}1)$-bit

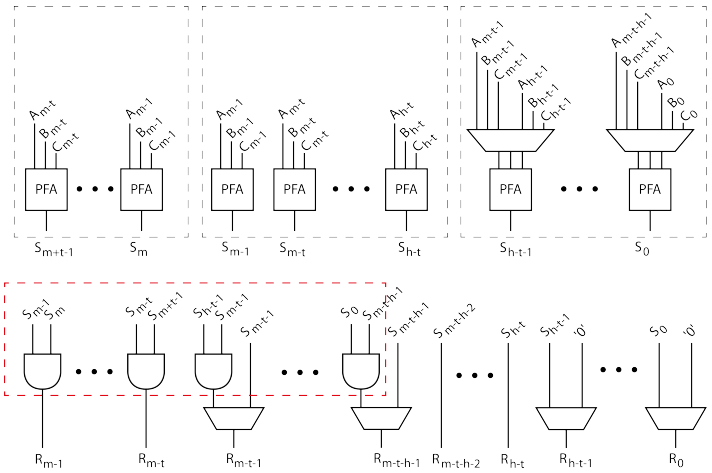


Figure 36: Adaptive adder architecture: A and B are inputs, C is carry values and PFA stands for partial full adder, m here denotes the size of the truncated mantissa $(n-1-t)$

adder that adds mantissa together and duplicates the addition of several higher-order bits depending on the k value of the biggest operand for fault *detection* and *mitigation*. As it was already mentioned, the number of duplicated bits depends on truncation parameter t and duplication level h.

The architecture of the *adaptive adder* is shown in Fig. 36 (subsection 4.2.3). The adder is based on the carry-lookahead adder, and the carry generation logic is excluded from the figure to save space. Duplicated results are compared, and if there is a fault, the faulty bit is set to zero using AND gates (marked on the figure with a red rectangle). Due to the truncation of the mantissa, up to t lower-order bits are excluded from the calculation, which affects only the bigger numbers with the $k > (n-t)$. This introduces a small error compared to the original Mitchell algorithm (discussed in the following section).

The k values of the operands are added separately using a small *hybrid adder*. A hybrid between a resource-intensive but fast carry look-ahead adder and a lightweight but slow due to the accumulated delay ripple carry (CR) adder is selected for this task. This adder is replicated three times, for extra fault

tolerance, as the order of the final output depends on the result of this addition. A majority voter selects the final result. Then, the antilogarithm algorithm is used to get the product of multiplication. The sum of k values determines the position of the leading one in the output product, followed by the sum of the mantissa parts using the appropriate shift operation.

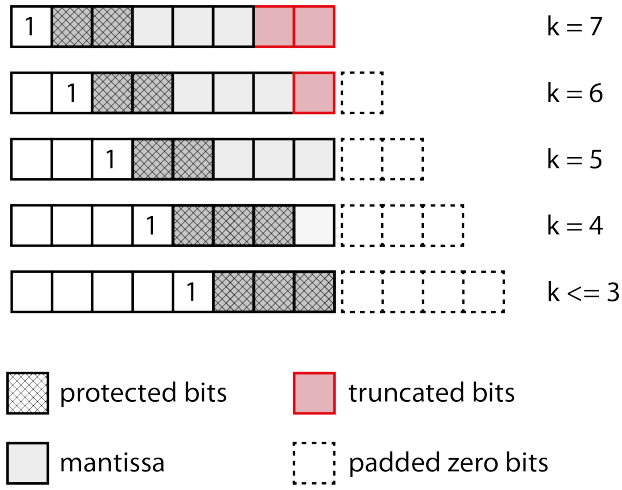


Figure 37: Fault-tolerance and error introduced based on different input values ($n = 8, t = 2, h = 3$)

4.2.3.3 Adaptive Adder

The adaptive adder is designed to detect and mitigate faults based on the multiplier inputs' k values. The adder is divided into three parts: t PFAs (Partial Full Adders) for duplicating the addition of t higher-order bits, $(h-t)$ PFAS with multiplexers on inputs that can duplicate the addition of up to $(h-t)$ higher-order bits or add lower-order bits, and $(n-1-h)$ PFAs for the addition of the mantissa. This way, for smaller operands that have smaller mantissa and do not require the whole adder, more resources can be used for fault tolerance. The size of the truncated mantissa can be expressed as $W_{X^t} = k \cdot [k < n-t] + (n-1-t) \cdot [k \geq n-t]$ using Iverson brackets notation. The available adaptive adder resources can be expressed as $W_{AR} = (n-1) - W_{X^t}$. Thus, the number of protected bits is $W_{PB} = W_{AR} \cdot [W_{AR} < h] + h \cdot [W_{AR} \geq h]$.

As an example, Fig. 37 shows the scheme in which the proposed multiplier introduces fault tolerance considering 8-bit inputs with a truncation parameter $t = 2$ and duplication level $h = 3$. As shown in this figure, five cases are considered. If the maximum LOD of the inputs is 7, two lower-order bits are discarded, and a two-bit adder of the adaptive adder is dedicated to recomputing the addition of two higher-order bits. These results are compared, and the mismatched bits are set to zero.

For $k = 6$, only the Least Significant Bit (LSB) is discarded, and two higher-order bits are protected the same way as in the previous case. When $k = 5$, no bits are discarded, and two higher-order bits are protected. For $k = 4$, three higher-order bits are protected, and only LSB is not monitored. In the case

of $k \leq 3$, all bits are protected, enabling the proposed multiplier to provide comprehensive fault detection and mitigation for all inputs.

4.2.3.4 LOD Design

The gate-level structure of the proposed LOD circuit is presented in Fig. 38. The circuit is divided into two functional parts: zero flag calculation and leading-one position detection (LOPD). LOPD consists of several stages. During the first stage, input is divided into nibbles that are processed in parallel. For each nibble, except for the first one, three signals are calculated: a, b, and c. For the first nibble, only two signals are calculated: b and c. Signal a defines whether there is a '1' bit in the nibble; signal b defines whether the output is even or odd; and signal c defines whether there is '1' in the two higher bits of the nibble. During the second stage, those signals form the final result based on the highest nibble with a '1' bit. Since the zero flag is only required at the last stage of the multiplier to determine whether the final product should be zero, it is calculated using the LOPD output values. The proposed design requires fewer logic gates (as demonstrated in the results section) than designs found in the literature (e.g., ScaleTRIM [155]) by reusing the output values for calculating the zero flag, knowing that it can have a longer delay.

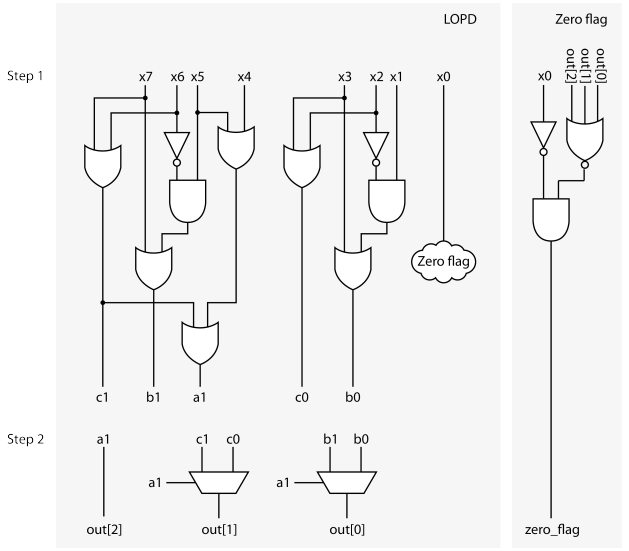


Figure 38: Gate-level structure of the proposed LOD

4.2.4 Experimental Results

Experimental results are reported in this subsection as follows: **4.2.4.1 Experimental Setup**

In this subsection, the FreePDK 45 nm Nangate technology library is used in Cadence Genus 2023 to compare the hardware characteristics (area, latency, power consumption) of the proposed methods with the state-of-the-art. The

accuracy is reported using Mean Absolute Relative Error (MARE) calculated as

$$\text{MARE} = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{P_i - \hat{P}_i}{P_i}\right) \tag{44}$$

where N is the number of tested input combinations, P_i and $\hat{P}_i$ are the exact and approximate results.

Additionally, a design of a MAC (multiply-accumulate) unit in a systolic array is synthesized for ASIC to better illustrate the results in an AI core. The impact on the accuracy of the proposed adaptive multiplier is studied on different networks (i.e., LeNet-5, AlexNet, ResNet-18, VGG-16, DenseNet) trained on MNIST and CIFAR-10 using 8-bit INT with the help of the AdaPT framework [164]. Finally, the impact of the proposed multiplier on the reliability of DNNs is studied using the mentioned benchmarks.

Random fault injection. Fault injection is performed, assuming the single bit-flip faults in the network's MAC operation of a systolic array for reliability assessment. According to the adopted single-bit fault model, a random bit-flip is injected into a random MAC unit of the systolic array core at a random execution time of the network, and the whole test set is fed to the network to obtain the accuracy of the network. This process is repeated several times to reach an acceptable confidence level, following the approach in [54]. This reference provides an equation to reach 95% confidence level and 1% error margin.

4.2.4.2 Hardware Utilization

In this section, the proposed LOD, and adaptive multiplier are compared in terms of power and area with state-of-the-art designs. Power–Delay Product (PDP) is used to show the efficiency of the design.

Table 17 shows the area, power, and delay of the proposed 8-bit LOD architecture compared to the state-of-the-art design. The proposed design has fewer gates and a smaller critical path compared to the other methods in the literature.

Table 17: LOD hardware comparison between the proposed method and the state-of-the-art

LOD Architecture	Delay (ps)	Power (μW)	PDP (ps$\times\mu$W)	Area (μm^2)
ScaleTrim [155]	156	1.12	174.72	9.84
Proposed	136	1.09	148.24	9.57

Tables 18, 19 and 20 report the accuracy, efficiency, and fault tolerance (FT) of 8-bit, 16-bit, and 32-bit approximate multipliers compared with the proposed method. Wallace, DRUM [150], TOSAM [152], and ScaleTrim [155] are used for this comparison. The proposed multiplier has similar hardware parameters to the state-of-the-art approximate multipliers with similar accuracy while providing reliability improvement with fault detection and mitigation capability.

Table 18: Accuracy and efficiency of 8-bit approximate multipliers compared with the proposed method

Multiplier Architecture	Delay (ns)	Power (μW)	Area (μm^2)	MARE (%)	FT	PDP (pJ)
Exact (Wallace)	0.85	360	417	0.00	No	306
DRUM(3)	0.70	104	143	12.6	No	72.8
TOSAM(0,2)	0.58	120	186	10.1	No	69.6
TOSAM(0,3)	0.68	144	198	7.7	No	97.9
DRUM(4)	1.00	172	208	6.4	No	172
TOSAM(1,5)	0.88	231	291	4.1	No	203.2
AdAM(2,3)	1.13	165	152	4.7	Yes	186.4
ScaleTrim(4,8)	1.8	143	216	3.3	No	257.4

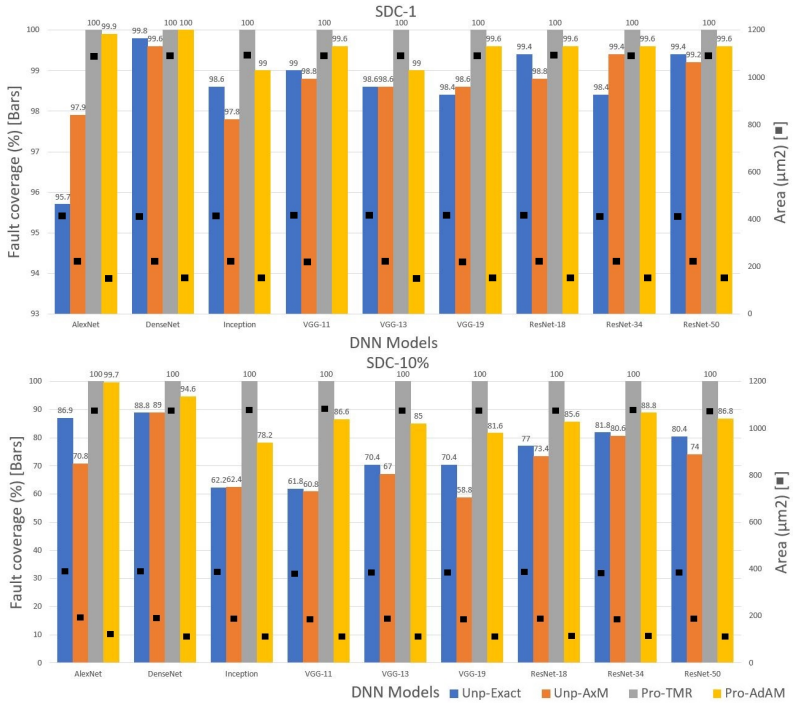


Figure 39: Hardware efficiency (area) and fault resilience (fault coverage considering SDC-1 and SDC-10%) trade-offs in different benchmarks: AlexNet (MNIST), DenseNet(CIFAR), Inception(CIFAR), ResNet-18 (CIFAR), ResNet-34 (CIFAR), ResNet-50 (CIFAR), VGG-11 (CIFAR), VGG-13 (CIFAR), VGG-16 (CIFAR). Unp-Exact: unprotected exact multiplier, Unp-AxM: unprotected approximate multiplier, Pro-TMR: exact multiplier protected with TMR, Pro-AdAM: proposed multiplier

Table 21 shows the results of MAC implementation with different multipliers. The MAC unit with the proposed multiplier takes less area and has a lower PDP

Table 19: Accuracy and efficiency of 16-bit approximate multipliers compared with the proposed method

Multiplier Architecture	Delay (ns)	Power (mW)	Area (μm^2)	MARE (%)	FT	PDP (pJ)
Exact (Wallace)	1.22	2.08	1785	0.00	No	2537.6
DRUM(3)	0.88	0.13	257	11.9	No	114.4
TOSAM(0,2)	0.74	0.16	342	10.9	No	118.4
ScaleTrim(3,4)	1.35	0.20	281	9.23	No	279.0
TOSAM(0,3)	0.84	0.21	423	7.6	No	176.4
DRUM(4)	1.12	0.27	381	5.9	No	302.4
ScaleTrim(7,0)	2.38	0.36	492	4.06	No	871.3
TOSAM(1,5)	1.00	0.35	532	4.0	No	350.0
AdAM(6,7)	1.00	0.10	440	3.97	Yes	100.0
AdAM(4,7)	1.06	0.13	451	3.87	Yes	137.8
AdAM(4,4)	0.96	0.12	434	3.87	Yes	115.2
AdAM(2,7)	1.32	0.15	495	3.97	Yes	171.6
AdAM(2,4)	1.13	0.13	451	3.85	Yes	146.9
DRUM(5)	1.36	0.43	532	2.9	No	584.8
ScaleTrim(9,0)	2.71	0.43	541	2.2	No	1170.6
TOSAM(2,6)	1.21	0.38	564	2.1	No	459.8

value.

4.2.4.3 DNN Accuracy

Table 22 compares the accuracy of different CNN architectures using the proposed approximate multiplier with the baseline accuracy using the exact multiplier. The evaluation shows that the accuracy of DNN with the proposed method is very close to the baseline. Hence, the proposed multiplier has a negligible effect on the accuracy of DNNs.

4.2.4.4 Reliability Analysis

To showcase the impact of the AdAM multiplier on reliability and performance trade-offs, the fault injection simulations are performed on a variety of 8-bit convolutional neural network architectures including AlexNet, DenseNet, Inception, VGG-11, VGG-13, VGG-19, ResNet-18, ResNet-34, and ResNet-50 with four different configurations: unprotected exact multipliers (Unp-Exact), unprotected approximate multipliers (Unp-AxM), protected exact multipliers with TMR (Pro-TMR), and protected approximate multiplier with AdAM (Pro-AdAM). The DNN reliability is evaluated by comparing the output probability vector of the golden run (i.e. the DNN that behaves as expected, without faults) and the faulty run (i.e. the DNN that includes the fault). The SDC rate is defined as the proportion of faults that caused misclassification in comparison with the golden model. Since in DNNs, there is often not a single correct output, but a list of ranked outputs, each with a confidence score [84], we need to use new criteria to determine what constitutes an SDC for a DNN application. Therefore, we consider four types of SDCs as follows:

Table 20: Accuracy and efficiency of 32-bit approximate multipliers compared with the proposed method

Multiplier Architecture	Delay (ns)	Power (mW)	Area (μm^2)	MARE (%)	FT	PDP (pJ)
Exact (Wallace)	1.67	10.26	7618	0.00	No	17134.2
DRUM(3)	1.08	0.20	520	11.90	No	216.0
TOSAM(0,2)	0.99	0.27	844	10.90	No	267.3
TOSAM(0,3)	1.02	0.28	780	7.61	No	285.6
DRUM(4)	1.33	0.36	738	5.90	No	478.8
TOSAM(1,5)	1.18	0.40	999	3.95	No	472.0
AdAM(8,15)	1.67	0.31	1105	3.8488	Yes	517.7
AdAM(8,8)	1.30	0.29	1041	3.8488	Yes	377.0
AdAM(6,15)	1.72	0.48	1148	3.8487	Yes	825.6
AdAM(6,8)	1.46	0.33	1141	3.8487	Yes	471.9
AdAM(4,15)	2.43	0.57	1425	3.8487	Yes	1385.1
AdAM(4,8)	1.80	0.46	1403	3.8487	Yes	828.0
AdAM(2,15)	2.49	0.59	1579	3.8487	Yes	1469.1
AdAM(2,8)	2.02	0.49	1519	3.8487	Yes	989.8
DRUM(5)	1.55	0.56	944	2.89	No	868.0
TOSAM(2,6)	1.30	0.54	1146	2.06	No	702.0
DRUM(6)	1.69	0.75	1059	1.47	No	1267.5
TOSAM(3,7)	1.44	0.69	1294	1.05	No	993.6
DRUM(7)	1.85	0.96	1235	0.73	No	1776.0
TOSAM(4,8)	1.57	0.83	1411	0.53	No	1303.1
DRUM(8)	1.93	1.17	1402	0.37	No	3545.2
TOSAM(5,9)	1.60	1.08	1625	0.27	No	1728.0

- **SDC-1:** Misclassification in the top-ranked output class.
- **SDC-5%:** More than 5% of variation in the top-ranked output confidence score compared to the golden model.
- **SDC-10%:** More than 10% of variation in the top-ranked output confidence score compared to the golden model.
- **SDC-20%:** More than 20% of variation in the top-ranked output confidence score compared to the golden model.

Fig. 39 demonstrates the fault tolerance comparison and reliability improvement (for SDC-1 and SDC-10% as two examples) of different networks by using the protected approximate multiplier proposed in this work compared to the unprotected exact and approximated networks, and protected networks using TMR. As illustrated, TMR has 100% of protection, but it also requires about 200% of area overhead. Different from TMR, in our technique we introduce a high-reliability improvement without introducing hardware overhead. The results

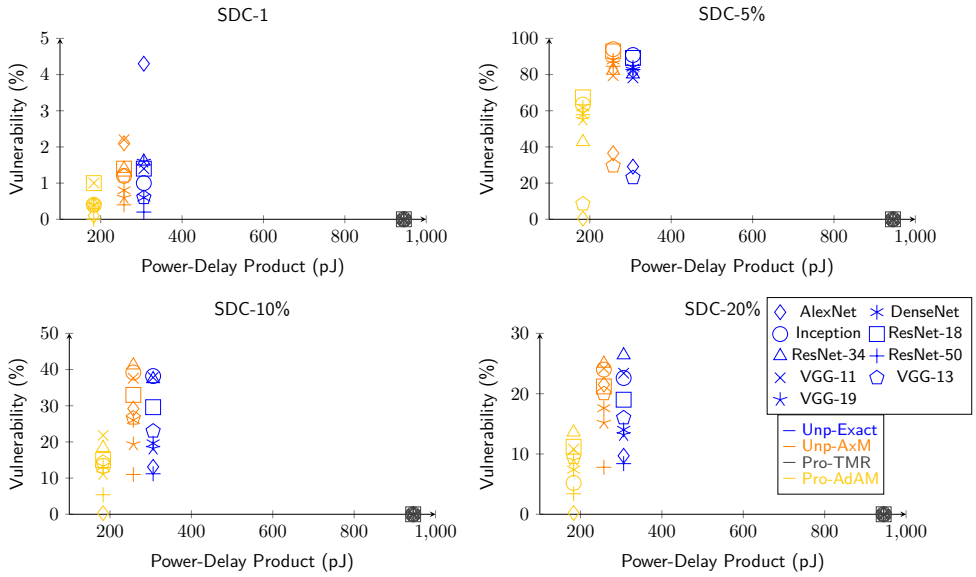


Figure 40: PDP and vulnerability tradeoffs (considering different SDCs) in different benchmarks: AlexNet, DenseNet, Inception, ResNet-18, ResNet-34, ResNet-50, VGG-11, VGG-13, VGG-16. Unp-exact: unprotected exact multiplier, Unp-AxM: unprotected approximated multiplier, Pro-TMR: exact multiplier protected with TMR, Pro-AdAM: proposed multiplier

for reliability improvement considering SDC-5% and SDC-20% are reported in Tables 23 and 24. Since the main objective of the proposed multiplier is to have the best trade-off in PDP and vulnerability, Fig. 40 illustrate these comparisons based on different vulnerability metrics (SDC-10, SDC-5%, SDC-10% and SDC-20%). In these charts, the closer to the origins (0,0), the higher the cost-efficiency of the fault tolerance, i.e. less vulnerability and less PDP. As shown, TMR is a less efficient solution for edge AI applications because of its high PDP, while the proposed method (AdAM) is the closest to the origin.

4.2.5 Conclusion

In this subsection, we propose an architecture of a novel adaptive fault-tolerant approximate multiplier tailored for ASIC-based DNN accelerators. AdAM employs an adaptive adder that relies on an unconventional use of input Leading One Detector (LOD) values for fault detection by optimizing unutilized adder resources. A gate-level optimized LOD design is also proposed to improve the hardware performance as part of the adaptive multiplier. The proposed architecture uses a lightweight fault mitigation technique that sets the detected faulty bits to zero.

It is demonstrated that the proposed architecture enables a multiplication with a reliability level close to the multipliers protected by TMR while at the same time utilizing 2.74 $\times$ less area and with 39.06% less power-delay product compared to the exact multiplier.

Table 21: Efficiency of 8-bit MAC unit of a systolic array with different multipliers

MAC Architecture	Delay (ns)	Power (mW)	PDP (ns×mW)	Area (μm^2)
Exact (Wallace)	1151	2.10	2417.10	975.95
ScaleTrim(4,8)	1106	2.00	2212.00	949.35
AdAM(2,3)	1098	1.47	1614.06	794.80

Table 22: Accuracy comparison of different CNNs with an exact (baseline) and the proposed approximate multiplier

DNN	Accuracy with Wallace (%)	Accuracy with AdAM(2,3) (%)
LeNet-5 (MNIST)	93.8	94.1
AlexNet (MNIST)	78.0	77.7
VGG-11 (CIFAR-10)	93.4	94.0
VGG-13 (CIFAR-10)	92.0	92.0
VGG-19 (CIFAR-10)	92.0	93.0
ResNet-18 (CIFAR-10)	93.8	93.2
ResNet-34 (CIFAR-10)	93.0	94.0
ResNet-50 (CIFAR-10)	95.0	94.0
DenseNet (CIFAR-10)	92.6	95.0
Inception (CIFAR-10)	92.6	95.0

4.3 FORTUNE: A Negative Memory Overhead Hardware-Agnostic Fault TOleRance TechniqUe in DNNs

This subsection is based on the following paper: [VII]

4.3.1 Introduction

As real-time data processing and AI demand grow in embedded systems, Quantized Deep Neural Networks (QNNs) have become vital for deploying models efficiently in resource-constrained environments, spanning applications from image classification to safety-critical applications like autonomous driving XV, IX. QNNs perform complex computations with reduced precision, minimizing computational and memory footprints, and achieving significant energy savings, crucial for energy-constrained platforms VI.

However, quantization introduces challenges in neural network accuracy, especially in critical applications like autonomous systems or medical diagnostics, where precision is non-negotiable I, XI. Post-training quantization (PTQ) methods address these challenges, preserving model integrity while reducing bit-widths to enhance efficiency [165]. Yet, QNNs' reliance on extensive memory resources makes them vulnerable to faults, particularly as transistors miniaturize [166]. Ensuring fault tolerance is essential, as even minor errors can significantly reduce

Table 23: Fault coverage (SDC-5%) in different benchmarks

SDC-5%	AlexNet	DenseNet	Inception	VGG-11	VGG-13	VGG-19	ResNet-18	ResNet-34	ResNet-50
Unp-Exact	70.9	17.2	21.8	9.2	10.8	10.8	10.8	17.2	16.2
Unp-AxM	63.5	15.4	20.6	6	7.2	17.8	9.4	13.6	12.2
Pro-TMR	100	100	100	100	100	100	100	100	100
Pro-AdAM	99.6	42.2	41.8	36.6	32.6	57.2	37.6	44.8	38.2

Table 24: Fault coverage (SDC-20%) in different benchmarks

SDC-20%	AlexNet	DenseNet	Inception	VGG-11	VGG-13	VGG-19	ResNet-18	ResNet-34	ResNet-50
Unp-Exact	90.3	91.6	76.6	77.4	81	81	84	86.8	86
Unp-AxM	78.5	92.2	75.8	76	78.8	75	80	84.8	82.4
Pro-TMR	100	100	100	100	100	100	100	100	100
Pro-AdAM	99.8	96.6	89.2	94.8	88.8	86.4	90.6	91	92.6

accuracy X. Figure 41 highlights fault locations in critical GPU architecture points. Faults can arise from temperature fluctuations, radiation, aging circuits, and electromagnetic interference. Enhancing fault tolerance in QNNs is thus crucial for safe deployment in safety-critical systems IX, XII.

Traditional fault-tolerant techniques like Triple Modular Redundancy (TMR) introduce significant computational overhead [168]. Selective hardening approaches focus on protecting parameters or neurons with greater impact on the network's output [169]. These methods, however, are mainly applicable to FPGA and ASIC platforms where hardware modifications are possible [57]. For general-purpose CPUs, GPUs, or fixed accelerators, hardware modifications are often infeasible.

Other methods, such as pruning corrupted parameters and retraining [11], fault-aware training [126], and Error Correction Codes (ECC) [170], introduce significant memory and computational overhead. Activation restriction techniques mitigate error propagation but are ineffective at high error rates [171]. Model-level hardening through selective duplication also incurs substantial memory overhead and requires additional layers for error correction [172].

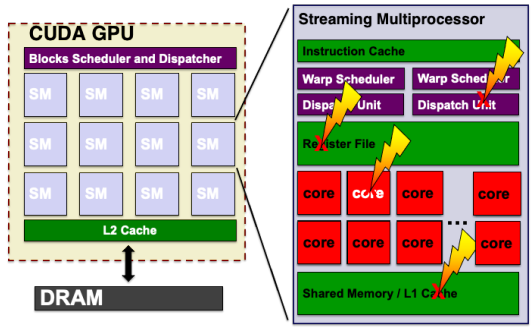


Figure 41: Fault locations in critical points of a parallel architecture such as the GPUs [167]

To address the extensive memory overhead and performance degradation challenges, we present FORTUNE, a model-level hardware agnostic methodology that explores different quantization levels of a DNN model, focusing on reliability, accuracy, memory overhead, and performance aspects. The proposed methodology introduces a novel fault tolerance technique that uses memory savings from quantization to the Most Significant Bit (MSB) within the same memory elements, maintaining QNN reliability even in the presence of memory faults. A comprehensive GPU-based framework implements this methodology. Specifically, the proposed fault tolerance methodology integrates into the framework's iterative optimization loop, allowing users to define thresholds that balance model accuracy, reliability, and performance. The extensive experiments show that, without this protection, QNNs suffer significant accuracy degradation in fault-prone environments. The results underscore the importance of the proposed technique to mnimizes memory utilization and optimizes protected QNN execution time on DNN accelerators. In this work, without loss of generality, we reported the results for GPU as an example.

The key contributions of this paper are:

- **Negative Overhead Fault Tolerance Technique**: Proposed a protection technique leveraging quantization to triplicate MSB, ensuring robustness against faults.
- **Design Space Exploration Framework**: Developed an open-source framework to assess the impact of quantization on QNN reliability, accuracy, memory utilization, and executing time for design space exploration based on a binary search algorithm.
- **Introduction of P_{drop} and Reliability-Aware Performance (RAP) metrics**: Introduced P_{drop}, the probability of accuracy drop over a device's lifetime with various BERs, and RAP, a metric for evaluating trade-offs in fault-prone and resource-constrained environments.
- **Validation**: Evaluated the proposed technique and framework on state-of-the-art DNN benchmarks.

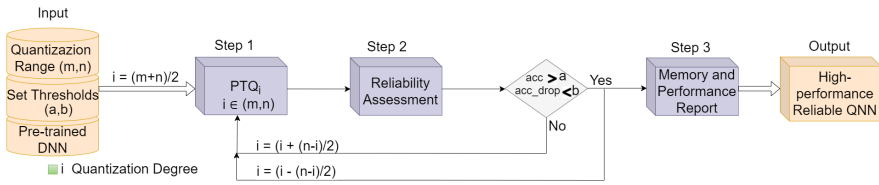


Figure 42: Proposed FORTUNE methodology for quantizing a DNN model, evaluating its reliability, and applying the protection.

The remainder of this section is structured as follows: subSection 4.2.2 presents the proposed methodology. subSection 4.2.3 discusses experimental results and the impact on resilience, memory utilization, and execution time. Finally, subSection 4.2.4 concludes the section.

4.3.2 Proposed Methodology

Figure 42 illustrates the systematic methodology for quantizing a DNN model and evaluating its reliability with different fault situations and the proposed protection. The steps of this methodology are explained by the algorithm 1 and also in details in this section. The goal is to achieve a quantized model that not only meets predefined accuracy and reliability thresholds but also provides insights into memory and performance overheads. The algorithm begins by getting a trained FP32 DNN model as input, along with three key parameters: an accuracy threshold a, a reliability threshold b, and a quantization range $[m, n]$ (Algorithm 1 - line 4-6). The accuracy threshold represents the minimum acceptable accuracy of the model after quantization, while the reliability threshold b specifies the maximum permissible drop in accuracy due to potential faults. The quantization range $[m, n]$ defines the range of bit widths to be explored during the quantization process.

Input: Trained FP32 DNN model, accuracy threshold a, reliability threshold b, quantization range $[m, n]$
Output: High-performance Reliable QNN
Load the trained DNN;
SET accuracy_threshold TO a;
SET reliability_threshold TO b;
SET quantization range TO $[m, n]$;
$bit\_width = \frac{n+m}{2}$;
process = True;
while *process* **do**
  quantized_weights ← QuantizeWeights(model.weights, bit_width);
  golden_accuracy ← EvaluateAccuracy(model, quantized_weights);
  if $accuracy > a$ *AND* $accuracy\_drop < b$ **then**
    Inject faults to weights of the model with different BERs;
    accuracy ← EvaluateAccuracy(model, quantized_weights);
    reliability_drop ← golden_accuracy − accuracy;
    memory_overhead ← CalculateMemoryOverhead(bit_width, protected_bits);
    performance_overhead ← CalculatePerformanceOverhead(bit_width, protected_bits);
    $bit\_width = bit\_width - \frac{n - bit\_width}{2}$;
  else
    $bit\_width = bit\_width + \frac{n - bit\_width}{2}$;
  end
  if $bit\_width > n$ **then**
    process = False;
  end
end

Algorithm 1: Fortune Algorithm

Step 1: Quantization. The algorithm iterates over each bit width within the specified quantization range. To perform a binary search, the initial bit

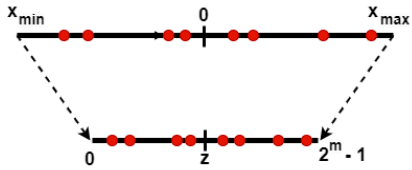


Figure 43: Affine linear quantization

width considered is the midpoint of the quantization range. Based on the results obtained at this midpoint, the next bit width for weights is selected from either the upper or lower half of the range, depending on whether the desired accuracy and reliability criteria are met (Algorithm 1 - lines 7, 18 and 20). This process is repeated iteratively, narrowing the search range until the optimal bit width is identified. For each selected bit width, the model weights are quantized through linear quantization (Algorithm 1 - line 10). Linear quantization is a widely used technique in model compression, particularly in the context of QNNs. It reduces the precision of weights by mapping a large range of values (typically 32-bit floating-point) to a smaller, fixed range represented by fewer bits. This process involves two main steps: scaling and rounding. The first step in linear quantization is to define the range $[x_{min}, x_{max}]$ within which all the values of the tensor will be mapped.

$$x_{min} = min(x) \tag{45}$$

$$x_{max} = max(x) \tag{46}$$

Next, the scaling factor s is calculated. This factor determines how the original floating-point values are scaled down to fit within the quantized range. For a given bit width m, the quantized range is $[0, 2^m - 1]$. The scaling factor s is computed as:

$$s = \frac{x_{max} - x_{min}}{2^m - 1} \tag{47}$$

As Figure 43 indicates, the tensor values are then quantized by mapping each value x_i to an integer value q_i within the range $[0, 2^m - 1]$ using the scaling factor s and a zero-point z:

$$z = -\frac{x_{min}}{s} \tag{48}$$

$$q_i = round(\frac{x_i}{s} + z) \tag{49}$$

Here, $round(.)$ denotes rounding to the nearest integer. The result is an integer value q_i that lies within the quantized range. Therefore, weights of the model are converted into unsigned m-bit integers through affine linear quantization.

Then, the accuracy of the model with quantized weights, referred to as the "golden accuracy," is evaluated. This accuracy serves as a baseline for further reliability testing (Algorithm 1 - line 11). If the golden accuracy falls below the predefined threshold a, the algorithm skips further evaluation for this bit width and proceeds to the next (Algorithm 1 - lines 19, 20).

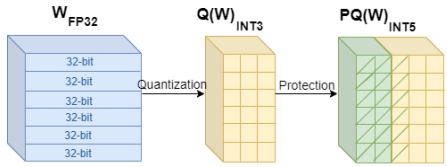


Figure 44: An example of protected 3-bit weights

Table 25: Benchmark NNs base accuracies (%)

Type	VGG11	ResNet18	AlexNet	Inception
8-bit	92.41	93.06	95.32	93.70
5-bit	92.19	92.81	95.46	93.24
4-bit	92.18	92.66	94.91	92.03
3-bit	89.83	90.84	93.26	80.71

Step 2: Reliability Evaluation and Enhancement. Then, the algorithm proceeds to evaluate its reliability under fault conditions. Faults are incrementally injected into the quantized model's weights, simulating different Bit Error Rates (BERs). After each fault injection, the model's accuracy is re-evaluated, and the accuracy drop is calculated as the difference between the golden accuracy and the current accuracy (Algorithm 1 - lines 13-15).

In the proposed approach, only the MSB bit is protected by replicating it in two redundant bits. During inference, these redundant bits, along with the protected bit, undergo a majority voting process to determine the final value of the protected bit. This method ensures that even with potential faults, the most critical bit remains reliable.

Figure44 illustrates an example of 3-bit weights with a protected bit. In this example, the FP32 weights are first converted into unsigned 3-bit integer values (shown in orange), and the MSB is replicated into two redundant bits (shown in green). Similarly, in 5-bit weights with one protected bit, two redundant bits are added. More generally, in i-bit quantization with one protection bits, two redundant bits are added, and one comparison is performed during inference.

Step 3: Memory Utilization and Execution Time. In this step, the algorithm computes the memory utilization and execution time associated with the given bit width (Algorithm 1 - lines 16, 17). These values provide insights into the trade-offs between quantization, reliability, and resource utilization. The algorithm repeats the above steps for all bit widths within the specified range, ultimately outputting a set of reliable and quantized networks. For each network, detailed reports on execution time and memory utilization are generated, allowing for an informed selection of the optimal quantization configuration.

P_{drop} **and** RAP

Accuracy drop is evaluated independently of any physical effects that faults might have on memory. To account for these effects on accuracy drop, we define P_{drop} as the probability of experiencing the accuracy drop during the device's lifetime:

$$P_{drop} = N^2 \times W^2 \times T/t \times P_{single} \times BER \times acc\_drop \tag{50}$$

Table 26: Memory Utilization, Execution Time, Vulnerability (accuracy drop due to fault injection) and P_{drop} in DNNs.

Model	Type	Memory Utilization	Executuin Time (%)	Vulnerability (%) {BER}				P_{drop} {BER}			
				1.00E-5	3.00E-5	1.00E-4	3.00E-4	1.00E-5	3.00E-5	1.00E-4	3.00E-4
VGG11	8-bit	225,064,448	100	5.62	22.30	62.40	74.52	50.21E-3	150.63E-3	502.10E-3	1506.30E-3
	P-5-bit	196,931,392	141.72	1.11	5.23	28.33	68.73	35.45E-3	106.35E-3	354.50E-3	1063.52E-3
	P-4-bit	168,798,336	141.72	1.45	6.01	28.19	67.99	25.76E-3	77.30E-3	257.66E-3	773.00E-3
	P-3-bit	140,665,280	141.72	1.49	5.35	34.20	67.13	17.66E-3	53.00E-3	176.68E-3	530.05E-3
ResNet18	8-bit	66,991,616	100	0.18	1.85	14.64	32.29	1.92E-3	5.78E-3	19.27E-3	57.82E-3
	P-5-bit	58,617,664	119.54	0.16	0.81	3.54	18.77	0.85E-3	2.57E-3	8.58E-3	25.74E-3
	P-4-bit	50,243,712	119.54	0.24	0.66	2.63	18.34	0.61E-3	1.84E-3	6.15E-3	18.47E-3
	P-3-bit	41,869,760	119.54	0.25	0.79	3.74	21.89	0.51E-3	1.53E-3	5.10E-3	15.31E-3
AlexNet	8-bit	466,316,032	100	0.00	0.49	3.14	30.71	88.82E-3	266.47E-3	888.23E-3	2664.70E-3
	P-5-bit	408,026,528	102.94	0.14	0.04	0.48	4.73	10.47E-3	31.43E-3	104.78E-3	314.46E-3
	P-4-bit	349,737,024	102.94	0.19	0.10	0.52	5.30	8.62E-3	25.87E-3	86.24E-3	258.72E-3
	P-3-bit	291,447,520	102.94	0.27	0.58	2.62	13.58	15.34E-3	46.03E-3	153.44E-3	460.33E-3
Inception	8-bit	173,011,456	100	0.09	0.16	0.45	1.43	0.57E-3	1.71E-3	5.71E-3	17.15E-3
	P-5-bit	151,234,496	291.73	0.00	0.02	0.02	0.19	0.06E-3	0.18E-3	0.60E-3	1.80E-3
	P-4-bit	129,758,592	291.73	0.01	0.05	0.08	0.14	0.03E-3	0.09E-3	0.33E-3	0.99E-3

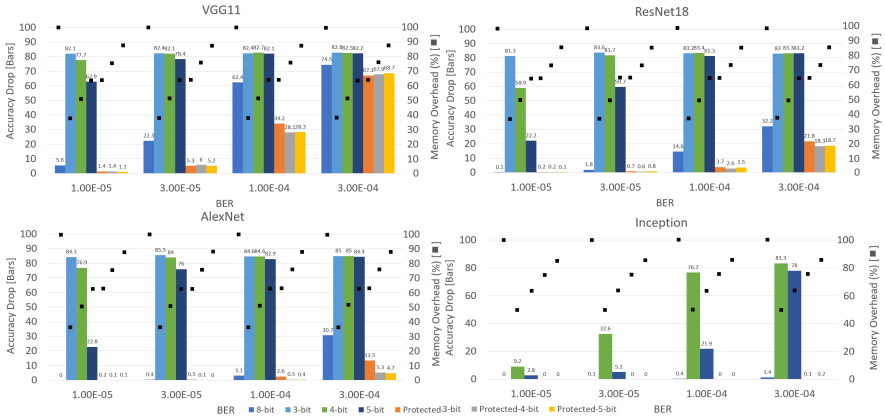


Figure 45: Vulnerability (accuracy drop due to fault injection) and memory utilization trade-offs in different benchmarks: VGG-11 (CIFAR), ResNet-18 (CIFAR), AlexNet (FashionMNIST) and Inception(CIFAR).

where N is the number of parameters, W is their bit width, T is device life time, t is test time interval, P_{single} is the probability of one bit flip during t and acc_drop is the reported accuracy drop in the BER. This metric is based on the probability of a one-bit flip stated in [173]. As the definition suggests, the more resilient the networks are, the smaller the value of P_{drop} becomes.

To account for performance along with accuracy drop and memory footprint, we define the Reliability-Aware Performance (RAP) metric as:

$$RAP = acc\_drop \times mem\_ovh \times perf\_ovh \tag{51}$$

where acc_drop represents the accuracy drop, $perf\_ovh$ refers to the execution time overhead, and mem_ovh denotes the memory utilization overhead. Smaller RAP values indicate more reliable networks with lower memory and performance overhead.

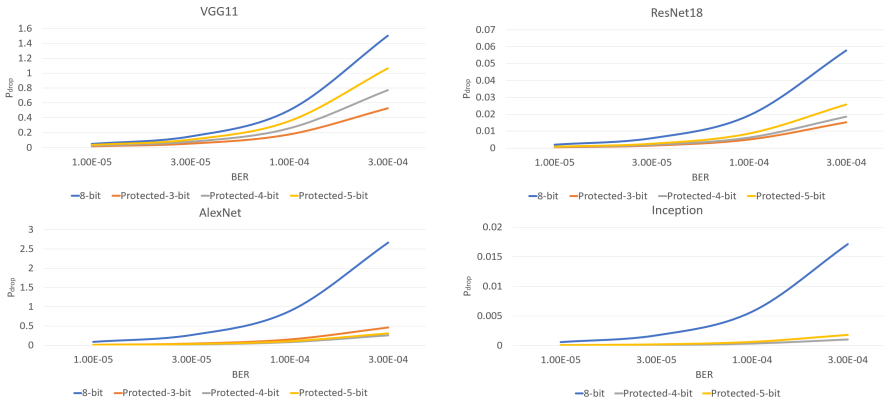


Figure 46: P_{drop} in VGG-11 (CIFAR), ResNet-18 (CIFAR), AlexNet (FashionMNIST) and Inception(CIFAR).

4.3.3 Experimental Results

Experimental results are reported in this subsection as follows: **4.3.3.1** Experimental Setup:

To evaluate the reliability and performance of the proposed method, we study four different neural networks. AlexNet is trained on the Fashion MNIST dataset, while VGG11, ResNet-18, and Inception are trained on the CIFAR-10 dataset. Quantization and reliability evaluations are conducted for all networks, resulting in the introduction of different reliable versions for each model. Then, the memory utilization and execution time associated with FORTUNE is assessed. To further quantify the effectiveness of the results, P_{drop} and RAP values are reported for each network to highlight the trade-offs involved.

4.3.3.2 Quantization and reliability evaluation:

To facilitate comparison across different networks, the results for four different bit widths are presented in this section. The unprotected 8-bit version is used as the baseline network. As previously mentioned, the weights are quantized using affine linear quantization, and no retraining is performed. The accuracy of the different quantized networks is presented in Table 25. Notably, the accuracy of Inception in its 3-bit version is 80.71%, representing an accuracy reduction of over 10%.

To evaluate reliability, a random fault injection is conducted across all weights in the DNNs under study. The number of injected faults is determined using a BER ranging from 10^{-5} to 3×10^{-4}, covering a comprehensive range of potential errors X. Fault injection is repeated several times to reach an acceptable confidence level, following the approach in [54]. This reference provides an equation to reach a 95% confidence level and 1% error margin. For each bit width and BER, the resulting drop in accuracy (with respect to the corresponding fault-free model) is reported in Table 26 as Vulnerability of the networks. Although the protected versions of AlexNet and ResNet-18 do not exhibit significant differences in Vulnerability values at lower BERs, a considerable difference becomes apparent at higher BERs. Conversely, VGG11 and Inception experience less vulnerability across all protected versions and BER levels. Additionally, the unprotected

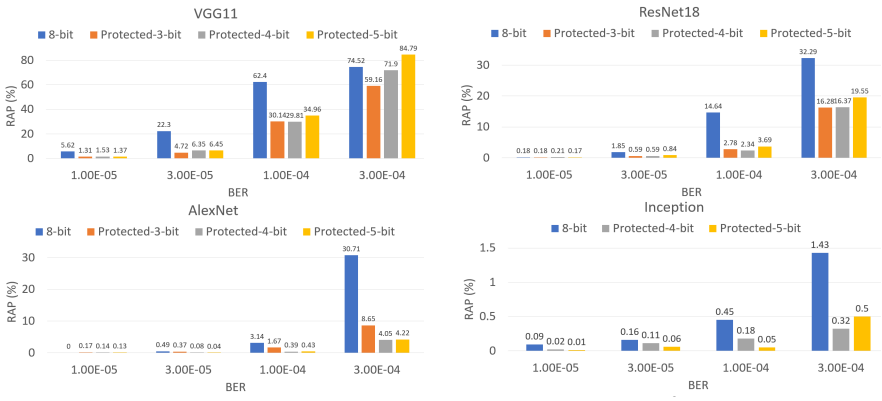


Figure 47: RAP in VGG-11 (CIFAR), ResNet-18 (CIFAR), AlexNet (FashionMNIST) and Inception(CIFAR) .

versions of the quantized networks show worst vulnerabilities, as illustrated in Figure 45.

4.3.3.3 Reliability, memory and performance trade-off:

Memory utilization is defined as the combination of the number of parameters and their respective bit widths, while execution time refers to the execution time of each network under study. Memory utilization and execution time are reported in Table 26, where execution time is normalized with respect to the unprotected 8-bit model. Figure 45 illustrates the trade-off between vulnerability and memory utilization for all protected and unprotected versions of the quantized networks. As the BER increases, all protected models exhibit lower vulnerability compared to the unprotected 8-bit model, while also utilizing less memory. For example, while the unprotected 5-bit model has the same memory utilization as the protected 3-bit model, the protected model demonstrates significantly lower vulnerability. To clarify, all quantized models (5-bit, 4-bit, and 3-bit) are significantly more vulnerable than their protected counterparts. Moreover, the protected versions still maintain lower memory utilization compared to the base 8-bit model.

P_{drop} values for the DNNs under study are reported in Table 26 and Figure 46. All final networks are considered more resilient when evaluated based on P_{drop}. This indicates that, when factoring in both memory footprint and vulnerability, all protected quantized networks exhibit greater resilience throughout their lifetime.

RAP values are reported in Figure 47. Except for VGG11 at the highest BER, all other networks demonstrate smaller RAP values in protected quantized versions, indicating better trade-off between reliability, memory utilization, and execution time. Smaller RAP values indicate that the networks introduced by FORTUNE are more resilient, while also requiring less memory and execution time compared to the base 8-bit models.

4.3.4 Conclusion

This section introduces FORTUNE, a novel framework designed to enhance fault tolerance in DNNs through quantization, offering a balanced trade-off between reliability, memory usage, and execution time. By leveraging memory savings

to protect the critical Most Significant Bit, FORTUNE improves fault tolerance without the high computational costs of conventional methods like TMR. As examples, we demonstrated memory reductions of 37.5% across networks, with vulnerability in AlexNet reduced by 56% compared to the 8-bit model and 84% compared to the unprotected 3-bit model. These improvements were achieved with less than a 3% increase in execution time. FORTUNE proposes a flexible framework that allows for the identification of the most suitable network configuration, optimizing the trade-off between reliability, memory efficiency, and execution time based on specific application requirements.

4.4 Chapter Conclusions

This chapter introduces two techniques to enhance the fault tolerance of DNN hardware accelerators: AdAM and FORTUNE. AdAM, through its adaptive approximation strategy, reduces power and area overhead while maintaining high levels of accuracy under fault conditions. Meanwhile, FORTUNE addresses the memory overhead challenge in fault tolerance, using a unique method to protect critical data bits, leading to reduced memory consumption and increased overall resilience. These methods outperform traditional fault tolerance strategies such as full Triple Modular Redundancy (TMR) by achieving comparable reliability with much lower resource costs. The techniques presented in this chapter provide a foundation for deploying reliable, low-power DNN accelerators in critical applications such as autonomous vehicles and AI-driven medical devices.

5 Conclusions and Future Directions

This thesis delves into both reliability assessment and enhancement of DNNs, with a particular focus on hardware accelerators. It presents a comprehensive analysis of how faults affect DNNs in hardware implementations such as FPGAs and ASICs and proposes innovative methods to mitigate these reliability issues.

Reliability Assessment: The thesis introduces various FI methods, analytical models, and hybrid techniques to evaluate the reliability of DNN hardware accelerators. The proposed frameworks, assess reliability across different abstraction levels. These methods evaluate fault propagation in DNN components and quantify their vulnerability using metrics like FIT rate and SDC rate. By combining software-level and hardware-level evaluations, the thesis provides a detailed picture of DNN behavior under fault conditions.

Comparison of Techniques: The thesis compares the proposed reliability assessment techniques:

- **FI Methods**: While FI methods provide the highest accuracy, simulating real-world fault scenarios, they are slower. For instance, in the RT-level simulations using QuestaSim, the fault injection rate reached only **0.007 simulations per second**, which is **2100× slower** than the hybrid method.

- **Analytical Methods**: Analytical models offer a significant speedup—up to **86% faster**—by skipping redundant simulations, but their accuracy is slightly compromised as they may not capture all fault interactions.

- **Hybrid Methods**: Combining analytical pruning with FI, hybrid methods strike a balance, achieving up to **3× speed-up** over traditional approaches and **2000× faster** than RT-level FI, with near-FI accuracy, making them the most efficient for large-scale reliability assessments.

Reliability Enhancement: Building on the assessment phase, the thesis also proposes methods to enhance DNN reliability. The AdAM (Adaptive Approximate Multiplier) and FORTUNE techniques introduce fault tolerance mechanisms at both architectural and hardware levels. These frameworks optimize resource utilization while maintaining acceptable levels of accuracy, offering solutions for improving fault resilience with minimal overhead. Techniques such as selective redundancy, approximate computing, and fault-tolerant multipliers are introduced to ensure that critical applications—such as autonomous driving and edge AI—achieve higher reliability without compromising performance. Key contributions to reliability enhancement include AdAM, a fault-tolerant approximate multiplier that balances computational complexity with reliability, enabling DNN accelerators to maintain accuracy despite faults; And FORTUNE, a hardware-agnostic fault tolerance technique that reduces memory overhead by using quantization while improving resilience in critical bits of DNN hardware.

Integrating Reliability Assessment and Enhancement: The reliability assessment and enhancement frameworks presented in this thesis are interconnected. Assessment tools such as fault injection and analytical methods identify critical components and failure points in DNN hardware. This information informs the design of fault-tolerant architectures like AdAM and FORTUNE, targeting the

most vulnerable components for maximum resilience with minimal performance degradation.

The contributions of this thesis can be summarized as follows:

- Development of novel reliability assessment frameworks for DNN hardware accelerators, incorporating fault injection, analytical, and hybrid methods.
- Proposal of fault tolerance techniques that balance performance, power consumption, and reliability, including fault-tolerant designs for DNN accelerators.
- Exploration of approximation and reliability trade-offs in DNN hardware to improve energy-efficient, reliable accelerator design.
- Introduction of adaptive methods like AdAM and FORTUNE that allow DNN accelerators to operate efficiently in fault-prone environments, particularly in safety-critical applications.
- Experimental validation of the proposed frameworks, demonstrating their effectiveness in real-world DNN accelerator implementations.

Ultimately, this thesis offers a holistic approach to DNN reliability by combining in-depth assessments with innovative enhancement strategies, providing a robust solution for improving the resilience of DNNs in critical applications.

Future Directions

Looking ahead, several avenues of research remain open to further enhance the robustness of DNN accelerators. First, the current frameworks could be extended to cover a broader range of DNN architectures beyond Convolutional Neural Networks (CNNs), such as Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) networks. Moreover, the reliability of DNNs during the training phase remains underexplored, and future studies should investigate the impact of faults on both model parameters and computational units during this phase.

Another promising direction lies in developing more sophisticated analytical methods that can complement fault injection techniques, leading to faster and more scalable reliability assessments. In particular, machine learning (ML) algorithms integrated with these frameworks could further improve the accuracy and efficiency of reliability predictions. Additionally, hybrid methods that combine fault injection and analytical techniques could play a crucial role in developing comprehensive and adaptable reliability assessment models for a wider range of applications and platforms, including edge AI and autonomous systems.

Future research should also investigate the control components of hardware accelerators, particularly in FPGAs and ASICs, as these components are crucial to overall system reliability but remain largely unexplored. Lastly, the development of hardware-aware fault injection simulators could significantly improve the real-time assessment of DNN reliability, providing deeper insights into the interactions between software and hardware in critical applications

List of Figures

List of Tables

References

[1] A. Bosio, I. O'Connor, M. Traiola, J. Echavarria, J. Teich, M. A. Hanif, M. Shafique, S. Hamdioui, B. Deveautour, P. Girard, *et al.*, "Emerging computing devices: Challenges and opportunities for test and reliability," in *2021 IEEE European Test Symposium (ETS)*, pp. 1–10, IEEE, 2021.

[2] H. Forsberg, J. Lindén, J. Hjorth, T. Månefjord, and M. Daneshtalab, "Challenges in using neural networks in safety-critical applications," in *2020 AIAA/IEEE 39th Digital Avionics Systems Conference (DASC)*, pp. 1–7, IEEE, 2020.

[3] A. Nardi and A. Armato, "Functional safety methodologies for automotive applications," in *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 970–975, IEEE, 2017.

[4] Y. Ibrahim, H. Wang, M. Bai, Z. Liu, J. Wang, Z. Yang, and Z. Chen, "Soft error resilience of deep residual networks for object recognition," *IEEE Access*, vol. 8, pp. 19490–19503, 2020.

[5] M. Shafique, M. Naseer, T. Theocharides, C. Kyrkou, O. Mutlu, L. Orosa, and J. Choi, "Robust machine learning systems: Challenges, current trends, perspectives, and the road ahead," *IEEE Design & Test*, vol. 37, no. 2, pp. 30–57, 2020.

[6] S. Mittal, "A survey on modeling and improving reliability of dnn algorithms and accelerators," *Journal of Systems Architecture*, vol. 104, p. 101689, 2020.

[7] A. Ruospo, E. Sanchez, L. M. Luza, L. Dilillo, M. Traiola, and A. Bosio, "A survey on deep learning resilience assessment methodologies," *Computer*, vol. 56, no. 2, pp. 57–66, 2023.

[8] F. Su, C. Liu, and H.-G. Stratigopoulos, "Testability and dependability of ai hardware: Survey, trends, challenges, and perspectives," *IEEE Design & Test*, 2023.

[9] C. Torres-Huitzil and B. Girau, "Fault and error tolerance in neural networks: A review," *IEEE Access*, vol. 5, pp. 17322–17341, 2017.

[10] S. Burel, A. Evans, and L. Anghel, "Mozart: Masking outputs with zeros for architectural robustness and testing of dnn accelerators," in *2021 IEEE 27th International Symposium on On-Line Testing and Robust System Design (IOLTS)*, pp. 1–6, IEEE, 2021.

[11] K. T. Chitty-Venkata and A. K. Somani, "Model compression on faulty array-based neural network accelerator," in *2020 IEEE 25th Pacific Rim International Symposium on Dependable Computing (PRDC)*, pp. 90–99, IEEE, 2020.

[12] N. Khoshavi, A. Roohi, C. Broyles, S. Sargolzaei, Y. Bi, and D. Z. Pan, "Shieldenn: Online accelerated framework for fault-tolerant deep neural

network architectures," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6, IEEE, 2020.

[13] E. Ozen and A. Orailoglu, "Low-cost error detection in deep neural network accelerators with linear algorithmic checksums," *Journal of Electronic Testing*, vol. 36, no. 6, pp. 703–718, 2020.

[14] F. F. dos Santos, P. F. Pimenta, C. Lunardi, L. Draghetti, L. Carro, D. Kaeli, and P. Rech, "Analyzing and increasing the reliability of convolutional neural networks on gpus," *IEEE Transactions on Reliability*, vol. 68, no. 2, pp. 663–677, 2018.

[15] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.

[16] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[17] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, pp. 1097–1105, 2012.

[18] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.

[19] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[20] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[21] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 779–788, 2016.

[22] C. J. B. Yann, Y. LeCun, and C. Cortes, ""The MNIST DATABASE of Handwritten Digits"." `http://yann.lecun.com/exdb/mnist/`. Accessed at June 2024.

[23] A. Krizhevsky, v. Nair, and G. Hinton, ""The CIFAR-10 Dataset"." `https://www.cs.toronto.edu/~kriz/cifar.html`. Accessed at June 2024.

[24] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255, Ieee, 2009.

[25] M. Menze and A. Geiger, "Object scene flow for autonomous vehicles," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3061–3070, 2015.

[26] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge," *International journal of computer vision*, vol. 88, no. 2, pp. 303–338, 2010.

[27] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized neural networks: Training neural networks with low precision weights and activations," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 6869–6898, 2017.

[28] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1," *arXiv preprint arXiv:1602.02830*, 2016.

[29] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, *et al.*, "Tensorflow: A system for large-scale machine learning," in *12th $USENIX$ symposium on operating systems design and implementation ($OSDI$ 16)*, pp. 265–283, 2016.

[30] "Keras: The python deep learning api." `https://keras.io/`. Accessed at 2024.

[31] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, pp. 8026–8037, 2019.

[32] J. Redmon, ""Darknet: Open Source Neural Networks in C"." `http://pjreddie.com/darknet/`. Accessed at 2024.

[33] ""Tiny-CNN Framework"." `https://github.com/tiny-dnn/tiny-dnn`. Accessed at 2024.

[34] G. Abich, J. Gava, R. Reis, and L. Ost, "Soft error reliability assessment of neural networks on resource-constrained iot devices," in *2020 27th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, pp. 1–4, IEEE, 2020.

[35] M. A. Talib, S. Majzoub, Q. Nasir, and D. Jamal, "A systematic literature review on hardware implementation of artificial intelligence algorithms," *The Journal of Supercomputing*, vol. 77, pp. 1897–1938, 2021.

[36] K. Guo, S. Zeng, J. Yu, Y. Wang, and H. Yang, "[dl] a survey of fpga-based neural network inference accelerators," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 12, no. 1, pp. 1–26, 2019.

[37] N. Hou, X. Yan, and F. He, "A survey on partitioning models, solution algorithms and algorithm parallelization for hardware/software co-design," *Design Automation for Embedded Systems*, vol. 23, no. 1, pp. 57–77, 2019.

[38] M. Dhouibi, A. K. Ben Salem, A. Saidi, and S. Ben Saoud, "Accelerating deep neural networks implementation: A survey," *IET Computers & Digital Techniques*, vol. 15, no. 2, pp. 79–96, 2021.

[39] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th annual international symposium on computer architecture*, pp. 1–12, 2017.

[40] D. Moolchandani, A. Kumar, and S. R. Sarangi, "Accelerating cnn inference on asics: A survey," *Journal of Systems Architecture*, vol. 113, p. 101887, 2021.

[41] J. Perez-Cerrolaza, J. Abella, L. Kosmidis, A. J. Calderon, F. Cazorla, and J. L. Flores, "Gpu devices for safety-critical systems: A survey," *ACM Computing Surveys*, vol. 55, no. 7, pp. 1–37, 2022.

[42] Y. Ibrahim, H. Wang, M. Bai, Z. Liu, J. Wang, Z. Yang, and Z. Chen, "Soft error resilience of deep residual networks for object recognition," *IEEE Access*, vol. 8, pp. 19490–19503, 2020.

[43] L. Lai, N. Suda, and V. Chandra, "Cmsis-nn: Efficient neural network kernels for arm cortex-m cpus," *arXiv preprint arXiv:1801.06601*, 2018.

[44] M. S. Mahdavinejad, M. Rezvan, M. Barekatain, P. Adibi, P. Barnaghi, and A. P. Sheth, "Machine learning for internet of things data analysis: A survey," *Digital Communications and Networks*, vol. 4, no. 3, pp. 161–175, 2018.

[45] R. Sanchez-Iborra and A. F. Skarmeta, "Tinyml-enabled frugal smart objects: Challenges and opportunities," *IEEE Circuits and Systems Magazine*, vol. 20, no. 3, pp. 4–18, 2020.

[46] R. C. Baumann, "Radiation-induced soft errors in advanced semiconductor technologies," *IEEE Transactions on Device and materials reliability*, vol. 5, no. 3, pp. 305–316, 2005.

[47] G. Chen, K. Chuah, M. Li, D. S. Chan, C. Ang, J. Zheng, Y. Jin, and D. Kwong, "Dynamic nbti of pmos transistors and its impact on device lifetime," in *2003 IEEE International Reliability Physics Symposium Proceedings, 2003. 41st Annual.*, pp. 196–202, IEEE, 2003.

[48] S. Borkar, "Designing reliable systems from unreliable components: the challenges of transistor variability and degradation," *Ieee Micro*, vol. 25, no. 6, pp. 10–16, 2005.

[49] I. Koren and C. M. Krishna, "Fault-tolerant systems," 2007.

[50] B. Johnson, "Fault-tolerant microprocessor-based systems," *IEEE Micro*, vol. 4, no. 06, pp. 6–21, 1984.

[51] A. Biswas, P. Racunas, R. Cheveresan, J. Emer, S. S. Mukherjee, and R. Rangan, "Computing architectural vulnerability factors for address-based structures," in *32nd International Symposium on Computer Architecture (ISCA'05)*, pp. 532–543, IEEE, 2005.

[52] M. Eslami, B. Ghavami, M. Raji, and A. Mahani, "A survey on fault injection methods of digital integrated circuits," *Integration*, vol. 71, pp. 154–163, 2020.

[53] A. Ruospo, L. M. Luza, A. Bosio, M. Traiola, L. Dilillo, and E. Sanchez, "Pros and cons of fault injection approaches for the reliability assessment of deep neural networks," in *2021 IEEE 22nd Latin American Test Symposium (LATS)*, pp. 1–5, IEEE, 2021.

[54] R. Leveugle, A. Calvez, P. Maistri, and P. Vanhauwaert, "Statistical fault injection: Quantified error and confidence," in *2009 Design, Automation & Test in Europe Conference & Exhibition*, pp. 502–506, IEEE, 2009.

[55] R. T. Syed, M. Ulbricht, K. Piotrowski, and M. Krstic, "A survey on fault-tolerant methodologies for deep neural networks," *Pomiary Automatyka Robotyka*, vol. 27, 2023.

[56] U. Zahid, G. Gambardella, N. J. Fraser, M. Blott, and K. Vissers, "Fat: Training neural networks for reliable inference under hardware faults," in *2020 IEEE International Test Conference (ITC)*, pp. 1–10, IEEE, 2020.

[57] K. Zhao, S. Di, S. Li, X. Liang, Y. Zhai, J. Chen, K. Ouyang, F. Cappello, and Z. Chen, "Ft-cnn: Algorithm-based fault tolerance for convolutional neural networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 7, pp. 1677–1689, 2020.

[58] F. Ponzina, M. Peon-Quiros, A. Burg, and D. Atienza, "E 2 cnns: Ensembles of convolutional neural networks to improve robustness against memory errors in edge-computing devices," *IEEE Transactions on Computers*, vol. 70, no. 8, pp. 1199–1212, 2021.

[59] Y. Li, M. Li, B. Luo, Y. Tian, and Q. Xu, "Deepdyve: Dynamic verification for deep neural networks," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pp. 101–112, 2020.

[60] K.-H. Huang and J. A. Abraham, "Algorithm-based fault tolerance for matrix operations," *IEEE transactions on computers*, vol. 100, no. 6, pp. 518–528, 1984.

[61] Parhami and Avizienis, "Detection of storage errors in mass memories using low-cost arithmetic error codes," *IEEE Transactions on Computers*, vol. 100, no. 4, pp. 302–308, 1978.

[62] L. K. Draghetti, F. F. dos Santos, L. Carro, and P. Rech, "Detecting errors in convolutional neural networks using inter frame spatio-temporal

correlation," in *2019 IEEE 25th International Symposium on On-Line Testing and Robust System Design (IOLTS)*, pp. 310–315, IEEE, 2019.

[63] H. Jiang, C. Liu, L. Liu, F. Lombardi, and J. Han, "A review, classification, and comparative evaluation of approximate arithmetic circuits," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 13, no. 4, p. 60, 2017.

[64] M. S. Ansari, H. Jiang, B. F. Cockburn, and J. Han, "Low-power approximate multipliers using encoded partial products and approximate compressors," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 8, no. 3, pp. 404–416, 2018.

[65] P. Kulkarni, P. Gupta, and M. Ercegovac, "Trading accuracy for power with an underdesigned multiplier architecture," in *IEEE International Conference on VLSI Design*, pp. 346–351, IEEE, 2011.

[66] H. R. Mahdiani, A. Ahmadi, S. M. Fakhraie, and C. Lucas, "Bio-inspired imprecise computational blocks for efficient VLSI implementation of soft-computing applications," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 57, no. 4, pp. 850–862, 2010.

[67] K. Y. Kyaw, W. L. Goh, and K. S. Yeo, "Low-power high-speed multiplier for error-tolerant application," in *Electron Devices and Solid-State Circuits (EDSSC), 2010 IEEE International Conference of*, pp. 1–4, IEEE, 2010.

[68] C.-H. Lin and C. Lin, "High accuracy approximate multiplier with error correction," in *Computer Design (ICCD), 2013 IEEE 31st International Conference on*, pp. 33–38, IEEE, 2013.

[69] A. Momeni, J. Han, P. Montuschi, and F. Lombardi, "Design and analysis of approximate compressors for multiplication," *IEEE Transactions on Computers*, vol. 64, no. 4, pp. 984–994, 2015.

[70] C. Liu, J. Han, and F. Lombardi, "A low-power, high-performance approximate multiplier with configurable partial error recovery," *Design, Automation and Test in Europe Conference and Exhibition*, pp. 1–4, 2014.

[71] H. Jiang, J. Han, F. Qiao, and F. Lombardi, "Approximate radix-8 Booth multipliers for low-power and high-performance operation," *IEEE Transactions on Computers*, vol. 65, no. 8, pp. 2638–2644, 2016.

[72] V. Mrazek, S. S. Sarwar, L. Sekanina, Z. Vasicek, and K. Roy, "Design of power-efficient approximate multipliers for approximate artificial neural networks," *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1–7, 2016.

[73] Z. Vasicek and L. Sekanina, "Evolutionary approach to approximate digital circuits design," *IEEE Transactions on Evolutionary Computation*, vol. 19, no. 3, pp. 432–444, 2015.

[74] J. N. Mitchell, "Computer multiplication and division using binary logarithms," *IRE Transactions on Electronic Computers*, no. 4, pp. 512–517, 1962.

[75] H. Saadat, H. Bokhari, and S. Parameswaran, "Minimally biased multipliers for approximate integer and floating-point multiplication," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2623–2635, 2018.

[76] W. Liu, J. Xu, D. Wang, C. Wang, P. Montuschi, and F. Lombardi, "Design and evaluation of approximate logarithmic multipliers for low power error-tolerant applications," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 65, no. 9, pp. 2856–2868, 2018.

[77] M. S. Kim, A. A. Del Barrio, R. Hermida, and N. Bagherzadeh, "Low-power implementation of Mitchell's approximate logarithmic multiplication for convolutional neural networks," *23rd Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 617–622, 2018.

[78] M. S. Kim, A. A. D. B. Garcia, L. T. Oliveira, R. Hermida, and N. Bagherzadeh, "Efficient Mitchell's approximate log multipliers for convolutional neural networks," *IEEE Transactions on Computers*, 2018.

[79] D. De Caro, N. Petra, and A. G. Strollo, "Efficient logarithmic converters for digital signal processing applications," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 58, no. 10, pp. 667–671, 2011.

[80] J. Y. L. Low and C. C. Jong, "Unified Mitchell-based approximation for efficient logarithmic conversion circuit," *IEEE Transactions on Computers*, vol. 64, no. 6, pp. 1783–1797, 2015.

[81] A. Benso and S. DiCarlo, "The art of fault injection," *Journal of Control Engineering and Applied Informatics*, vol. 13, no. 4, pp. 9–18, 2011.

[82] A. Ruospo, A. Balaara, A. Bosio, and E. Sanchez, "A pipelined multi-level fault injector for deep neural networks," in *2020 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, pp. 1–6, IEEE, 2020.

[83] A. Siddique, K. Basu, and K. A. Hoque, "Exploring fault-energy trade-offs in approximate dnn hardware accelerators," in *2021 22nd International Symposium on Quality Electronic Design (ISQED)*, pp. 343–348, IEEE, 2021.

[84] G. Li, S. K. S. Hari, M. Sullivan, T. Tsai, K. Pattabiraman, J. Emer, and S. W. Keckler, "Understanding error propagation in deep learning neural network (dnn) accelerators and applications," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–12, 2017.

[85] P. M. Basso, F. F. dos Santos, and P. Rech, "Impact of tensor cores and mixed precision on the reliability of matrix multiplication in gpus," *IEEE Transactions on Nuclear Science*, vol. 67, no. 7, pp. 1560–1565, 2020.

[86] F. Libano, P. Rech, B. Neuman, J. Leavitt, M. Wirthlin, and J. Brunhaver, "How reduced data precision and degree of parallelism impact the reliability of convolutional neural networks on fpgas," *IEEE Transactions on Nuclear Science*, vol. 68, no. 5, pp. 865–872, 2021.

[87] F. Libano, B. Wilson, M. Wirthlin, P. Rech, and J. Brunhaver, "Understanding the impact of quantization, accuracy, and radiation on the reliability of convolutional neural networks on fpgas," *IEEE Transactions on Nuclear Science*, vol. 67, no. 7, pp. 1478–1484, 2020.

[88] R. T. Syed, M. Ulbricht, K. Piotrowski, and M. Krstic, "Fault resilience analysis of quantized deep neural networks," in *2021 IEEE 32nd International Conference on Microelectronics (MIEL)*, pp. 275–279, IEEE, 2021.

[89] A. P. Arechiga and A. J. Michaels, "The effect of weight errors on neural networks," in *2018 IEEE 8th Annual Computing and Communication Workshop and Conference (CCWC)*, pp. 190–196, IEEE, 2018.

[90] I. Choi, J.-Y. Hong, J. Jeon, and J.-S. Yang, "Rq-dnn: Reliable quantization for fault-tolerant deep neural networks," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*, pp. 1–2, IEEE, 2023.

[91] L.-H. Hoang, M. A. Hanif, and M. Shafique, "Ft-clipact: Resilience analysis of deep neural networks and improving their fault tolerance using clipped activation," in *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1241–1246, IEEE, 2020.

[92] S. I. Venieris, A. Kouris, and C.-S. Bouganis, "Toolflows for mapping convolutional neural networks on fpgas: A survey and future directions," *arXiv preprint arXiv:1803.05900*, 2018.

[93] K. Abdelouahab, M. Pelcat, J. Serot, and F. Berry, "Accelerating CNN inference on FPGA: A survey," *arXiv preprint arXiv:1806.01683*, 2018.

[94] R. S. Molina, V. Gil-Costa, M. L. Crespo, and G. Ramponi, "High-level synthesis hardware design for FPGA-based accelerators: Models, methodologies, and frameworks," *IEEE Access*, vol. 10, pp. 90429–90455, 2022.

[95] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre, and K. Vissers, "Finn: A framework for fast, scalable binarized neural network inference," in *Proceedings of the 2017 ACM/SIGDA international symposium on field-programmable gate arrays*, pp. 65–74, 2017.

[96] S. I. Venieris and C.-S. Bouganis, "fpgaconvnet: Mapping regular and irregular convolutional neural networks on fpgas," *IEEE transactions on neural networks and learning systems*, vol. 30, no. 2, pp. 326–342, 2018.

[97] A. Ghaffari and Y. Savaria, "Cnn2gate: Toward designing a general framework for implementation of convolutional neural networks on fpga," *arXiv preprint arXiv:2004.04641*, 2020.

[98] P. G. Mousouliotis and L. P. Petrou, "Cnn-grinder: from algorithmic to high-level synthesis descriptions of cnns for low-end-low-cost fpga socs," *Microprocessors and Microsystems*, vol. 73, p. 102990, 2020.

[99] J. E. Stone, D. Gohara, and G. Shi, "Opencl: A parallel programming standard for heterogeneous computing systems," *Computing in science & engineering*, vol. 12, no. 3, p. 66, 2010.

[100] E. Wang, J. J. Davis, and P. Y. Cheung, "A pynq-based framework for rapid cnn prototyping," in *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pp. 223–223, IEEE, 2018.

[101] M. Bushnell and V. Agrawal, *Essentials of electronic testing for digital, memory and mixed-signal VLSI circuits*, vol. 17. Springer Science & Business Media, 2004.

[102] A. Mahmoud, N. Aggarwal, A. Nobbe, J. R. S. Vicarte, S. V. Adve, C. W. Fletcher, I. Frosio, and S. K. S. Hari, "Pytorchfi: A runtime perturbation tool for dnns," in *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, pp. 25–31, IEEE, 2020.

[103] N. Narayanan, Z. Chen, B. Fang, G. Li, K. Pattabiraman, and N. Debardeleben, "Fault injection for tensorflow applications," *IEEE Transactions on Dependable and Secure Computing*, 2022.

[104] Z. Chen, G. Li, K. Pattabiraman, and N. DeBardeleben, "Binfi: an efficient fault injector for safety-critical machine learning systems," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–23, 2019.

[105] U. K. Agarwal, A. Chan, and K. Pattabiraman, "Lltfi: Framework agnostic fault injection for machine learning applications (tools and artifact track)," in *2022 IEEE 33rd International Symposium on Software Reliability Engineering (ISSRE)*, pp. 286–296, IEEE, 2022.

[106] S. Pappalardo *et al.*, "Resilience-performance tradeoff analysis of a deep neural network accelerator," in *2023 26th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS)*, pp. 181–186, IEEE, 2023.

[107] A. Azizimazreah, Y. Gu, X. Gu, and L. Chen, "Tolerating soft errors in deep learning accelerators with reliable on-chip memory designs," in *2018 IEEE International Conference on Networking, Architecture and Storage (NAS)*, pp. 1–10, IEEE, 2018.

[108] W. Li, G. Ge, K. Guo, X. Chen, Q. Wei, Z. Gao, Y. Wang, and H. Yang, "Soft error mitigation for deep convolution neural network on fpga accelerators," in *2020 2nd IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, pp. 1–5, IEEE, 2020.

[109] M. Jasemi, S. Hessabi, and N. Bagherzadeh, "Enhancing reliability of emerging memory technology for machine learning accelerators," *IEEE Transactions on Emerging Topics in Computing*, vol. 9, no. 4, pp. 2234–2240, 2020.

[110] E. Ozen and A. Orailoglu, "Boosting bit-error resilience of dnn accelerators through median feature selection," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 11, pp. 3250–3262, 2020.

[111] B. F. Goldstein, V. C. Ferreira, S. Srinivasan, D. Das, A. S. Nery, S. Kundu, and F. M. França, "A lightweight error-resiliency mechanism for deep neural networks," in *2021 22nd International Symposium on Quality Electronic Design (ISQED)*, pp. 311–316, IEEE, 2021.

[112] S. Burel, A. Evans, and L. Anghel, "Mozart+: Masking outputs with zeros for improved architectural robustness and testing of dnn accelerators," *IEEE Transactions on Device and Materials Reliability*, vol. 22, no. 2, pp. 120–128, 2022.

[113] ""N2D2 CAD framework for DNNs"." `https://github.com/cea-list/N2D2`. Accessed at 2024.

[114] L.-H. Hoang, M. A. Hanif, and M. Shafique, "Tre-map: Towards reducing the overheads of fault-aware retraining of deep neural networks by merging fault maps," in *2021 24th Euromicro Conference on Digital System Design (DSD)*, pp. 434–441, IEEE, 2021.

[115] Y.-Y. Tsai and J.-F. Li, "Evaluating the impact of fault-tolerance capability of deep neural networks caused by faults," in *2021 IEEE 34th International System-on-Chip Conference (SOCC)*, pp. 272–277, IEEE, 2021.

[116] T.-H. Nguyen, M. Imran, J. Choi, and J.-S. Yang, "Low-cost and effective fault-tolerance enhancement techniques for emerging memories-based deep neural networks," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pp. 1075–1080, IEEE, 2021.

[117] A. Samajdar, Y. Zhu, P. Whatmough, M. Mattina, and T. Krishna, "Scale-sim: Systolic cnn accelerator simulator," *arXiv preprint arXiv:1811.02883*, 2018.

[118] Y. Zhao, K. Wang, and A. Louri, "Fsa: An efficient fault-tolerant systolic array-based dnn accelerator architecture," in *2022 IEEE 40th International Conference on Computer Design (ICCD)*, pp. 545–552, IEEE, 2022.

[119] P. Quinton, "Automatic synthesis of systolic arrays from uniform recurrent equations," *ACM SIGARCH Computer architecture news*, vol. 12, no. 3, pp. 208–214, 1984.

[120] S. Hadjis *et al.*, "Caffe con troll: Shallow ideas to speed up deep learning," in *Proceedings of the Fourth Workshop on Data analytics in the Cloud*, pp. 1–4, 2015.

[121] G. Li, S. K. S. Hari, M. Sullivan, T. Tsai, K. Pattabiraman, J. Emer, and S. W. Keckler, "Understanding error propagation in deep learning neural network (dnn) accelerators and applications," in *SC17*, 2017.

[122] S. Pappalardo *et al.*, "A fault injection framework for ai hardware accelerators," in *2023 IEEE 24th Latin American Test Symposium (LATS)*, IEEE, 2023.

[123] P. Choudhary, L. Bhargava, V. Singh, and A. K. Suhag, "Approximate computing: Evolutionary methods for functional approximation of digital circuits," *Materials Today: Proceedings*, 2022.

[124] J. D. Booth, "Algorithm-based fault tolerance at scale," 2022.

[125] A. Bosio, P. Bernardi, A. Ruospo, and E. Sanchez, "A reliability analysis of a deep neural network," in *2019 IEEE Latin American Test Symposium (LATS)*, pp. 1–6, IEEE, 2019.

[126] N. Cavagnero, F. D. Santos, M. Ciccone, G. Averta, T. Tommasi, and P. Rech, "Fault-aware design and training to enhance dnns reliability with zero-overhead," *arXiv preprint arXiv:2205.14420*, 2022.

[127] A. Ghaffari and Y. Savaria, "Cnn2gate: Toward designing a general framework for implementation of convolutional neural networks on fpga," *arXiv preprint arXiv:2004.04641*, 2020.

[128] M. Riazati, M. Daneshtalab, M. Sjödin, and B. Lisper, "Deephls: A complete toolchain for automatic synthesis of deep neural networks to fpga," in *2020 27th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, pp. 1–4, 2020.

[129] M. Riazati, M. Daneshtalab, M. Sjödin, and B. Lisper, "Autodeephls: Deep neural network high-level synthesis using fixed-point precision," in *2022 IEEE 4th International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, pp. 122–125, IEEE, 2022.

[130] S. I. Venieris and C.-S. Bouganis, "fpgaconvnet: Mapping regular and irregular convolutional neural networks on fpgas," *IEEE transactions on neural networks and learning systems*, vol. 30, no. 2, pp. 326–342, 2018.

[131] S. I. Venieris and C.-S. Bouganis, "fpgaconvnet: A framework for mapping convolutional neural networks on fpgas," in *2016 IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pp. 40–47, IEEE, 2016.

[132] K. Guo, L. Sui, J. Qiu, S. Yao, S. Han, Y. Wang, and H. Yang, "Angel-eye: A complete design flow for mapping cnn onto customized hardware," in *2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pp. 24–29, IEEE, 2016.

[133] H. Sharma, J. Park, D. Mahajan, E. Amaro, J. K. Kim, C. Shao, A. Mishra, and H. Esmaeilzadeh, "From high-level deep neural models to FPGAs," in *49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 1–12, IEEE, 2016.

[134] S. S. Sarwar, S. Venkataramani, A. Ankit, A. Raghunathan, and K. Roy, "Energy-efficient neural computing with approximate multipliers," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 14, no. 2, p. 16, 2018.

[135] S. Hong, P. Frigo, Y. Kaya, C. Giuffrida, and T. Dumitraș, "Terminal brain damage: Exposing the graceless degradation in deep neural networks under hardware fault attacks," in *28th USENIX Security Symposium (USENIX Security 19)*, pp. 497–514, 2019.

[136] S. Kundu, A. Soyyiğit, K. A. Hoque, and K. Basu, "High-level modeling of manufacturing faults in deep neural network accelerators," in *IEEE 26th International Symposium on On-Line Testing and Robust System Design (IOLTS)*, pp. 1–4, IEEE, 2020.

[137] J. J. Zhang, T. Gu, K. Basu, and S. Garg, "Analyzing and mitigating the impact of permanent faults on a systolic array based neural network accelerator," in *2018 IEEE 36th VLSI Test Symposium (VTS)*, pp. 1–6, IEEE, 2018.

[138] R. David, J. Duke, A. Jain, V. Janapa Reddi, N. Jeffries, J. Li, N. Kreeger, I. Nappier, M. Natraj, T. Wang, *et al.*, "Tensorflow lite micro: Embedded machine learning for tinyml systems," *Proceedings of Machine Learning and Systems*, vol. 3, pp. 800–811, 2021.

[139] V. Mrazek, R. Hrbacek, Z. Vasicek, and L. Sekanina, "Evoapprox8b: Library of approximate adders and multipliers for circuit design and benchmarking of approximation methods," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, pp. 258–261, March 2017.

[140] M.-C. Hsueh, T. K. Tsai, and R. K. Iyer, "Fault injection techniques and tools," *Computer*, vol. 30, no. 4, pp. 75–82, 1997.

[141] N. Khoshavi, C. Broyles, and Y. Bi, "Compression or corruption? a study on the effects of transient faults on bnn inference accelerators," in *2020 21st International Symposium on Quality Electronic Design (ISQED)*, pp. 99–104, IEEE, 2020.

[142] N. Khoshavi, C. Broyles, Y. Bi, and A. Roohi, "Fiji-fin: A fault injection framework on quantized neural network inference accelerator," in *2020 19th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pp. 1139–1144, IEEE, 2020.

[143] M. S. Ansari, V. Mrazek, B. F. Cockburn, L. Sekanina, Z. Vasicek, and J. Han, "Improving the accuracy and hardware efficiency of neural networks using approximate multipliers," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 2, pp. 317–328, 2019.

[144] B. Rokh, A. Azarpeyvand, and A. Khanteymoori, "A comprehensive survey on model quantization for deep neural networks in image classification," *ACM Transactions on Intelligent Systems and Technology*, vol. 14, no. 6, pp. 1–50, 2023.

[145] M. Nourazar, V. Rashtchi, A. Azarpeyvand, and F. Merrikh-Bayat, "Code acceleration using memristor-based approximate matrix multiplier: Application to convolutional neural networks," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 12, pp. 2684–2695, 2018.

[146] P. Nayak, D. Zhang, and S. Chai, "Bit efficient quantization for deep neural networks," in *2019 Fifth Workshop on Energy Efficient Machine Learning and Cognitive Computing-NeurIPS Edition (EMC2-NIPS)*, pp. 52–56, IEEE, 2019.

[147] M. Andjelkovic, O. Schrape, A. Breitenreiter, and M. Krstic, "Set and seu hardened clock gating cell," in *2023 38th Conference on Design of Circuits and Integrated Systems (DCIS)*, pp. 1–6, 2023.

[148] P. Rech, "Artificial neural networks for space and safety-critical applications: Reliability issues and potential solutions," *IEEE Transactions on Nuclear Science*, 2024.

[149] S. Narayanamoorthy, H. A. Moghaddam, Z. Liu, T. Park, and N. S. Kim, "Energy-efficient approximate multiplication for digital signal processing and classification applications," *IEEE transactions on very large scale integration (VLSI) systems*, vol. 23, no. 6, pp. 1180–1184, 2014.

[150] S. Hashemi, R. I. Bahar, and S. Reda, "Drum: A dynamic range unbiased multiplier for approximate applications," in *2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 418–425, IEEE, 2015.

[151] M. Taheri, H. Zandevakili, and A. Mahani, "A high-performance memristor-based smith-waterman dna sequence alignment using fpni structure," *Journal of Applied Research in Electrical Engineering*, vol. 1, no. 1, pp. 59–68, 2022.

[152] S. Vahdat, M. Kamal, A. Afzali-Kusha, and M. Pedram, "Tosam: An energy-efficient truncation-and rounding-based scalable approximate multiplier," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 5, pp. 1161–1173, 2019.

[153] L. Sayadi, S. Timarchi, and A. Sheikh-Akbari, "Two efficient approximate unsigned multipliers by developing new configuration for approximate 4: 2 compressors," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 70, no. 4, pp. 1649–1659, 2023.

[154] M. H. Haider and S.-B. Ko, "Booth encoding based energy efficient multipliers for deep learning systems," *IEEE Transactions on Circuits and Systems II: Express Briefs*, 2023.

[155] E. Farahmand, A. Mahani, B. Ghavami, M. A. Hanif, and M. Shafique, "scaletrim: Scalable truncation-based integer approximate multiplier with linearization and compensation," *arXiv preprint arXiv:2303.02495*, 2023.

[156] J. N. Mitchell, "Computer multiplication and division using binary logarithms," *IRE Transactions on Electronic Computers*, no. 4, pp. 512–517, 1962.

[157] G. Alsuhli, V. Sakellariou, H. Saleh, M. Al-Qutayri, B. Mohammad, and T. Stouraitis, *Number Systems for Deep Neural Network Architectures*. Springer, 2023.

[158] G. Li, S. K. S. Hari, M. Sullivan, T. Tsai, K. Pattabiraman, J. Emer, and S. W. Keckler, "Understanding error propagation in deep learning neural network (dnn) accelerators and applications," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–12, 2017.

[159] A. Siddique, K. Basu, and K. A. Hoque, "Exploring fault-energy trade-offs in approximate dnn hardware accelerators," in *2021 22nd International Symposium on Quality Electronic Design (ISQED)*, pp. 343–348, IEEE, 2021.

[160] G. Ammes, G. B. Manske, P. F. Butzen, A. I. Reis, and R. P. Ribas, "Atmr design by construction based on two-level als," in *2023 36th SBC/SBMicro/IEEE/ACM Symposium on Integrated Circuits and Systems Design (SBCCI)*, pp. 1–6, IEEE, 2023.

[161] H. R. Mahdiani, S. M. Fakhraie, and C. Lucas, "Relaxed fault-tolerant hardware implementation of neural networks in the presence of multiple transient errors," *IEEE transactions on neural networks and learning systems*, vol. 23, no. 8, pp. 1215–1228, 2012.

[162] T. Arifeen, A. S. Hassan, and J.-A. Lee, "Approximate triple modular redundancy: A survey," *IEEE Access*, vol. 8, pp. 139851–139867, 2020.

[163] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra, "Weight uncertainty in neural network," in *International conference on machine learning*, pp. 1613–1622, PMLR, 2015.

[164] D. Danopoulos, G. Zervakis, K. Siozios, D. Soudris, and J. Henkel, "Adapt: Fast emulation of approximate dnn accelerators in pytorch," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2022.

[165] Z. Yuan, J. Liu, J. Wu, D. Yang, Q. Wu, G. Sun, W. Liu, X. Wang, and B. Wu, "Benchmarking the reliability of post-training quantization: a particular focus on worst-case performance," *arXiv preprint arXiv:2303.13003*, 2023.

[166] R. Canal *et al.*, "Predictive reliability and fault management in exascale systems: State of the art and perspectives," *ACM Computing Surveys*, vol. 53, no. 5, pp. 1–32, 2020.

[167] R. G. Alía, A. Coronetti, K. Bilko, M. Cecchetto, G. Datzmann, S. Fiore, and S. Girard, "Heavy ion energy deposition and see intercomparison within the radnext irradiation facility network," *IEEE Transactions on Nuclear Science*, vol. 70, no. 8, pp. 1596–1605, 2023.

[168] C. Schorn, A. Guntoro, and G. Ascheid, "Accurate neuron resilience prediction for a flexible reliability management in neural network accelerators," in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 979–984, IEEE, 2018.

[169] A. Ruospo and E. Sanchez, "On the reliability assessment of artificial neural networks running on ai-oriented mpsocs," *Applied Sciences*, vol. 11, no. 14, p. 6455, 2021.

[170] S.-S. Lee and J.-S. Yang, "Value-aware parity insertion ecc for fault-tolerant deep neural network," in *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 724–729, IEEE, 2022.

[171] B. Ghavami, M. Sadati, Z. Fang, and L. Shannon, "Fitact: Error resilient deep neural networks via fine-grained post-trainable activation functions," in *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1239–1244, IEEE, 2022.

[172] M. H. Ahmadilivani, S. Mousavi, J. Raik, M. Daneshtalab, and M. Jenihhin, "Cost-effective fault tolerance for cnns using parameter vulnerability based hardening and pruning," *IOLTS'24*, In press.

[173] Z. Yan, Y. Shi, W. Liao, M. Hashimoto, X. Zhou, and C. Zhuo, "When single event upset meets deep neural networks: Observations, explorations, and remedies," in *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 163–168, IEEE, 2020.

Acknowledgements

First and foremost, I would like to express my deepest gratitude to my supervisors, **Prof. Maksim Jenihhin** and **Adjunct Prof. Masoud Daneshtalab**, for their unwavering support, guidance, and trust throughout my PhD journey. Their insightful advice, patience, and encouragement allowed me to explore my ideas freely, and their belief in my abilities helped me grow both personally and academically. Thank you for giving me the space to let my imagination flow, while always steering me in the right direction.

I owe immense thanks to my **parents**, who have been my constant pillars of strength. Their endless love, support, and encouragement have been invaluable, and without their unwavering belief in me, I would not have come this far. Your unwavering support and commitment to hard work have not only shaped my character but have also been a constant source of inspiration throughout my academic journey

A special thanks to my **colleagues**, who became more than just co-workers; they became friends for life. Your camaraderie, intellectual discussions, and unwavering support made this journey much more enjoyable and fulfilling. Whether through academic challenges or personal moments, you were always there by my side, and for that, I am truly grateful.

I would also like to extend my sincere appreciation to the **opponents** and the **thesis committee**, who dedicated their valuable time and effort to fairly and thoroughly assess my work. Your insightful feedback and constructive criticism were integral to refining and improving my thesis.

Finally, I am thankful for everyone who, directly or indirectly, contributed to my research and provided me with the motivation to complete this important milestone in my life. Your contributions are deeply appreciated.

Abstract
Methods for Reliability Assessment and Enhancement of Deep Neural Networks Hardware Accelerators

This thesis addresses the **reliability assessment and enhancement** in DNNs deployed on hardware accelerators such as FPGAs, ASICs, and GPUs, which are critical in **safety-critical applications** like autonomous driving, healthcare, and industrial automation. As DNNs become increasingly integrated into these domains, their reliability under hardware faults caused by aging, process variations, and environmental factors becomes a crucial concern.

The thesis introduces several **novel methodologies** aimed at improving the fault resilience of DNN hardware accelerators. Among the key contributions are:

- **AdAM** – A technique that dynamically adjusts computation precision to balance reliability, power efficiency, and hardware resource utilization.
- **APPRAISER** – A fault resilience analysis framework that accelerates fault injection campaigns, enabling faster and more scalable reliability assessments.
- **FORTUNE** – A hardware-agnostic fault tolerance technique that reduces memory overhead while protecting critical data bits, thus enhancing fault tolerance with minimal resource consumption.
- **SAFFIRA** – A Software Level Systolic-Array Simulator designed for assessing the reliability of DNN accelerators.
- **DeepAxe** – A framework that explores approximation and reliability trade-offs in dataflow-based DNN accelerators, offering a balance between performance and fault resilience.
- **Hybrid Analytical and Hierarchical Fault Injection-based Reliability Assessment** – A combined approach that integrates analytical models with fault injection techniques for comprehensive reliability evaluation.

The effectiveness of these methods is demonstrated through extensive **experimental validation** on various DNN architectures (e.g., LeNet, VGG, ResNet), achieving significant improvements in fault tolerance, power consumption, and hardware efficiency. These contributions offer **scalable, efficient, and low-cost solutions** for deploying reliable DNNs in **real-world safety-critical applications**, laying the groundwork for more resilient AI systems in domains like autonomous vehicles and AI-driven medical devices.

Kokkuvõte
Süvanärvivõrkude riistvara kiirendite töökindluse hindamine ja täiustamine

See doktoritöö keskendub sügavate närvivõrkude (**DNN**) **usaldusväärsuse hindamisele ja täiustamisele**, kui neid rakendatakse riistvara kiirendajatel nagu FPGA-d, ASIC-id ja GPU-d, mis on olulised **turvalisuskriitilistes rakendustes**, nagu autonoomne juhtimine, tervishoid ja tööstusautomaatika. Kuna DNN-e kasutatakse üha enam nendes valdkondades, muutub nende töökindlus riistvaravigade, nagu vananemine, protsessimuutused ja keskkonnategurid, tõttu oluliseks probleemiks.

Töös tutvustatakse mitmeid **uuenduslikke meetodeid**, mille eesmärk on parandada DNN riistvara kiirendajate rikete taluvust. Peamised panused on:

- **AdAM** – Tehnika, mis dünaamiliselt kohandab arvutustäpsust, et tasakaalustada töökindlust, energiatarbimist ja riistvararessursside kasutamist.
- **APPRAISER** – Rikete taluvuse analüüsimise raamistik, mis kiirendab rikete süstimisteste, võimaldades kiiremat ja paremini skaleeritavat usaldusväärsuse hindamist.
- **FORTUNE** – Riistvarast sõltumatu veataluvuse tehnika, mis vähendab mäluülekannet, kaitstes samal ajal kriitilisi andmebittide, suurendades sellega rikete taluvust vähese ressursikulu juures.
- **SAFFIRA** – Tarkvaratasemel süstoolne massiivide simulaator, mis on loodud DNN kiirendajate töökindluse hindamiseks.
- **DeepAxe** – Raamistik, mis uurib andmevoo DNN kiirendajate täpsuse ja töökindluse kompromisse, pakkudes tasakaalu jõudluse ja rikete taluvuse vahel.
- **Hübriidne analüütiline ja hierarhiline rikete süstimise-põhine töökindluse hindamine** – Kombineeritud lähenemisviis, mis integreerib analüütilised mudelid rikete süstimistehnikatega, et pakkuda põhjalikku töökindluse hindamist.

Nende meetodite tõhusust on tõestatud ulatuslike **eksperimentaalsete katsetega**, kasutades erinevaid DNN arhitektuure (nt LeNet, VGG, ResNet), saavutades märkimisväärseid täiustusi rikete taluvuses, energiatarbimises ja riistvara tõhususes. Need panused pakuvad **skaleeritavaid, tõhusaid ja madalate kuludega lahendusi**, mis võimaldavad DNN-e usaldusväärselt kasutada **reaalsetes turvalisuskriitilistes rakendustes**, luues tugeva aluse vastupidavamate tehisintellektisüsteemide kasutuselevõtuks, näiteks autonoomsetes sõidukites ja tehisintellekti juhitavates meditsiiniseadmetes.

Appendix 1

I

M. Taheri et al., "Exploration of Activation Fault Reliability in Quantized Systolic Array-Based DNN Accelerators," 2024 25th International Symposium on Quality Electronic Design (ISQED), San Francisco, CA, USA, 2024, pp. 1-8, doi: 10.1109/ISQED60706.2024.10528372.

2024 25th International Symposium on Quality Electronic Design (ISQED) | 979-8-3503-0927-0/24/$31.00 ©2024 IEEE | DOI: 10.1109/ISQED60706.2024.10528372

Exploration of Activation Fault Reliability in Quantized Systolic Array-Based DNN Accelerators

Mahdi Taheri[1], Natalia Cherezova[1], Mohammad Saeed Ansari[2], Maksim Jenihhin[1], Ali Mahani[3,4], Masoud Daneshtalab[1,5], Jaan Raik[1]

[1]Tallinn University of Technology, Tallinn, Estonia
[2]University of Alberta, Edmonton, Canada,
[3]Shahid Bahonar University of Kerman, Kerman, Iran
[4]York University, Toronto, Canada
[5]Mälardalen University, Västerås, Sweden
[1]mahdi.taheri@taltech.ee

***Abstract*—The stringent requirements for the Deep Neural Networks (DNNs) accelerator's reliability stand along with the need for reducing the computational burden on the hardware platforms, i.e. reducing the energy consumption and execution time as well as increasing the efficiency of DNN accelerators. Moreover, the growing demand for specialized DNN accelerators with tailored requirements, particularly for safety-critical applications, necessitates a comprehensive design space exploration to enable the development of efficient and robust accelerators that meet those requirements. Therefore, the trade-off between hardware performance, i.e. area and delay, and the reliability of the DNN accelerator implementation becomes critical and requires tools for analysis. This paper presents a comprehensive methodology for exploring and enabling a holistic assessment of the trilateral impact of quantization on model accuracy, activation fault reliability, and hardware efficiency. A fully automated framework is introduced that is capable of applying various quantization-aware techniques, fault injection, and hardware implementation, thus enabling the measurement of hardware parameters. Moreover, this paper proposes a novel lightweight protection technique integrated within the framework to ensure the dependable deployment of the final systolic-array-based FPGA implementation. The experiments on established benchmarks demonstrate the analysis flow and the profound implications of quantization on reliability, hardware performance, and network accuracy, particularly concerning the transient faults in the network's activations.**

***Index Terms*—deep neural networks, design space exploration, quantization, fault simulation, reliability assessment**

I. Introduction

In the past decades, Deep Neural Networks (DNNs) demonstrated a significant improvement in accuracy by adopting intense parameterized models [1]. As a consequence, the size of these models has drastically increased, imposing challenges in deploying them on resource-constrained platforms [2]. FPGAs are a widely used solution for flexible and efficient DNN accelerator implementations and have shown superior hardware performance in terms of latency and power [3]. In practice, deployment of an FPGA-based DNN accelerator for the safety- and mission-critical applications (e.g., autonomous driving) requires addressing the trade-off between different

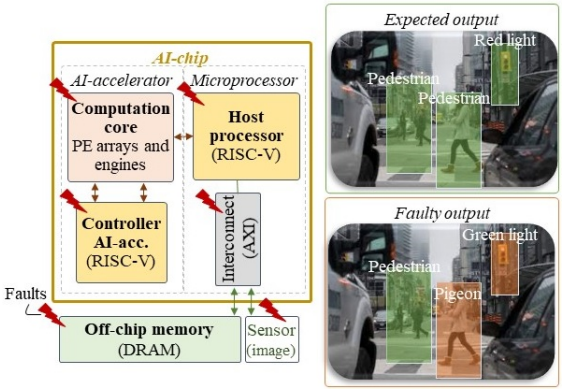


Fig. 1: Hardware-induced reliability threats in an example DNN accelerator and their possible impact on the output

design parameters of *hardware performance*, e.g., area, power, delay, and *reliability*. A compromise between conflicting requirements can be achieved by simplifying the implementation to sacrifice the precision of results but benefiting from lower resource utilization, energy consumption, and higher system efficiency. *Quantization* is one such concept that is being widely used in neural network deployments [4], [5]. Quantization is used to compress the model for storage and computation reduction. However, recent research shows that faults in memory can cause a significant drop in DNN accuracy, which raises concern about the impact of quantization on the reliability of the network [6].

The reliability of DNN accelerators expresses their ability to produce correct outputs in the presence of hardware faults originating from various phenomena, e.g., radiation-induced soft errors in memory or logic [7]. DNNs are known to be inherently fault-resilient due to the high number of learning process iterations and several parallel neurons with multiple

computation units. Nevertheless, faults may impact the output accuracy of DNNs drastically [8], and in the case of resource-constrained critical applications, the reliability of DNNs is required to be evaluated and guaranteed [9]. The complexity of such evaluation motivates an *automated toolchain* with quantization and reliability analysis to support *Design Space Exploration (DSE)* for DNN accelerators already at the early design stage, i.e. starting from a high-level description, followed by providing an FPGA prototype for the selected design.

While the protection of weights stored in ROM can be ensured through error correction codes (ECC) or similar protection techniques, the dynamic nature of activations, which are stored for a short period of time in usually unprotected memories, poses a critical concern. Thus, it is crucial to thoroughly investigate the consequences of faults in the network's activations.

This paper presents a framework containing a fully automated toolchain to perform a study on the impact of quantization on network accuracy, hardware performance, and reliability drop in the presence of activation faults (Fig. 1) in systolic-array-based FPGA accelerators. To the best of our knowledge, this is the first framework that holistically considers those parameters. A novel lightweight mitigation technique is proposed and integrated into the framework to study potential trade-offs of compensating the reliability drops. The proposed methodology enables the analysis both at the level of the network model and at the level of individual layers of the network.

This framework is empowered by techniques for quantizing the networks and restricting the activation ranges to be limited to a certain level throughout the whole network execution by applying an extra scaling function in the network inference. This framework uses the high-level description of a DNN as an input and is capable of providing a transient-fault-resilient systolic-array-based FPGA implementation of the network utilizing the design parameters selected by the DSE. The main contributions in this work are as follows:

- A methodology for holistic exploration of quantization and reliability trade-offs in systolic-array implementation that enables assessing the trilateral impact of quantization on accuracy, activation fault reliability, and hardware performance.
- A fully-automated framework that is capable of applying quantization-aware training, post-training quantization, range-restriction, fault simulation, and implementing the whole methodology down to hardware implementation to measure actual hardware parameters like area, latency, etc.
- A lightweight and effective protection technique is developed and adopted in the framework toolchain to provide the final reliable systolic-array-based FPGA implementation of the network
- Demonstration and analysis of the results on the impact of quantization on reliability, hardware performance, and accuracy of the neural networks due to the transient faults in the activations for two well-known benchmarks.

The rest of the paper is organized as follows. Related works are discussed in Section II, the methodology and framework are presented in Section III, the experimental setup and results are provided in Section IV, and finally, the work is concluded in Section V.

II. Related Works

A. DNN reliability and quantization studies

Several works examine the impact of different fault models on the basis of a number of layers in DNNs and different data types [10]. Investigation into the effects of data precision is done in [11], where authors conducted a comparison of the resilience of FP16, FP32, and FP64 in the context of Matrix Multiplication. Their findings indicated that the reduction of precision not only enhances GPU performance and efficiency but also contributes to its overall resilience.

Another study [12] involved the deployment of MNIST Convolutional Neural Networks (CNNs) on FPGAs utilizing FP32, FP16. The results of the experiment demonstrated that decreasing the data precision in CNNs can lead to a substantial enhancement in overall resilience. This improvement was attributed to the reduced memory usage. Furthermore, [13] noted that the application of binary quantization to weights in convolutional layers results in decreased vulnerability factors, although it does increase the criticality of faults. [14] showed that the impact of faults is higher in most significant bits (MSBs) and with more aggressive compression the most significant bits are more probable to be exposed to faults. The aforementioned works show that quantization from higher data representations like FP32 down to INT16 has a positive impact on the performance and overall resilience, though *on the lower quantization ranks this matter should be studied and is not always impacting positively on the resilience*

In [15], it is shown that in some cases, the impact of the faults in the weight memories of a DNN can be negligible. Even though in the above-mentioned works, *impact of faults (soft errors modeled as bit flips) in the weights of a DNN during inference is examined*, to further enhance our comprehension of the impact of quantization on the reliability of DNNs in systolic-array-based DNN accelerators, this work is enriched with an FI engine capable of injecting faults into the activations of the DNNs in the systolic architecture.

B. Fault mitigation techniques

The process of quantization and outlier regularization offers the potential to restrict the numerical range within a DNN, thereby eliminating the possibility of generating excessively large values due to faults [6], [16].

Hoang et al. analyzed how various boundary values affect the network's accuracy. They have found that the best boundary values for each layer are not necessarily the maximum values of the layers' activations [17]. Hence, they propose an interval search algorithm to find appropriate boundary values for the ReLU activation function at each layer, named FT-ClipAct. The proposed clipped activation function maps their outputs to 0 if activations exceed the boundaries. Although

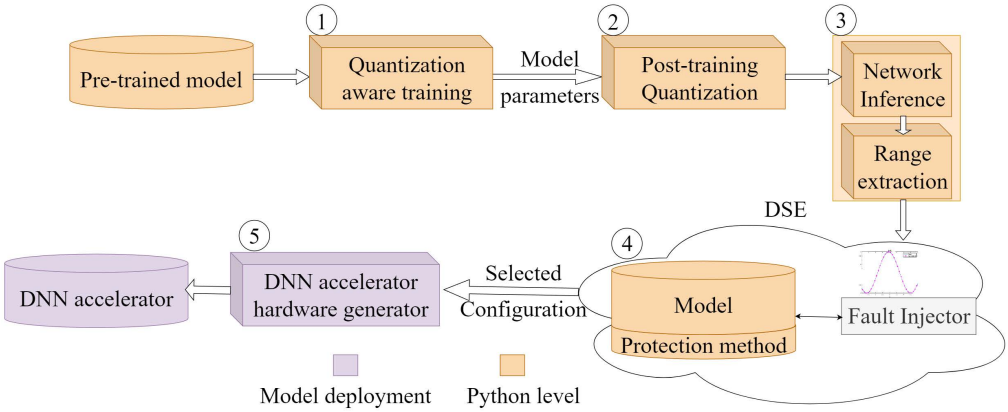


Fig. 2: Proposed methodology flow

these methods can decrease the effect of faults in DNNs, they remove a significant portion of non-zero activations by replacing them with zero, leading to an accuracy drop in high error rates. It is also noteworthy that the mentioned methods do not consider low integer quantization and are mostly working with FP32 and FP16. In this paper, we introduce a novel *lightweight range-checking* circuit that, despite the other works, can consider the maximum values of the layers' activations and replace the out-ranged values with either lower- or upper-bound to avoid fault propagation and also avoid removing a significant portion of non-zero activations by replacing them with only zero. This protection technique is employed in the DNN accelerator hardware generation step of the framework to provide the user with a prototype of the reliable accelerator.

C. DNN hardware accelerator frameworks

The advantages of implementing and deploying DNNs on FPGAs are advocated in several recent works. The existing FPGA-based toolchains to map CNNs are presented in the surveys [18]–[21]. The FINN framework [22] is released by Xilinx for the exploration of quantized CNNs' inference on FPGAs that also provide customized data-flow architectures for each network. Heterogeneous systems are another design strategy in the automated toolchains that propose hardware/software co-design [23]–[25]. In these designs, computational units, e.g., adders and multipliers, are mainly implemented on Programmable Logic (PL) that is controlled by a control unit in a CPU using a dedicated framework, e.g., OpenCL [26]. In this work, we introduce a hardware generation step as part of the framework, to explore DNN inference on an FPGA-based accelerator with a customizable systolic array. It seamlessly integrates with the PYNQ framework [27], leveraging the original PYNQ bootable image. This integration enhances versatility and compatibility, enabling users to implement their network on different FPGA devices supporting PYNQ. Furthermore, the reconfigurable systolic array implementation introduced in this step provides flexibility and scalability. Users can customize this step to meet their specific network requirements by providing trained parameters and network architectures, resulting in efficient and high-performance DNN inference.

To the best of our knowledge, none of the previous works explored the impact of using different levels of full quantization (weights, activations and biases) of a DNN in the presence of transient faults in the activations on the reliability, accuracy, and delay/resource utilization of the target DNN accelerator.

The approach proposed in this paper goes beyond the state of the art by establishing a fully automated tool for enabling efficient quantization in FPGA-based DNN accelerators aimed at safety-critical applications. The proposed framework contains a high-level simulator to study the impact of quantization on the reliability and accuracy of the network by considering the hardware architecture, with and without protection techniques, followed by an efficient and user-friendly heterogeneous FPGA implementation of the selected DNN configuration.

III. Proposed Methodology

Fig. 2 illustrates the methodology flow established in the toolchain for reliability and hardware performance analysis of quantized DNN hardware accelerators. This framework takes the DNNs' *Pre-trained model* description as the input. The design, training, and testing of the DNNs are performed in Python. *Quantization-aware training* and the *Post-training quantization*, *Range extraction* and *DSE* steps are seamlessly integrated into the same environment and are responsible for extracting the required data for the hardware generation step. This step is responsible for the hardware implementation of the selected configuration to measure actual hardware parameters like area, latency, etc.

Step 1: Quantization-aware training. For this purpose, a full quantization is implemented, targeting all activations, weights, and biases. The framework first takes the description of the network provided by the user and then uses the TFlite library for quantization-aware training. The user can replace their preferred quantization library with the toolchain for this step. The main output of this step is the quantized network's parameters (weights and biases) and network architecture.

Step 2: Post-training quantization. In the post-training quantization step, the user can define any further quantization that can be applied to the network with a negligible accuracy loss depending on the level of the quantization. This framework supports quantizing the network down to 4-bit INT. The output accuracy of the generated network is also provided at this step and is kept as a baseline for the further steps of the methodology. For this step, the following algorithm is applied to the network parameters:

The mapping equation is defined as:

$$\tilde{x} = \text{clamp}\left(\left\lfloor \frac{x}{S} \right\rfloor + Z; q_{\text{min}}, q_{\text{max}}\right)$$

$$S = \frac{x_{\text{max}} - x_{\text{min}}}{2^b - 1}$$

Where Z is the offset defined as zero-point, x_{max} and x_{min} represent the maximum and the minimum value in the vector. The quantization range $[q_{\text{min}}, q_{\text{max}}]$ is determined by the bit-width. We focus solely on uniform unsigned symmetric quantization, as it is the most commonly employed quantization setup. Hence, q_{min} is equal to 0, and q_{max} is equal to $2^b - 1$, where b denotes the bit-width, determining the number of integer grids.

Step 3: Inference and range extraction. In this step, after running the inference, the ranges of the activations are extracted for evaluation and reliability study. The ranges are extracted based on the set of validation data, and then the framework extracts the next set of ranges for each layer based on the test data and validates the extracted data correspondingly.

Step 4: Design Space Exploration.

Step 4-A: Fault simulation. Reliability analysis relies on a Fault Injection (FI) in a *systolic-array-based simulation* of the network in Python, assuming the single bit-flip faults in the activations. While the multiple-bit fault model is more accurate, it requires a prohibitively large number of fault combinations to be considered. Fortunately, it has been shown that high fault coverage obtained using the single-bit model results in a high fault coverage of multiple-bit faults [28]. Therefore, a vast majority of practical FI and test methods are based on the single-bit fault assumption. However, this framework is capable of applying multiple-bit-flips as a fault model depending on the user demand.

The reliability analysis step applies the accuracy drop comparison of the network-under-test as one of the assessment metrics. In addition, the framework assesses the reliability of the DNN by comparing the output probability vector of the golden run (i.e. the DNN that behaves as expected, without faults) and the faulty run (i.e. the DNN that includes the fault). These metrics involve the SDC (Silent Data Corruption) rate. Specifically, one of the two metrics is "absolute", and the other one is "relative". The SDC rate is defined as the proportion of faults that caused misclassification in comparison with the golden model.

- **SDC-1:** Fault caused a misclassification in the top-ranked output class.
- **SDC-5:** Fault caused the top-ranked element not to exist in the top-5 predicted output classes.
- **SDC-10%:** Fault caused a variation in the output confidence score of the top-ranked output class more than 10% compared to the golden model.

After choosing the preferred quantization in **Step 2**, the designer can go through the systolic-array-based fault injector provided for the reliability evaluation of the Quantized DNN (QDNN). The final design is fed to the next step *hardware generator* for the DNN hardware accelerator generation and hardware performance evaluation process.

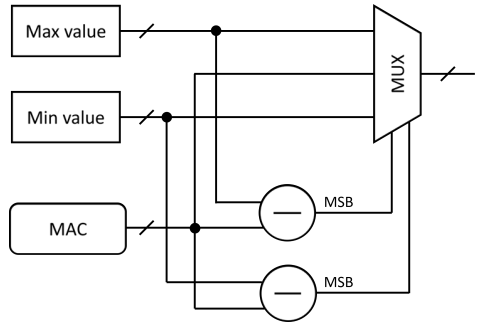


Fig. 3: Proposed lightweight mitigation technique

Step 4-B: Fault mitigation. Analyzing the output values of the network's intermediary layers post-training reveals identifiable upper and lower bounds for the neuron's output values. Leveraging this characteristic, we can ensure that any out-of-range outputs are reassigned to the respective upper or lower-bound values. This approach can be effectively implemented using specialized hardware units, as outlined below.

Out-range Error Detection: If the neuron's output value exceeds the predetermined upper or lower bound, it indicates a fault in the neuron's input values. To address this, a comparison is made between the neuron's output value and the two pre-established threshold values. For effective error detection, this paper introduces the following strategy.

For each layer, we store two values of upper bound and lower bound as the reference threshold for the out-ranged values. The output of the MAC (Multiply-Accumulate) unit is compared with the threshold values using two subtractors (negative values indicate that the output is beyond the threshold). The result of this comparison defines the final output (Fig. 3). The general overhead of this mitigation technique is two stored values for each layer, and two subtractors to compare

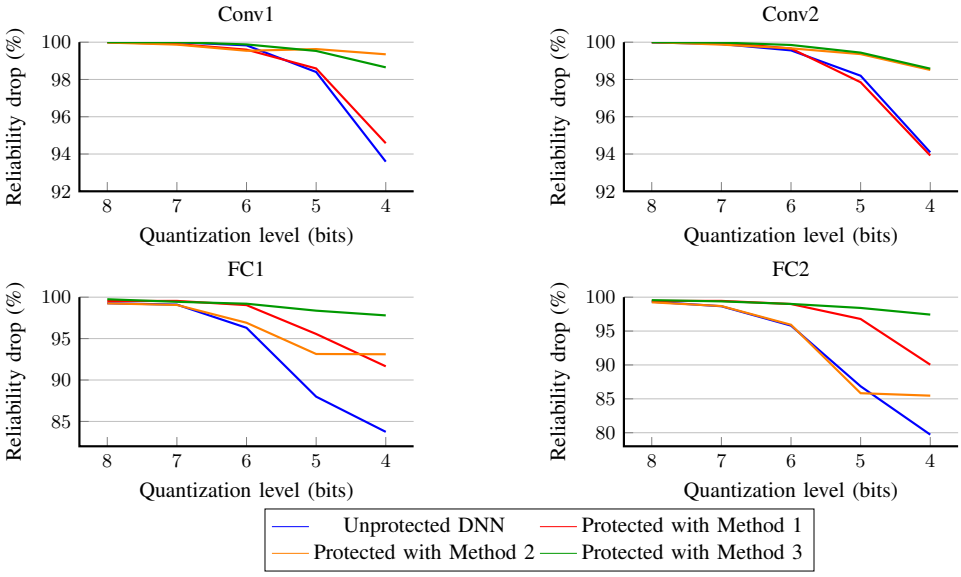


Fig. 4: Lenet-5 layer-level reports of reliability drop (based on FI for different quantized networks)

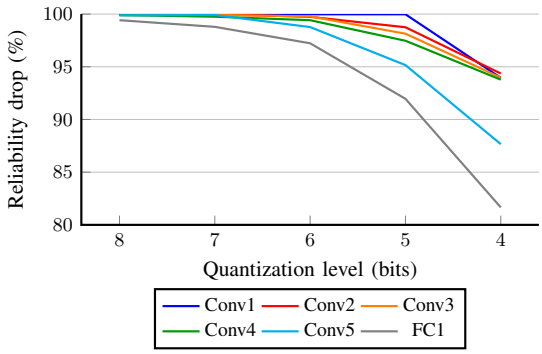


Fig. 5: AlexNet layer-level reports of reliability drop (%) based on different quantization levels (unprotected design)

the MAC output value with the range threshold values and provide the select signal for the MUX to make the decision.

Three variations of this protection technique were implemented in the software to provide users with insights into the reliability enhancements this framework offers:

1) **Method 1:** When out-of-range value is detected it is replaced by the lower bound (min value).
2) **Method 2:** When out-of-range value is detected it is replaced by the upper bound (max value).
3) **Method 3:** When out-of-range value is detected it is replaced by either lower or upper bound depending on the sign of the MAC output.

This protection technique is designed for easy replacement with any other protection methods (i.e. FT-ClipAct [17]) within this framework toolchain without compromising the overall versatility of the framework.

Step 5: Hardware generation.

At this step, a systolic-array-based QDNN accelerator for FPGA SoC is generated based on the parameters of the quantized network provided by **Step 4** to assess hardware utilization and requirements.

The following tasks are executed at this step:

1. Network parameters are analyzed to determine the size of the systolic array, bit precision, and AXI bus bandwidth for data transfer. This analysis takes into account the number of kernels and feature map sizes. The goal is to optimize hardware accelerator performance for the generated network and improve overall efficiency.

2. The board is configured with the PYNQ bootable image. PYNQ provides Python and Jupyter Notebook support to AMD-Xilinx embedded devices. Included Python APIs allow to control both processing system and programmable logic (FPGA). PYNQ setup was selected to provide the users with a familiar interactive Python environment.

3. Network weights and biases are loaded on the board as NumPy array files. The network is described using a provided Python package that interfaces with the accelerator.

4. FPGA is configured from the Jupyter Notebook with the generated accelerator. Then, inference can be run using the provided input data.

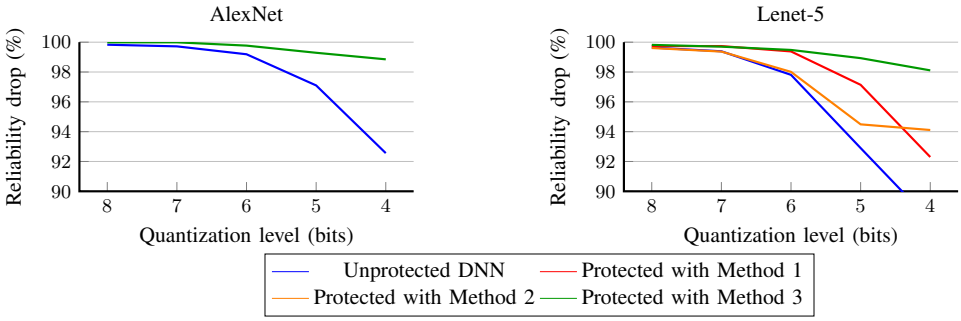


Fig. 6: Model-level reports of reliability drop (%) based on different quantization degrees for AlexNet (left) and LeNet-5 (right)

TABLE I: Lenet-5 layer-level reports of fault criticality (%) based on FI for different quantized networks

% of critical faults	Unprotected					Protected with Method 1					Protected with Method 2					Protected with Method 3				
Lenet-5	8 bit	7 bit	6 bit	5 bit	4 bit	8 bit	7 bit	6 bit	5 bit	4 bit	8 bit	7 bit	6 bit	5 bit	4 bit	8 bit	7 bit	6 bit	5 bit	4 bit
conv1	0.31	0.52	1.37	3.27	9.12	0.01	0	0.49	2.82	9.06	0.3	0.6	1.45	1.69	3.76	0	0	0.37	1.46	3.49
conv2	0.29	0.46	1.33	3.62	9.38	0.07	0.08	0.84	3.42	8.49	0.21	0.57	1.27	2.45	4.71	0.07	0.08	0.51	1.71	4.08
fc1	1.67	2.03	5.65	14.88	21.15	1.04	0.9	2.14	6.67	11.21	1.72	1.78	4.91	9.23	11.13	0.82	1.18	1.82	3.2	4.53
fc2	1.6	2.41	5.88	16.31	25.5	1.24	1.23	1.98	4.79	13.68	1.59	2.22	5.94	17.42	19.41	0.97	1.26	2.24	3.09	5.07

TABLE II: AlexNet layer-level reports of fault criticality (%) based on FI for different quantized networks

% of critical faults	Unprotected					Protected with Method 3				
AlexNet	8 bit	7 bit	6 bit	5 bit	4 bit	8 bit	7 bit	6 bit	5 bit	4 bit
conv1	0.5	0.79	1.76	4.03	8.81	0.05	0.06	0.52	1.87	3.56
conv2	0.58	1.05	1.35	1.66	4.11	0.03	0.03	1.035	1.39	3.31
conv3	1.46	1.47	5.11	11.48	23.91	0.07	0.08	1.14	1.29	4.38
conv4	0.99	1.63	2.46	7.13	14.26	0.03	0.04	1.30	4.13	5.17
conv5	0.90	2.10	3.69	7.82	14.31	0.04	0.09	1.61	3.44	5.17
fc1	3.02	4.95	8.15	16.38	31.19	0.14	0.20	1.90	5.11	8.66

TABLE III: SDC report for two unprotected Lenet-5 examples with different quantization levels

Metric (%)	16-bit	8-bit
SDC-1	3.18	5.24
SDC-5	28.04	37.26
SDC-10%	14.30	17.65

IV. Experimental Results

A. Experimental setup

Two networks are studied in this work: Lenet-5 and AlexNet. Lenet-5 is trained on the MNIST dataset, and AlexNet is trained on the CIFAR-10 dataset. Both networks are trained according to the **Step 1** methodology using quantization-aware training. Lenet-5 is trained using 16-bit INT data type, AlexNet is trained using 8-bit INT. For the study, different levels of quantization are applied in the **Step 2** using post-training quantization.

Simulations are performed on 2 $\times$ Intel Xeon Gold 6148 2.40 GHz (40 cores, 80 threads per node) with 96 GB RAM. To speed up the simulation process, the framework supports multi-thread parallelism.

To show the hardware characteristics of the output QDNN, studied networks are implemented on the Zynq UltraScale+ ZCU104 Evaluation Board (xczu7ev-ffvc1156-2-e).

B. Fault simulator

The fault simulator that is used in **Step 4** calculates the sufficient number of faults required for the reliability analysis. QDNNs generated by **Step 2** are validated by means of fault injection over the test set.

Random fault injection. According to the adopted fault model, a random single bit-flip is injected into a random activation in a random layer of the network, and the whole test set is fed to the network to obtain the accuracy of the network. This process is repeated several times to reach an acceptable confidence level, which depends on the number of neurons and data representation bit length based on [29]. This work provides an equation to reach 95% confidence level and 1% error margin. The framework adopts the formula presented in this work and provides a sufficient number of repetitions required for reliability analysis.

C. Validation results

The accuracy results for the quantized networks are reported in Table IV. Further, fault injection is applied on each network automatically as part of the defined configuration of the framework, and reliability drop and fault criticality are

TABLE IV: Model-level design space exploration results for Lenet-5 and AlexNet

Network	BP	GIOPS	Resource utilization			Accu-	Reliability	HW utilization (LUT)			Fault criticality improvement, %		
			LUT	FF	DSP	racy, %	improvement, %	M1	M2	M3	M1	M2	M3
Lenet-5	16	0.058	5298	12,892	9	95.41	—	144	144	576	—	—	—
	8	0.079	3475	7003	9	94.02	64.33	72	72	288	57.78	8.75	65.61
	7	—	—	—	—	93.93	67.95	68	68	135	71.24	14.95	67.74
	6	—	—	—	—	93.52	71.90	63	63	99	57.25	6.16	65.91
	5	—	—	—	—	92.49	81.17	68	68	81	36.30	31.34	66.86
	4	0.087	2114	3865	9	89.65	81.17	36	36	63	25.85	44.92	69.20
AlexNet	16	0.338	16,654	35,503	64	—	—	1024	1024	2048	—	—	—
	8	0.465	12,138	20,539	64	73.03	92.96	512	512	1024	—	—	94.27
	7	—	—	—	—	72.26	89.79	480	480	960	—	—	95.19
	6	—	—	—	—	72.11	73.72	448	448	704	—	—	58.59
	5	—	—	—	—	70.69	66.32	480	480	576	—	—	54.13
	4	0.562	6428	10,067	64	69.15	78.07	256	256	448	—	—	60.08

reported in Fig. 4 and Table I for the Lenet-5 and in Fig. 5 and Table II for AlexNet. *Reliability drop* is defined as the percentage of accuracy loss in the presence of the faults in the activations in a systolic-array-based simulation model of the network. *Fault criticality* is defined as percentages of the faults that show a negative impact on the network accuracy and lead to misclassification. In Fig. 4 and Table I, the results for all versions of the proposed protection technique are documented for Lenet-5. Table II, only the network protected with Method 3 is compared with the unprotected network for AlexNet, and in Fig. 5 the reports the reliability drop without the protection techniques to show the impact of faults in activations, on different quantization level and layers of an AlexNet network. primarily due to space limitations within the paper. Fig. 6 shows the reliability drop of different quantized versions of AlexNet and LeNe-5 in the presence of different protection techniques.

From the previous works [12], it is evident that the reduction in memory size and quantization can lead to enhanced resilience and mitigate the impact of weight faults due to a reduced memory footprint. However, according to the presented charts, quantization may simultaneously heighten the network's vulnerability to faults in activations and logic. This is particularly crucial in lower precision networks, where even minor bit alterations can have significant ramifications. That is why reliability studies in the DNNs should be done for each QDNN to ensure the impact of quantization on the network's reliability.

Fig. 4 shows that protection Method 3 is capable of improving the reliability of the network in the presence of a fault for more than 34.23% in the worst case for Lenet-5. These numbers are calculated based on the following equation:

$$\%\ \text{of Improvement} = \left(\frac{\text{New Value} - \text{Old Value}}{\text{Old Value}}\right) \times 100$$

The same results are reported for AlexNet in Table IV, which shows an improvement of more than 51.79% in the worst case. Improvements in fault criticality for both networks at the model level are also reported in Table IV, which demonstrates the positive impact of the protection technique on reducing the criticality of faults in both networks. These data also showcase the increasing fault criticality in different networks by increasing the level of quantization. Based on the results reported in Table IV, protection Method 3, which shows the best results for improving reliability among all of the proposed protection techniques, introduces less than 10% overhead compared to the LUTs required for the unprotected network implementation. Meanwhile, full protection of the network with TMR (Triple Module Redundancy) introduces more than 200% hardware overhead.

The fault injection procedure is performed for different quantizations and different versions of the proposed protection technique, and the accuracy drop, due to quantization and fault injection, is profiled. Further, in Table III, SDC metrics of two examples of quantized Lenet-5 are reported. It can be seen that these two networks are susceptible to injected faults. Specifically, the SDC-10% and SDC-5 are very high: on average, about 3.18% of the time the faulty inference misclassified the input in the 16-bit network and 5.24% in the 8-bit network; furthermore, in 28.04% cases for the 16-bit network and 37.26% cases for the 8-bit network, the expected class is not even in the TOP-5 predictions. In addition, it can be observed that the 16-bit quantized network shows better performance in the presence of faults compared to the 8-bit network. In general, these results show that the DNNs used in this experiment are not suitable for a safety-critical application.

Hardware resource utilization and inference latency in GIOPS (Giga Integer Operations Per Second) for different quantization levels are reported in Table IV alongside accuracy, reliability improvement due to the quantization, and hardware overhead and fault criticality improvement for fault mitigation techniques. These results of model-level design space exploration are provided for the user to understand the trade-off between reliability, accuracy, and required computational resources.

V. Conclusion

This paper presents a comprehensive methodology for exploring and enabling a holistic assessment of the trilateral impact of quantization on model accuracy, activation fault reliability, and hardware efficiency. A fully automated framework is introduced that is capable of applying various quantization techniques, fault injection, and hardware implementation, thus enabling the measurement of crucial hardware parame-

ters like area and latency. Moreover, this paper proposes a novel lightweight protection technique integrated within the framework to ensure the dependable deployment of the final systolic-array-based FPGA implementation. The experiments on established benchmarks demonstrate the analysis flow and the profound implications of quantization on reliability, hardware performance, and network accuracy, particularly concerning the transient faults in the network's activations.

VI. Acknowledgement

This work was supported in part by the Estonian Research Council grant PUT PRG1467 "CRASHLESS" and by Estonian-French PARROT project "EnTrustED".

References

[1] M. Taheri, "Dnn hardware reliability assessment and enhancement," *27th IEEE European Test Symposium (ETS).*, May 2022.

[2] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer, "A survey of quantization methods for efficient neural network inference," *arXiv preprint arXiv:2103.13630*, 2021.

[3] M. H. Ahmadilivani, M. Taheri, J. Raik, M. Daneshtalab, and M. Jenihhin, "A systematic literature review on hardware reliability assessment methods for deep neural networks," *ACM Computing Surveys*, vol. 56, no. 6, pp. 1–39, 2024.

[4] M. Taheri, M. Riazati, M. H. Ahmadilivani, M. Jenihhin, M. Daneshtalab, J. Raik, M. Sjödin, and B. Lisper, "Deepaxe: A framework for exploration of approximation and reliability trade-offs in dnn accelerators," in *2023 24th International Symposium on Quality Electronic Design (ISQED)*. IEEE, 2023, pp. 1–8.

[5] M. Taheri, M. H. Ahmadilivani, M. Jenihhin, M. Daneshtalab, and J. Raik, "Appraiser: Dnn fault resilience analysis employing approximation errors," in *2023 26th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS)*. IEEE, 2023, pp. 124–127.

[6] I. Choi, J.-Y. Hong, J. Jeon, and J.-S. Yang, "Rq-dnn: Reliable quantization for fault-tolerant deep neural networks," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2023, pp. 1–2.

[7] M. H. Ahmadilivani, M. Barbareschi, S. Barone, A. Bosio, M. Daneshtalab, S. Della Torca, G. Gavarini, M. Jenihhin, J. Raik, A. Ruospo *et al.*, "Special session: Approximation and fault resiliency of dnn accelerators," in *2023 IEEE 41st VLSI Test Symposium (VTS)*. IEEE, 2023, pp. 1–10.

[8] M. Taheri, M. Taheri, and A. Hadjahmadi, "Noise-tolerance gpu-based age estimation using resnet-50," *arXiv preprint arXiv:2305.00848*, 2023.

[9] A. Siddique, K. Basu, and K. A. Hoque, "Exploring fault-energy trade-offs in approximate dnn hardware accelerators," in *2021 22nd International Symposium on Quality Electronic Design (ISQED)*. IEEE, 2021, pp. 343–348.

[10] G. Li, S. K. S. Hari, M. Sullivan, T. Tsai, K. Pattabiraman, J. Emer, and S. W. Keckler, "Understanding error propagation in deep learning neural network (dnn) accelerators and applications," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2017, pp. 1–12.

[11] P. M. Basso, F. F. dos Santos, and P. Rech, "Impact of tensor cores and mixed precision on the reliability of matrix multiplication in gpus," *IEEE Transactions on Nuclear Science*, vol. 67, no. 7, pp. 1560–1565, 2020.

[12] F. Libano, P. Rech, B. Neuman, J. Leavitt, M. Wirthlin, and J. Brunhaver, "How reduced data precision and degree of parallelism impact the reliability of convolutional neural networks on fpgas," *IEEE Transactions on Nuclear Science*, vol. 68, no. 5, pp. 865–872, 2021.

[13] F. Libano, B. Wilson, M. Wirthlin, P. Rech, and J. Brunhaver, "Understanding the impact of quantization, accuracy, and radiation on the reliability of convolutional neural networks on fpgas," *IEEE Transactions on Nuclear Science*, vol. 67, no. 7, pp. 1478–1484, 2020.

[14] R. T. Syed, M. Ulbricht, K. Piotrowski, and M. Krstic, "Fault resilience analysis of quantized deep neural networks," in *2021 IEEE 32nd International Conference on Microelectronics (MIEL)*. IEEE, 2021, pp. 275–279.

[15] A. P. Arechiga and A. J. Michaels, "The effect of weight errors on neural networks," in *2018 IEEE 8th Annual Computing and Communication Workshop and Conference (CCWC)*. IEEE, 2018, pp. 190–196.

[16] M. Taheri, S. Sheikhpour, A. Mahani, and M. Jenihhin, "A novel fault-tolerant logic style with self-checking capability," in *2022 IEEE 28th International Symposium on On-Line Testing and Robust System Design (IOLTS)*. IEEE, 2022, pp. 1–6.

[17] L.-H. Hoang, M. A. Hanif, and M. Shafique, "Ft-clipact: Resilience analysis of deep neural networks and improving their fault tolerance using clipped activation," in *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2020, pp. 1241–1246.

[18] S. I. Venieris, A. Kouris, and C.-S. Bouganis, "Toolflows for mapping convolutional neural networks on fpgas: A survey and future directions," *arXiv preprint arXiv:1803.05900*, 2018.

[19] K. Guo, S. Zeng, J. Yu, Y. Wang, and H. Yang, "[dl] a survey of fpga-based neural network inference accelerators," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 12, no. 1, pp. 1–26, 2019.

[20] K. Abdelouahab, M. Pelcat, J. Serot, and F. Berry, "Accelerating cnn inference on fpgas: A survey," *arXiv preprint arXiv:1806.01683*, 2018.

[21] R. S. Molina, V. Gil-Costa, M. L. Crespo, and G. Ramponi, "High-level synthesis hardware design for fpga-based accelerators: Models, methodologies, and frameworks," *IEEE Access*, vol. 10, pp. 90 429–90 455, 2022.

[22] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre, and K. Vissers, "Finn: A framework for fast, scalable binarized neural network inference," in *Proceedings of the 2017 ACM/SIGDA international symposium on field-programmable gate arrays*, 2017, pp. 65–74.

[23] S. I. Venieris and C.-S. Bouganis, "fpgaconvnet: Mapping regular and irregular convolutional neural networks on fpgas," *IEEE transactions on neural networks and learning systems*, vol. 30, no. 2, pp. 326–342, 2018.

[24] A. Ghaffari and Y. Savaria, "Cnn2gate: Toward designing a general framework for implementation of convolutional neural networks on fpga," *arXiv preprint arXiv:2004.04641*, 2020.

[25] P. G. Mousouliotis and L. P. Petrou, "Cnn-grinder: from algorithmic to high-level synthesis descriptions of cnns for low-end-low-cost fpga socs," *Microprocessors and Microsystems*, vol. 73, p. 102990, 2020.

[26] J. E. Stone, D. Gohara, and G. Shi, "Opencl: A parallel programming standard for heterogeneous computing systems," *Computing in science & engineering*, vol. 12, no. 3, p. 66, 2010.

[27] E. Wang, J. J. Davis, and P. Y. Cheung, "A pynq-based framework for rapid cnn prototyping," in *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2018, pp. 223–223.

[28] M. Bushnell and V. Agrawal, *Essentials of electronic testing for digital, memory and mixed-signal VLSI circuits*. Springer Science & Business Media, 2004, vol. 17.

[29] R. Leveugle, A. Calvez, P. Maistri, and P. Vanhauwaert, "Statistical fault injection: Quantified error and confidence," in *2009 Design, Automation & Test in Europe Conference & Exhibition*. IEEE, 2009, pp. 502–506.

Appendix 2

II

M. Taheri, N. Cherezova, S. Nazari, A. Rafiq, A. Azarpeyvand, T. Ghasempouri, M. Daneshtalab, J. Raik, and M. Jenihhin, "AdAM: Adaptive fault-tolerant approximate multiplier for edge DNN accelerators," in 2024 IEEE European Test Symposium (ETS), 2024.

AdAM: Adaptive Fault-Tolerant Approximate Multiplier for Edge DNN Accelerators

Mahdi Taheri[1], Natalia Cherezova[1*], Samira Nazari[3*], Ahsan Rafiq[1], Ali Azarpeyvand[1,3], Tara Ghasempouri[1], Masoud Daneshtalab[1,2], Jaan Raik[1] and Maksim Jenihhin[1]

[1]Tallinn University of Technology, Tallinn, Estonia
[2]Mälardalen University, Västerås, Sweden
[3]University of Zanjan, Zanjan, Iran
[1]mahdi.taheri@taltech.ee

***Abstract*—Multiplication is the most resource-hungry operation in the neural network's processing elements. In this paper, we propose an architecture of a novel adaptive fault-tolerant approximate multiplier tailored for ASIC-based DNN accelerators. AdAM employs an adaptive adder relying on an unconventional use of the leading one position value of the inputs for fault detection through the optimization of unutilized adder resources. The proposed architecture uses a lightweight fault mitigation technique that sets the detected faulty bits to zero. The hardware resource utilization and the DNN accelerator's reliability metrics are used to compare the proposed solution against the triple modular redundancy (TMR) in multiplication, unprotected exact multiplication, and unprotected approximate multiplication. It is demonstrated that the proposed architecture enables a multiplication with a reliability level close to the multipliers protected by TMR utilizing 63.54% less area and having 39.06% lower power-delay product compared to the exact multiplier.**

***Index Terms*—deep neural networks, approximate computing, circuits design, reliability, resiliency assessment**

I. Introduction

The role of Deep Neural Networks (DNNs) in a wide range of safety- and mission-critical applications (e.g., autonomous driving) is expanding. Therefore, deployment of a DNN accelerator requires addressing the trade-off between different design parameters and *reliability* [1] [2]. Even though DNNs possess certain intrinsic fault-tolerant and error-resilient characteristics, it is insufficient to conclude the reliability of DNNs without considering the different characteristics of a hardware accelerator for vital applications. With the continuous scaling-down of the process, there is a discernible trend indicating that the Soft Error Rate (SER) of combinational circuits may surpass that of sequential circuits [3] [4]. Therefore, the main focus of this study is introducing a novel reliability technique to mitigate the soft errors in the combinational logic of an AI computation core.

This work presents the architecture of a novel adaptive fault-tolerant approximate multiplier (AdAM) tailored for ASIC-based DNN accelerators. Yet, the proposed multiplier can be implemented on FPGA as well. The contributions of the paper are as follows:

- The architecture of a novel adaptive fault-tolerant approximate multiplier tailored for DNN accelerators, including an adaptive adder relying on an unconventional use of the leading one position value of the inputs for fault detection through the optimization of unutilized adder resources
- Implementation and validation of the multiplier in a design synthesized for ASIC
- Reliability assessment and comparison of the proposed multiplier with exact and approximate state-of-the-art multipliers using several DNN benchmarks

The main objective of the proposed multiplier is to have the best trade-off between power-delay product (PDP) and vulnerability (accuracy drop due to the fault) which is demonstrated in the results section. Moreover, it is demonstrated that the proposed architecture enables a multiplication with a reliability level close to the multipliers protected by TMR utilizing 63.54% less area and having 39.06% lower PDP compared to the exact multiplier.

The remainder of the paper is organized as follows. Section II summarizes related works, the proposed method is presented in Section III, Section IV provides the experimental setup and discusses the results, and finally, the work is concluded in Section V.

II. Related Works

Multipliers are one of the primary arithmetic building blocks widely used in DNNs. Various approximate multipliers are proposed in the literature. ScaleTRIM is a scalable approximate unsigned LOD multiplier for DNNs that exploits curve fitting and linearization for fitting input products and a novel error compensation method using lookup tables [5]. More details about recent works on the approximation for DNNs can be found in [6].

Error introduced by approximation is deterministic and its impact can be studied on the accuracy drop of the network comprehensively. However, soft errors are unpredictable effects in contaminated and harsh environments that may lead to DNNs malfunction and accuracy drop drastically [7]. Recent research investigates the reliability of DNNs alongside approximation [8] [9] and quantization [10]. In [11], DNNs and approximated DNNs are tested in the presence of faults,

* These authors contributed equally

and the results demonstrated that approximated DNNs are more resilient under special conditions.

To increase reliability and mitigate faults, Triple Modular Redundancy (TMR) and Gate-Sizing (GS) are two well-established hardening methods widely employed to mitigate the soft error rate in combinational circuits. Despite achieving 100% fault coverage for a single fault in one module of a combinational circuit, TMR incurs a substantial near 200% area and power overhead [12]. Therefore, numerous algorithms and frameworks are developed to enhance the efficiency of applying these methods and balance their hardening effects and design costs [13].

Approximate TMR (ATMR) is a technique that replaces some modules of TMR with approximate ones while ensuring the majority voter gives the correct output [14]. However, ATMR still requires duplicating the whole combinational circuit, even at the finest level of granularity.

To tackle this issue, this work presents an adaptive reliable multiplier that provides a high level of reliability while using less area than an exact multiplier.

III. AdAM architecture

The proposed architecture is an adaptive fault-tolerant approximate multiplier tailored for DNN accelerators. This architecture includes an adaptive adder relying on an unconventional use of the leading one position value of the inputs for fault *detection* and *mitigation* through the optimization of unutilized adder resources. The proposed multiplier is an adaptation of the classical Mitchell multiplier [15]. Mitchell multiplier employs approximate logarithms of the input values. By adding these logarithms, Mitchell's algorithm estimates the product. The final result is obtained by taking the antilogarithm of this sum. Another level of approximation is introduced in the adaptive adder considering the application of this multiplier in DNNs with a proven negligible impact on the network accuracy (see Subsection III-A and Table II).

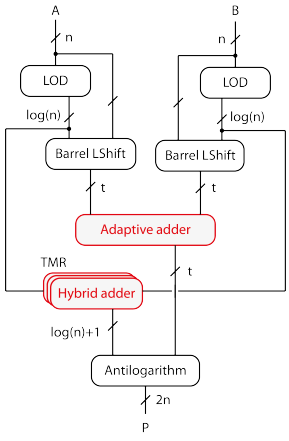


Fig. 1: AdAM architecture

The proposed architecture of the multiplier is presented in Fig. 1 (the contributions and extensions to the logarithmic Mitchell multiplier are marked with red color). Assuming each operand has $n = 8$ bits, a Leading One Detector (LOD) circuit is used to find the index of the first '1' bit in each operand. This index denoted as k, is the characteristic or integer part of the logarithm and has $log_2(n) = 3$ bits. The multiplier shifts the operands left by k bits, aligning the leading one with the Most Significant Bit (MSB). $(n–1)$ bits after the leading one represent the mantissa part denoted as m. The mantissa is truncated to $t = 5$ bits. The truncated operands are passed to the adaptive $(n–1)$-bit adder that adds mantissa together and duplicates the addition of 2 or 3 MSBs depending on the k value of the biggest operand for fault *detection* and *mitigation*. The architecture of the adaptive adder is shown in Fig. 2. The adder is based on the carry lookahead adder. Duplicated results are compared, and if there is a fault, the faulty bit is set to zero using AND gates (marked on the figure with a red rectangle). Due to the truncation of the mantissa, up to 2 Least Significant Bits (LSB) are excluded from the calculation, which affects only the bigger numbers with the k equal to 7 or 6. This introduces a small error compared to the original Mitchell algorithm that is discussed in the results section. The k values of the operands are added separately. This adder is replicated three times, as the order of the final output depends on the result of this addition. A majority voter selects the final result. Then, the antilogarithm algorithm is used to get the product of multiplication. The sum of k values determines the position of the leading one in the output product, which is followed by the sum of the mantissa parts using the appropriate shift operation.

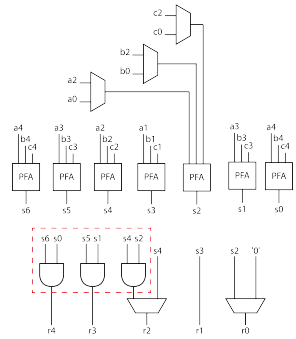


Fig. 2: Adaptive adder architecture: a and b are inputs, c is carry values and PFA stands for partial full adder

A. Adaptive adder

The adaptive adder is designed to perform fault detection and mitigation based on the LOD values of the multiplier inputs. Fig. 3 shows the scheme in which the proposed multiplier introduces fault tolerance. As shown in this figure, five cases are considered. If the maximum LOD of the inputs is 7, two LSBs are discarded, and a two-bit adder of the

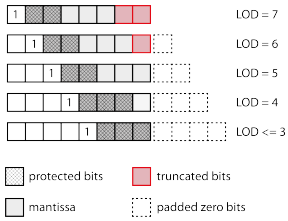


Fig. 3: Fault-tolerance and error introduced based on different LOD cases

adaptive adder is dedicated to recomputing the addition of two MSBs. These results are compared, and the mismatched bits are replaced by zeros in case of a mismatch.

For LOD = 6, only the LSB is discarded, and two higher-order bits are protected the same way as in the previous case. When LOD = 5, no bits are discarded, and two higher-order bits are protected. For LOD = 4, three higher-order bits are protected, and only LSB is not monitored. In the case of LOD $\leq$ 3, all bits are protected, enabling the proposed multiplier to provide comprehensive fault detection and mitigation for all inputs.

IV. Experimental Results

A. Experimental Setup

In this paper, the FreePDK 45 nm Nangate technology library is used in Cadence Genus 2023 to compare the hardware characteristics of the proposed methods with the state-of-the-art. The impact on the accuracy of the proposed adaptive multiplier is studied on different networks (i.e., LeNet-5, AlexNet, and VGG-16) trained on MNIST and CIFAR-10 using 8-bit INT with the help of the ADAPT framework [16]. Finally, the impact of the proposed multiplier on the reliability of DNNs is studied using AlexNet and VGG-16. To perform the reliability simulations for the case studies, a systolic-array simulator is developed and integrated into the ADAPT framework, and the impact of transient faults in the multiply-accumulate (MAC) units of the systolic array is studied in the network.

Random fault injection. Fault injection is performed, assuming the single bit-flip faults in the network's MAC operation of a systolic array for reliability assessment. Considering a prohibitively large number of fault combinations required for the multiple-bit fault model, it has been shown that high fault coverage obtained using the single-bit model results in a high fault coverage of multiple-bit faults [17]. According to the adopted single-bit fault model, a random bit-flip is injected into a random MAC unit of the systolic array core at a random execution time of the network, and the whole test set is fed to the network to obtain the accuracy of the network. This process is repeated several times to reach an acceptable confidence level, based on [18]. This work provides an equation to reach 95% confidence level and 1% error margin.

B. Hardware utilization

In this section, the adaptive multiplier is compared in terms of power and area with state-of-the-art designs.

In Table I, the accuracy, efficiency, and fault tolerance (FT) of 8-bit approximate multipliers are compared with the proposed method. Wallace, DRUM [19], TOSAM [20], and ScaleTrim [5] are used for this comparison. The accuracy is reported using Mean Absolute Relative Error (MARE). The proposed multiplier has similar hardware parameters to the state-of-the-art approximate multipliers with similar accuracy while providing reliability improvement with fault detection and mitigation capability.

TABLE I: Accuracy and efficiency of 8-bit approximate multipliers compared with the proposed method

Multiplier Architecture	Delay (ns)	Power (μW)	Area (μm^2)	MARE (%)	FT	PDP (pJ)
Exact (Wallace)	0.85	360	417	0.00	No	306
DRUM(3)	0.70	104	143	12.6	No	72.8
TOSAM(0,3)	0.68	144	198	7.7	No	97.9
DRUM(4)	1.00	172	208	6.4	No	172
TOSAM(1,5)	0.88	231	291	4.1	No	203.2
ScaleTrim(4,8)	1.8	143	216	3.3	No	257.4
AdAM	1.13	165	152	4.7	Yes	186.45

C. DNN accuracy

Table II compares the accuracy of different CNN architectures using the proposed approximate multiplier with the baseline accuracy using the exact multiplier. The evaluation shows that the accuracy of DNN with the proposed method is very close to the baseline. Hence, the proposed multiplier has a negligible effect on the accuracy of DNNs.

TABLE II: Accuracy comparison of different CNNs with an exact (baseline) and the proposed approximate multiplier

DNN	Baseline accuracy (%)	With proposed multiplier (%)
LeNet-5 (MNIST)	93.8	94.1
AlexNet (CIFAR-10)	78.0	77.7
VGG-16 (CIFAR-10)	93.4	94.0

D. Reliability analysis

To showcase the impact of the AdAM multiplier on reliability, the fault injection simulations are performed on AlexNet and VGG-16 with four different configurations. The DNN reliability is evaluated by comparing the output probability vector of the golden run (i.e. the DNN that behaves as expected, without faults) and the faulty run (i.e. the DNN that includes the fault). The SDC rate is defined as the proportion of faults that caused misclassification in comparison with the golden model. Since in DNNs, there is often not a single correct output, but a list of ranked outputs, each with a confidence score, the new criteria to determine what constitutes an SDC for a DNN application is defined in [21].

Fig. 4 demonstrates the fault tolerance comparison and reliability improvement of different networks by using the exact unprotected multipliers, using approximate unprotected multipliers (ScaleTRIM), using exact multipliers protected with TMR, and using AdAM. As illustrated, TMR has 100% of protection but it also requires about 200% of area overhead. Despite using TMR in our architecture for a small adder, we introduce very high reliability improvement without introducing hardware overhead. Since the main objective of the proposed multiplier is to have the best trade-off between PDP and vulnerability (accuracy drop due to the fault), Fig. 5 illustrates this comparison. In these charts, the closer to the origins (0,0), the higher the cost-efficiency of the fault tolerance, i.e. lower vulnerability and PDP. As shown, TMR is an inefficient solution for edge AI applications because of its high PDP, while the proposed method (AdAM) is the closest to the origin.

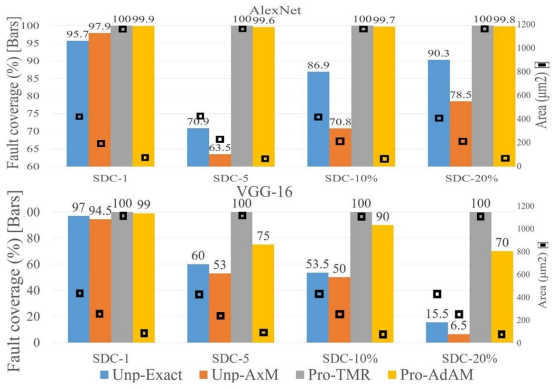


Fig. 4: Hardware efficiency (area) and fault resilience (fault coverage) trade-offs in AlexNet (up) and VGG-16 (down). Unp-exact: unprotected exact multiplier, Unp-AxM: unprotected approximate multiplier, Pro-TMR: exact multiplier protected by TMR, Pro-AdAM: proposed multiplier

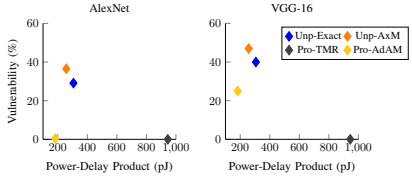


Fig. 5: PDP and vulnerability tradeoffs (considering SDC-5) in different methods

V. Conclusion

In this paper, we propose an architecture of a novel adaptive fault-tolerant approximate multiplier tailored for ASIC-based DNN accelerators. AdAM employs an adaptive adder relying on an unconventional use of the leading one position value of the inputs for fault detection through the optimization of unutilized adder resources. The proposed architecture uses a lightweight fault mitigation technique that sets the detected faulty bits to zero. It is demonstrated that the proposed multiplier provides a reliability level close to the multipliers protected by Triple Modular Redundancy (TMR) while utilizing 63.54% less area and having 39.06% lower power-delay product compared to the exact multiplier.

VI. Acknowledgement

This work was supported in part by the Estonian Research Council grant PUT PRG1467 "CRASHLESS" and by Estonian-French PARROT project "EnTrustED".

References

[1] M. H. Ahmadilivani and et al, "A systematic literature review on hardware reliability assessment methods for deep neural networks," *arXiv preprint arXiv:2305.05750*, 2023.

[2] M. Taheri, "Dnn hardware reliability assessment and enhancement," *27th IEEE European Test Symposium (ETS).*, May 2022.

[3] N. Mahatme and et al, "Comparison of combinational and sequential error rates for a deep submicron process," *TNS*, pp. 2719–2725, 2011.

[4] M. Taheri and et al, "A novel fault-tolerant logic style with self-checking capability," in *IOLTS*. IEEE, 2022, pp. 1–6.

[5] E. Farahmand and et al, "ScaleTRIM: scalable truncation-based integer approximate multiplier with linearization and compensation," *arXiv preprint arXiv:2303.02495*, 2023.

[6] G. Armeniakos and et al, "Hardware approximate techniques for deep neural network accelerators: A survey," *ACM Comput. Surv.*, vol. 55, no. 4, nov 2022.

[7] G. Li and et al, "Understanding error propagation in deep learning neural network (DNN) accelerators and applications," in *SC'17*, 2017.

[8] M. H. Ahmadilivani and et al, "Special session: Approximation and fault resiliency of DNN accelerators," in *VTS*, 2023, pp. 1–10.

[9] M. Taheri and et al, "Appraiser: Dnn fault resilience analysis employing approximation errors," in *DDECS*. IEEE, 2023, pp. 124–127.

[10] M. Taheri, N. Cherezova, and et al, "Exploration of activation fault reliability in quantized systolic array-based dnn accelerators," *arXiv preprint arXiv:2401.09509*, 2024.

[11] M. Taheri, M. H. Ahmadilivani, and et al, "Deepaxe: A framework for exploration of approximation and reliability trade-offs in DNN accelerators," in *ISQED*. IEEE, 2023, pp. 1–8.

[12] S. Mittal, "A survey on modeling and improving reliability of DNN algorithms and accelerators," *J. Syst. Archit.*, vol. 104, p. 101689, 2020.

[13] G. Ammes and et al, "ATMR design by construction based on two-level ALS," in *36th SBCCI*. IEEE, 2023, pp. 1–6.

[14] T. Arifeen and et al, "Approximate triple modular redundancy: A survey," *IEEE Access*, vol. 8, pp. 139 851–139 867, 2020.

[15] J. N. Mitchell, "Computer multiplication and division using binary logarithms," *IRE Trans. on Electronic Computers*, pp. 512–517, 1962.

[16] D. Danopoulos and et al, "Adapt: Fast emulation of approximate DNN accelerators in Pytorch," *IEEE TCAD*, 2022.

[17] M. Bushnell and V. Agrawal, *Essentials of electronic testing for digital, memory and mixed-signal VLSI circuits*. SSBM, 2004, vol. 17.

[18] R. Leveugle and et al, "Statistical fault injection: Quantified error and confidence," in *DATE*, 2009, pp. 502–506.

[19] S. Hashemi, R. I. Bahar, and S. Reda, "Drum: A dynamic range unbiased multiplier for approximate applications," in *ICCAD*, 2015, pp. 418–425.

[20] S. Vahdat and et al, "TOSAM: an energy-efficient truncation-and rounding-based scalable approximate multiplier," *TVLSI*, vol. 27, no. 5, pp. 1161–1173, 2019.

[21] G. Li, S. K. S. Hari, M. Sullivan, T. Tsai, K. Pattabiraman, J. Emer, and S. W. Keckler, "Understanding error propagation in deep learning neural network (dnn) accelerators and applications," in *SC17*, 2017.

Appendix 3

III

M. Taheri, N. Cherezova, S. Nazari, A. Azarpeyvand, T. Ghasempouri, M. Daneshtalab, J. Raik, and M. Jenihhin, "AdAM: Adaptive Approximate Multiplier for Fault Tolerance in DNN Accelerators," in IEEE Transactions on Device and Materials Reliability, doi: 10.1109/TDMR.2024.3523386.

Mahdi Taheri[1], Natalia Cherezova[1], Samira Nazari[2], Ali Azarpeyvand[1,2], Tara Ghasempouri[1], Masoud Daneshtalab[1,3], Jaan Raik[1], and Maksim Jenihhin[1]

[1]Tallinn University of Technology
[2]University of Zanjan
[3]Mälardalen University

May 06, 2024

AdAM: Adaptive Approximate Multiplier for Fault Tolerance in DNN Accelerators

Mahdi Taheri[1*], Natalia Cherezova[1*], Samira Nazari[3], Ali Azarpeyvand[1,3], Tara Ghasempouri[1], Masoud Daneshtalab[1,2], Jaan Raik[1] and Maksim Jenihhin[1]

[1]Tallinn University of Technology, Tallinn, Estonia
[2]Mälardalen University, Västerås, Sweden
[3]University of Zanjan, Zanjan, Iran
[1]mahdi.taheri@taltech.ee

***Abstract*—Deep Neural Network (DNN) hardware accelerators are essential in a spectrum of safety-critical edge-AI applications with stringent reliability, energy efficiency, and latency requirements. Multiplication is the most resource-hungry operation in the neural network's processing elements. This paper proposes a scalable adaptive fault-tolerant approximate multiplier (AdAM) tailored for ASIC-based DNN accelerators at the algorithm and circuit levels. AdAM employs an adaptive adder that relies on an unconventional use of input Leading One Detector (LOD) values for fault detection by optimizing unutilized adder resources. A gate-level optimized LOD design and a hybrid adder design are also proposed as a part of the adaptive multiplier to improve the hardware performance. The proposed architecture uses a lightweight fault mitigation technique that sets the detected faulty bits to zero. The hardware resource utilization and the DNN accelerator's reliability metrics are used to compare the proposed solution against the Triple Modular Redundancy (TMR) in multiplication, unprotected exact multiplication, and unprotected approximate multiplication. It is demonstrated that the proposed architecture enables a multiplication with a reliability level close to the multipliers protected by TMR while at the same time utilizing 2.74 $\times$ less area and with 39.06% less power-delay product compared to the exact multiplier. Moreover, it has similar area, delay, and power consumption parameters compared to the state-of-the-art approximate multipliers with similar accuracy while providing fault detection and mitigation capability.**

***Index Terms*—deep neural networks, approximate computing, circuit design, reliability, DNN accelerator**

I. Introduction

In the past decades, Deep Neural Networks (DNNs) demonstrated a significant improvement in accuracy by adopting computationally intense models [1]. Consequently, the size of these models has increased drastically, imposing challenges in their deployment on resource-constrained platforms [2].

Different DNN compression techniques such as model quantization and pruning [2] as well as approximate computing (AxC) [3] [4] enable the use of DNNs in edge devices. While these techniques decrease the accuracy of DNNs, they bring the benefits of lower resource utilization and energy consumption and higher system efficiency [5]. As an example, quantizing DNNs down to 8-bit INT gained popularity in edge AI applications, because of the minimal impact on accuracy drop and a significant reduction in memory footprint [6].

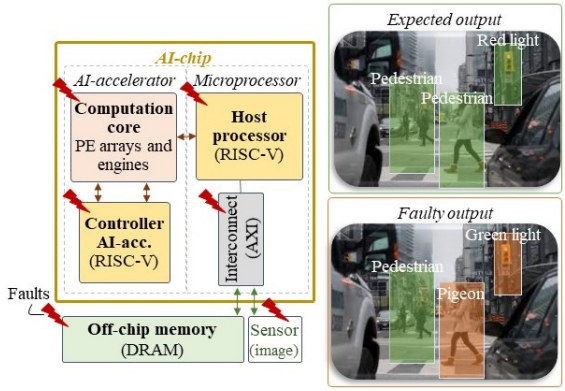


Fig. 1: Possible locations of faults in AI chip [7]

On the other hand, the role of DNNs in a wide range of safety- and mission-critical applications e.g., autonomous driving, is expanding. Therefore, deploying a DNN accelerator requires addressing the trade-off between different design parameters and *reliability* [4]. Although DNNs possess certain intrinsic fault-tolerant and error-resilient characteristics, it is insufficient to conclude the reliability of DNNs without considering the characteristics of the corresponding hardware accelerator.

Consider Fig. 1 that demonstrates different possible fault locations in an AI accelerator and their negative effect on the object detection task. In the example, a pedestrian has been identified as a bird, and a red light is misclassified as a green one leading to a potentially disastrous situation. In this work, which is an extension of the previous paper by the authors [8], the faults in the computational core of the AI chip are considered.

Moreover, with the continuous scaling-down of the process, there is a discernible trend indicating that the Soft Error Rate (SER) of combinational circuits may surpass that of sequential

* These authors contributed equally

circuits [9], [10]. Therefore, the main focus of this study is introducing a novel reliability technique to mitigate soft errors in the combinational logic of DNN accelerators while keeping resource utilization and energy consumption low.

This work presents an architecture of an adaptive fault-tolerant approximate multiplier (AdAM) tailored for ASIC-based DNN accelerators. The multiplier is based on the logarithmic Mitchell's multiplier that substitutes multiplication with the addition of approximated logarithms of the operands. The proposed multiplier protects higher-order bits of the product based on the maximum position of the leading one bit in the input strands of the multiplier. This multiplier is a negative overhead fault tolerance approximate multiplier compared to the exact multipliers. The contributions of the paper are as follows:

- The architecture of a novel adaptive fault-tolerant approximate multiplier tailored for DNN accelerators, including an adaptive adder relying on an unconventional use of the leading one position value of the inputs for fault detection through optimizing unutilized adder resources.
- DNN accelerator fault detection and protection methodology with formal definitions for scalability and reproducibility.
- Implementation and validation of the multiplier design. Reliability and hardware performance trade-off assessment and comparison of the proposed multiplier with exact and approximate state-of-the-art multipliers using several state-of-the-art DNN benchmarks.

The proposed multiplier provides a reliability level close to the multipliers protected by Triple Modular Redundancy (TMR) while utilizing 2.74 $\times$ less area and having 39% less power-delay product compared to the unprotected exact multiplier. In fact, it has similar area, delay, and power consumption parameters compared to the state-of-the-art approximate multipliers with similar accuracy while providing fault detection and mitigation capability.

The remainder of the paper is organized as follows: Section II summarizes related works. Section III presents the proposed method. Section IV presents the experimental setup and discusses the results. Finally, Section V concludes the work.

II. Related Work

Multipliers are one of the primary arithmetic building blocks widely used in DNNs. Approximate computing is a promising technique for designing digital circuits with lower area and power consumption while achieving a higher working frequency, particularly when the target application has some error resiliency, such as DNNs [3], [5]. Various approximate multipliers are proposed in the literature. These options encompass dynamic segment multiplier (DSM) structure [11], dynamic range unbiased multiplier (DRUM) structure [12], memristor-based multipliers [3], [13], simplified adder-based or truncated multipliers [14], utilization of inexact compressors [15], and approximate booth multipliers [16]. One of the other approximation techniques to further speed up the multiplication is to move to the logarithmic numbering system to compute addition instead of multiplication. The general idea is to obtain the logarithm of the inputs, calculate their sum, and convert the output value to the final result through an antilogarithm operation [17], [18]. The complexity and accuracy of this method come from the logarithm and antilogarithm steps. Truncating the input operand of multiplication and using logarithmic approximation also has a dual effect on the area, speed, and power consumption improvement. The first approximate logarithmic multiplier was proposed by Mitchell, who used binary logarithms to approximate multiplication [18]. There are several studies conducted on improving the performance and accuracy of this method by considering DNN application [19]. Tosam is a scalable approximate multiplier that reduces the number of partial products by truncating each of the input operands based on their leading one-bit position and improves delay, area, and energy consumption up to 41%, 90%, and 98%, respectively, [14]. ScaleTRIM is a scalable approximate unsigned LOD multiplier for DNNs that exploits curve fitting and linearization for fitting input products and a novel error compensation method using lookup tables [17].

The error introduced by the approximation is deterministic, and its impact can be studied comprehensively on the accuracy drop of the network. However, soft errors are unpredictable in contaminated and harsh environments that can lead to DNNs malfunction and accuracy drop drastically [20]. In contrast to the proposed approximate multipliers, AdAM considers reallocating resources saved by approximation for fault tolerance. Recent research investigates the reliability of DNNs alongside approximation [5], [4]. In [21], DNNs and approximated DNNs are tested in the presence of stuck-at faults, and the results demonstrated that approximated DNNs are more resilient under special conditions.

Triple Modular Redundancy (TMR) and Gate-Sizing (GS) are two well-established hardening methods widely employed to mitigate soft errors in combinational circuits. Despite achieving 100% fault coverage for a single fault in one module of a combinational circuit, TMR incurs a substantial near 200% area and power overhead [22]. Therefore, numerous algorithms and frameworks are developed to enhance the efficiency of applying these methods and balance their hardening effects and design costs [23].

A relaxed fault-tolerant (RFT) hardening method that exploits the inherent fault tolerance of different applications to reduce implementation costs was proposed in [24]. However, RFT lacks generality and flexibility. Approximate TMR (ATMR) is a technique that replaces some modules of TMR with approximate ones while ensuring that the majority voter gives the correct output [25]. This technique is investigated for various purposes and platforms, including some that are exclusive to FPGAs and some that work with both FPGAs and ASICs [23]. However, ATMR still requires duplicating the whole combinational circuit, even at the finest level of granularity.

To address the high overhead incurred by traditional fault tolerance methods, this work presents an adaptive approximate multiplier that provides a high level of reliability while using

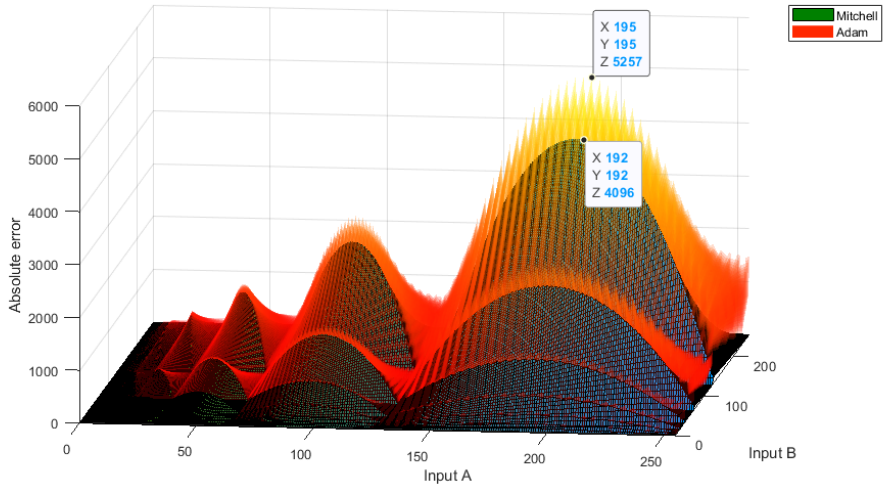


Fig. 2: Comparison of absolute error distribution between the original 8-bit Mitchell's multiplier and AdAM: top point shows the maximum absolute error for Adam, lower point shows the maximum absolute error for Mitchell's multiplier

less area and PDP than an exact multiplier.

III. AdAM architecture

We propose an architecture for adaptive fault-tolerant approximate multiplier tailored for DNN accelerators. This architecture includes an adaptive adder relying on an unconventional use of input Leading One Detector (LOD) values for fault *detection* and *mitigation* through the optimization of unutilized adder resources. A gate-level optimized LOD design and a lightweight triplicated hybrid adder design are used to enhance further the proposed architecture's reliability, resource utilization, and efficiency. The base for the proposed multiplier is the classical Mitchell multiplier [18]. However, the methodology can be applied to all logarithmic approximate multipliers. Another level of approximation is introduced in the adaptive adder (Fig. 3) considering the application of this multiplier in DNNs with a proven negligible impact on the network accuracy (see Table VI in the results section). Mitchell multiplier employs approximate logarithms of the input values. By adding these logarithms, Mitchell's algorithm estimates the product. The final result is obtained by taking the antilogarithm of this sum.

A. Definitions

Consider two n-bit positive integers A and B that can be represented as

$$A = \sum_{i=0}^{k_A} 2^i a_i, B = \sum_{i=0}^{k_B} 2^i b_i, \quad a_i = b_i = \{0,1\} \tag{1}$$

where k is the position of the leading one bit, $0 \leq k < n$. By factoring 2^k we get

$$A = 2^{k_A}\left(1 + \sum_{i=0}^{k_A-1} 2^{i-k_A} a_i\right) = 2^{k_A}(1 + X_A) \tag{2}$$

$$B = 2^{k_B}\left(1 + \sum_{i=0}^{k_B-1} 2^{i-k_B} b_i\right) = 2^{k_B}(1 + X_B) \tag{3}$$

Since $k \geq 0$, X is in the range $0 \leq X < 1$ and is the fraction term of the number. The logarithms of A and B then can be expressed as

$$log(A) = k_A + log(1 + X_A) \tag{4}$$

$$log(B) = k_B + log(1 + X_B) \tag{5}$$

Mitchell's method approximates $log(1+X)$ with the value of X. This way, logarithms are approximated as

$$log(A) \approx k_A + X_A \tag{6}$$

$$log(B) \approx k_B + X_B \tag{7}$$

The logarithm of $A * B$ can be approximated as

$$log(A * B) \approx k_A + X_A + k_B + X_B \tag{8}$$

The antilogarithm of the above expression is the final product:

$$\hat{P} = 2^{k_A + k_B}(1 + X_A + X_B) \tag{9}$$

AdAM architecture introduces a truncation parameter t that defines the level of fault tolerance, i.e. the number of pro-

tected higher-order bits of the fractional part. Considering the truncation parameter, fractional part X is defined as

$$X_A^t = \sum_{i=0}^{k_A-1} 2^{i-k_A} a_i \cdot [|i-k_A| \geq n-t] \quad (10)$$

$$X_B^t = \sum_{i=0}^{k_B-1} 2^{i-k_B} b_i \cdot [|i-k_B| \geq n-t] \quad (11)$$

where n is the number of bits, t is the truncation parameter, $[\cdot]$ are the Iverson brackets. The final product then is expressed as

$$\hat{P} = 2^{k_A+k_B}(1 + X_A^t + X_B^t) \quad (12)$$

Truncation of the fractional parts introduces additional errors to the result. The absolute error $(P - \hat{P})$ behavior of the original 8-bit Mitchell's multiplier and AdAM is compared in Fig. 2 over the entire input domain. The maximum errors introduced by each method are marked in the figure. There is no additional error when the truncated bits in both operands are '0', as the number of '1' in the truncated bits increases, additional error increases as well.

However, it has been shown that the trained synaptic weights in NN applications do not have a uniform distribution, and they are mostly centered around zero [26]. Therefore, the impact of the proposed multiplier on the accuracy of different networks is reported in the result section.

The level of fault tolerance of the proposed multiplier is defined by two parameters: the aforementioned truncation parameter t and duplication level h. The truncation parameter t defines the minimum number of protected bits, and duplication level h defines the maximum possible number of protected bits. Smaller operands have smaller mantissa values that do not require the whole adder. Therefore, more adder resources can be used for fault tolerance by duplicating the addition of more higher-order bits. Duplication level h does not affect the accuracy of the multiplier, it affects area, power, and delay. A higher h value means a higher level of fault tolerance and higher resource utilization. A smaller value means a lower level of fault tolerance and lower resource utilization. Different configurations of the proposed multiplier are denoted as AdAM(t,h).

B. Hardware implementation

The proposed architecture of the multiplier is presented in Fig. 3 (the contributions and extensions to the logarithmic Mitchell multiplier are marked with red color). First, a novel optimized *Leading One Detector (LOD)* circuit (subsection III-D) is used to find the index of the first '1' bit in each operand. This index denoted as k, is the characteristic or integer part of the logarithm and has $log_2(n)$ bits. Then, the operands are shifted left by k bits, aligning the leading one with the Most Significant Bit (MSB). $(n–1)$ bits after the leading one represent the mantissa part denoted as m. The mantissa is truncated to $(n-1-t)$ bits. The truncated operands are passed to the adaptive $(n–1)$-bit adder that adds mantissa together and duplicates the addition of several higher-order

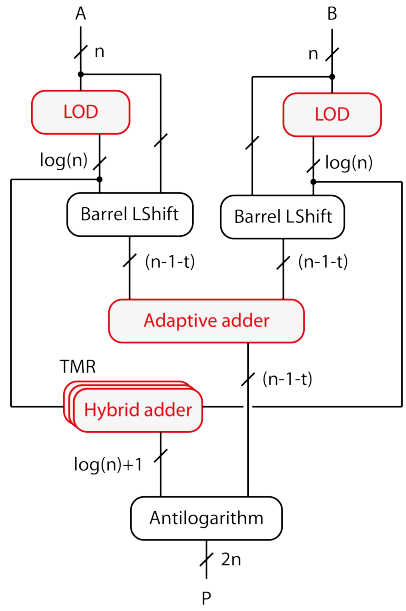


Fig. 3: AdAM architecture (the contributions and extensions to the logarithmic Mitchell multiplier are marked with red color)

bits depending on the k value of the biggest operand for fault *detection* and *mitigation*. As it was already mentioned, the number of duplicated bits depends on truncation parameter t and duplication level h.

The architecture of the *adaptive adder* is shown in Fig. 4 (subsection III-C). The adder is based on the carry-lookahead adder, and the carry generation logic is excluded from the figure to save space. Duplicated results are compared, and if there is a fault, the faulty bit is set to zero using AND gates (marked on the figure with a red rectangle). Due to the truncation of the mantissa, up to t lower-order bits are excluded from the calculation, which affects only the bigger numbers with the $k > (n-t)$. This introduces a small error compared to the original Mitchell algorithm (discussed in the following section).

The k values of the operands are added separately using a small *hybrid adder*. A hybrid between a resource-intensive but fast carry look-ahead adder and a lightweight but slow due to the accumulated delay ripple carry (CR) adder is selected for this task. This adder is replicated three times, for extra fault tolerance, as the order of the final output depends on the result of this addition. A majority voter selects the final result. Then, the antilogarithm algorithm is used to get the product of multiplication. The sum of k values determines the position of the leading one in the output product, followed by the sum of the mantissa parts using the appropriate shift operation.

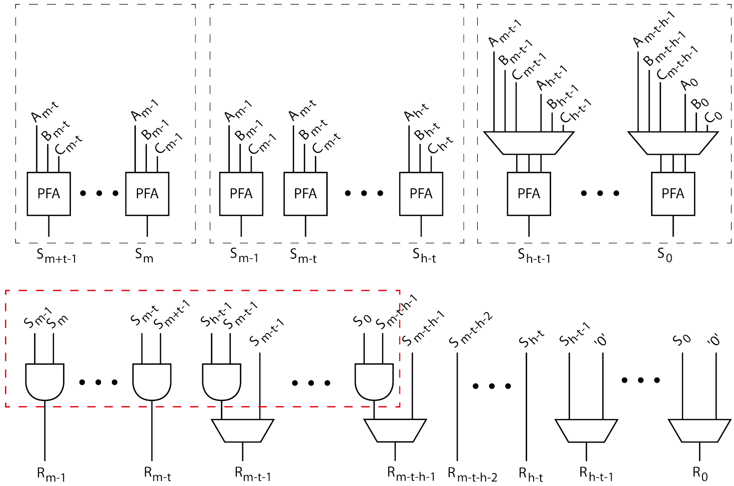


Fig. 4: Adaptive adder architecture: A and B are inputs, C is carry values and PFA stands for partial full adder, m here denotes the size of the truncated mantissa $(n-1-t)$

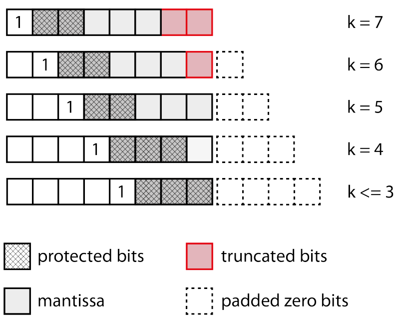


Fig. 5: Fault-tolerance and error introduced based on different input values ($n = 8, t = 2, h = 3$)

C. Adaptive adder

The adaptive adder is designed to detect and mitigate faults based on the multiplier inputs' k values. The adder is divided into three parts: t PFAs (Partial Full Adders) for duplicating the addition of t higher-order bits, $(h - t)$ PFAS with multiplexers on inputs that can duplicate the addition of up to $(h - t)$ higher-order bits or add lower-order bits, and $(n - 1 - h)$ PFAs for the addition of the mantissa. This way, for smaller operands that have smaller mantissa and do not require the whole adder, more resources can be used for fault tolerance. The size of the truncated mantissa can be expressed as $W_{X^t} = k \cdot [k < n-t] + (n-1-t) \cdot [k \geq n-t]$ using Iverson brackets notation. The available adaptive adder resources can be expressed as $W_{AR} = (n-1) - W_{X^t}$. Thus, the number of protected bits is $W_{PB} = W_{AR} \cdot [W_{AR} < h] + h \cdot [W_{AR} \geq h]$.

As an example, Fig. 5 shows the scheme in which the proposed multiplier introduces fault tolerance considering 8-bit inputs with a truncation parameter $t = 2$ and duplication level $h = 3$. As shown in this figure, five cases are considered. If the maximum LOD of the inputs is 7, two lower-order bits are discarded, and a two-bit adder of the adaptive adder is dedicated to recomputing the addition of two higher-order bits. These results are compared, and the mismatched bits are set to zero.

For $k = 6$, only the Least Significant Bit (LSB) is discarded, and two higher-order bits are protected the same way as in the previous case. When $k = 5$, no bits are discarded, and two higher-order bits are protected. For $k = 4$, three higher-order bits are protected, and only LSB is not monitored. In the case of $k \leq 3$, all bits are protected, enabling the proposed multiplier to provide comprehensive fault detection and mitigation for all inputs.

D. LOD design

The gate-level structure of the proposed LOD circuit is presented in Fig. 6. The circuit is divided into two functional parts: zero flag calculation and leading-one position detection (LOPD). LOPD consists of several stages. During the first stage, input is divided into nibbles that are processed in parallel. For each nibble, except for the first one, three signals are calculated: a, b, and c. For the first nibble, only two signals are calculated: b and c. Signal a defines whether there is a '1' bit in the nibble; signal b defines whether the output is even or odd; and signal c defines whether there is '1' in the two higher bits of the nibble. During the second stage, those signals form the final result based on the highest nibble with a '1' bit. Since the zero flag is only required at the last stage of the multiplier to determine whether the final product should be zero, it is calculated using the LOPD output values. The proposed design requires fewer logic gates (as demonstrated in the results section) than designs found in the literature (e.g.,

ScaleTRIM [17]) by reusing the output values for calculating the zero flag, knowing that it can have a longer delay.

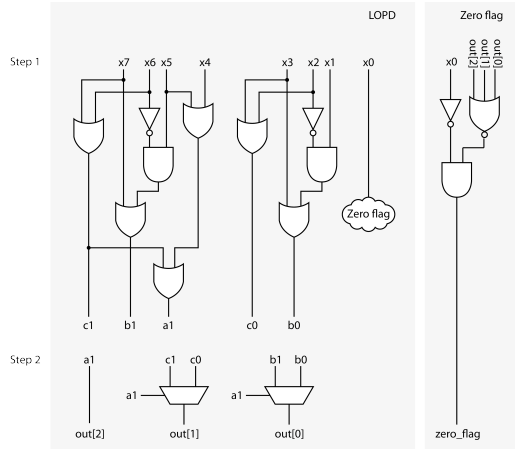


Fig. 6: Gate-level structure of the proposed LOD

IV. Experimental Results

A. Experimental Setup

In this paper, the FreePDK 45 nm Nangate technology library is used in Cadence Genus 2023 to compare the hardware characteristics (area, latency, power consumption) of the proposed methods with the state-of-the-art. The accuracy is reported using Mean Absolute Relative Error (MARE) calculated as

$$\text{MARE} = \frac{1}{N}\sum_{i=0}^{N}\left|\frac{P_i - \hat{P}_i}{P_i}\right|,$$

where N is the number of tested input combinations, P_i and $\hat{P}_i$ are the exact and approximate results.

Additionally, a design of a MAC (multiply-accumulate) unit in a systolic array is synthesized for ASIC to better illustrate the results in an AI core. The impact on the accuracy of the proposed adaptive multiplier is studied on different networks (i.e., LeNet-5, AlexNet, ResNet-18, VGG-16, DenseNet) trained on MNIST and CIFAR-10 using 8-bit INT with the help of the AdaPT framework [27]. Finally, the impact of the proposed multiplier on the reliability of DNNs is studied using the mentioned benchmarks.

Random fault injection. Fault injection is performed, assuming the single bit-flip faults in the network's MAC operation of a systolic array for reliability assessment. According to the adopted single-bit fault model, a random bit-flip is injected into a random MAC unit of the systolic array core at a random execution time of the network, and the whole test set is fed to the network to obtain the accuracy of the network. This process is repeated several times to reach an acceptable confidence level, following the approach in [28]. This reference provides an equation to reach 95% confidence level and 1% error margin.

B. Hardware utilization

In this section, the proposed LOD, and adaptive multiplier are compared in terms of power and area with state-of-the-art designs. Power–Delay Product (PDP) is used to show the efficiency of the design.

Table I shows the area, power, and delay of the proposed 8-bit LOD architecture compared to the state-of-the-art design. The proposed design has fewer gates and a smaller critical path compared to the other methods in the literature.

TABLE I: LOD hardware comparison between the proposed method and the state-of-the-art

LOD Architecture	Delay (ps)	Power (μW)	PDP (ps$\times\mu$W)	Area (μm^2)
ScaleTrim [17]	156	1.12	174.72	9.84
Proposed	136	1.09	148.24	9.57

TABLE II: Accuracy and efficiency of 8-bit approximate multipliers compared with the proposed method

Multiplier Architecture	Delay (ns)	Power (μW)	Area (μm^2)	MARE (%)	FT	PDP (pJ)
Exact (Wallace)	0.85	360	417	0.00	No	306
DRUM(3)	0.70	104	143	12.6	No	72.8
TOSAM(0,2)	0.58	120	186	10.1	No	69.6
TOSAM(0,3)	0.68	144	198	7.7	No	97.9
DRUM(4)	1.00	172	208	6.4	No	172
TOSAM(1,5)	0.88	231	291	4.1	No	203.2
AdAM(2,3)	1.13	165	152	4.7	Yes	186.4
ScaleTrim(4,8)	1.8	143	216	3.3	No	257.4

TABLE III: Accuracy and efficiency of 16-bit approximate multipliers compared with the proposed method

Multiplier Architecture	Delay (ns)	Power (mW)	Area (μm^2)	MARE (%)	FT	PDP (pJ)
Exact (Wallace)	1.22	2.08	1785	0.00	No	2537.6
DRUM(3)	0.88	0.13	257	11.9	No	114.4
TOSAM(0,2)	0.74	0.16	342	10.9	No	118.4
ScaleTrim(3,4)	1.35	0.20	281	9.23	No	279.0
TOSAM(0,3)	0.84	0.21	423	7.6	No	176.4
DRUM(4)	1.12	0.27	381	5.9	No	302.4
ScaleTrim(7,0)	2.38	0.36	492	4.06	No	871.3
TOSAM(1,5)	1.00	0.35	532	4.0	No	350.0
AdAM(6,7)	1.00	0.10	440	3.97	Yes	100.0
AdAM(4,7)	1.06	0.13	451	3.87	Yes	137.8
AdAM(4,4)	0.96	0.12	434	3.87	Yes	115.2
AdAM(2,7)	1.32	0.15	495	3.97	Yes	171.6
AdAM(2,4)	1.13	0.13	451	3.85	Yes	146.9
DRUM(5)	1.36	0.43	532	2.9	No	584.8
ScaleTrim(9,0)	2.71	0.43	541	2.2	No	1170.6
TOSAM(2,6)	1.21	0.38	564	2.1	No	459.8

Tables II, III and IV report the accuracy, efficiency, and fault tolerance (FT) of 8-bit, 16-bit, and 32-bit approximate multipliers compared with the proposed method. Wallace, DRUM [12], TOSAM [14], and ScaleTrim [17] are used for this comparison. The proposed multiplier has similar hardware parameters to the state-of-the-art approximate multipliers with

TABLE IV: Accuracy and efficiency of 32-bit approximate multipliers compared with the proposed method

Multiplier Architecture	Delay (ns)	Power (mW)	Area (μm^2)	MARE (%)	FT	PDP (pJ)
Exact (Wallace)	1.67	10.26	7618	0.00	No	17134.2
DRUM(3)	1.08	0.20	520	11.90	No	216.0
TOSAM(0,2)	0.99	0.27	844	10.90	No	267.3
TOSAM(0,3)	1.02	0.28	780	7.61	No	285.6
DRUM(4)	1.33	0.36	738	5.90	No	478.8
TOSAM(1,5)	1.18	0.40	999	3.95	No	472.0
AdAM(8,15)	1.67	0.31	1105	3.8488	Yes	517.7
AdAM(8,8)	1.30	0.29	1041	3.8488	Yes	377.0
AdAM(6,15)	1.72	0.48	1148	3.8487	Yes	825.6
AdAM(6,8)	1.46	0.33	1141	3.8487	Yes	471.9
AdAM(4,15)	2.43	0.57	1425	3.8487	Yes	1385.1
AdAM(4,8)	1.80	0.46	1403	3.8487	Yes	828.0
AdAM(2,15)	2.49	0.59	1579	3.8487	Yes	1469.1
AdAM(2,8)	2.02	0.49	1519	3.8487	Yes	989.8
DRUM(5)	1.55	0.56	944	2.89	No	868.0
TOSAM(2,6)	1.30	0.54	1146	2.06	No	702.0
DRUM(6)	1.69	0.75	1059	1.47	No	1267.5
TOSAM(3,7)	1.44	0.69	1294	1.05	No	993.6
DRUM(7)	1.85	0.96	1235	0.73	No	1776.0
TOSAM(4,8)	1.57	0.83	1411	0.53	No	1303.1
DRUM(8)	1.93	1.17	1402	0.37	No	3545.2
TOSAM(5,9)	1.60	1.08	1625	0.27	No	1728.0

similar accuracy while providing reliability improvement with fault detection and mitigation capability.

TABLE V: Efficiency of 8-bit MAC unit of a systolic array with different multipliers

MAC Architecture	Delay (ns)	Power (mW)	PDP (ns×mW)	Area (μm^2)
Exact (Wallace)	1151	2.10	2417.10	975.95
ScaleTrim(4,8)	1106	2.00	2212.00	949.35
AdAM(2,3)	1098	1.47	1614.06	794.80

TABLE VI: Accuracy comparison of different CNNs with an exact (baseline) and the proposed approximate multiplier

DNN	Accuracy with Wallace (%)	Accuracy with AdAM(2,3) (%)
LeNet-5 (MNIST)	93.8	94.1
AlexNet (MNIST)	78.0	77.7
VGG-11 (CIFAR-10)	93.4	94.0
VGG-13 (CIFAR-10)	92.0	92.0
VGG-19 (CIFAR-10)	92.0	93.0
ResNet-18 (CIFAR-10)	93.8	93.2
ResNet-34 (CIFAR-10)	93.0	94.0
ResNet-50 (CIFAR-10)	95.0	94.0
DenseNet (CIFAR-10)	92.6	95.0
Inception (CIFAR-10)	92.6	95.0

Table V shows the results of MAC implementation with different multipliers. The MAC unit with the proposed multiplier takes less area and has a lower PDP value.

C. DNN accuracy

Table VI compares the accuracy of different CNN architectures using the proposed approximate multiplier with the baseline accuracy using the exact multiplier. The evaluation shows that the accuracy of DNN with the proposed method is very close to the baseline. Hence, the proposed multiplier has a negligible effect on the accuracy of DNNs.

D. Reliability analysis

To showcase the impact of the AdAM multiplier on reliability and performance trade-offs, the fault injection simulations are performed on a variety of 8-bit convolutional neural network architectures including AlexNet, DenseNet, Inception, VGG-11, VGG-13, VGG-19, ResNet-18, ResNet-34, and ResNet-50 with four different configurations: unprotected exact multipliers (Unp-Exact), unprotected approximate multipliers (Unp-AxM), protected exact multipliers with TMR (Pro-TMR), and protected approximate multiplier with AdAM (Pro-AdAM). The DNN reliability is evaluated by comparing the output probability vector of the golden run (i.e. the DNN that behaves as expected, without faults) and the faulty run (i.e. the DNN that includes the fault). The SDC rate is defined as the proportion of faults that caused misclassification in comparison with the golden model. Since in DNNs, there is often not a single correct output, but a list of ranked outputs, each with a confidence score [29], we need to use new criteria to determine what constitutes an SDC for a DNN application. Therefore, we consider four types of SDCs as follows:

- **SDC-1:** Misclassification in the top-ranked output class.
- **SDC-5%:** More than 5% of variation in the top-ranked output confidence score compared to the golden model.
- **SDC-10%:** More than 10% of variation in the top-ranked output confidence score compared to the golden model.
- **SDC-20%:** More than 20% of variation in the top-ranked output confidence score compared to the golden model.

Fig. 7 demonstrates the fault tolerance comparison and reliability improvement (for SDC-1 and SDC-10% as two examples) of different networks by using the protected approximate multiplier proposed in this work compared to the unprotected exact and approximated networks, and protected networks using TMR. As illustrated, TMR has 100% of protection, but it also requires about 200% of area overhead. Different from TMR, in our technique we introduce a high-reliability improvement without introducing hardware overhead. The results for reliability improvement considering SDC-5% and SDC-20% are reported in Tables VII and VIII. Since the main objective of the proposed multiplier is to have the best trade-off in PDP and vulnerability, Fig. 8 illustrate these comparisons based on different vulnerability metrics (SDC-10, SDC-5%, SDC-10% and SDC-20%). In these charts, the closer to the origins (0,0), the higher the cost-efficiency of the fault tolerance, i.e. less vulnerability and less PDP. As shown, TMR is a less efficient solution for edge AI applications because of its high PDP, while the proposed method (AdAM) is the closest to the origin.

V. Conclusion

In this paper, we propose an architecture of a novel adaptive fault-tolerant approximate multiplier tailored for ASIC-based DNN accelerators. AdAM employs an adaptive adder that

TABLE VII: Fault coverage (SDC-5%) in different benchmarks

SDC-5%	AlexNet	DenseNet	Inception	VGG-11	VGG-13	VGG-19	ResNet-18	ResNet-34	ResNet-50
Unp-Exact	70.9	17.2	21.8	9.2	10.8	10.8	10.8	17.2	16.2
Unp-AxM	63.5	15.4	20.6	6	7.2	17.8	9.4	13.6	12.2
Pro-TMR	100	100	100	100	100	100	100	100	100
Pro-AdAM	99.6	42.2	41.8	36.6	32.6	57.2	37.6	44.8	38.2

TABLE VIII: Fault coverage (SDC-20%) in different benchmarks

SDC-20%	AlexNet	DenseNet	Inception	VGG-11	VGG-13	VGG-19	ResNet-18	ResNet-34	ResNet-50
Unp-Exact	90.3	91.6	76.6	77.4	81	81	84	86.8	86
Unp-AxM	78.5	92.2	75.8	76	78.8	75	80	84.8	82.4
Pro-TMR	100	100	100	100	100	100	100	100	100
Pro-AdAM	99.8	96.6	89.2	94.8	88.8	86.4	90.6	91	92.6

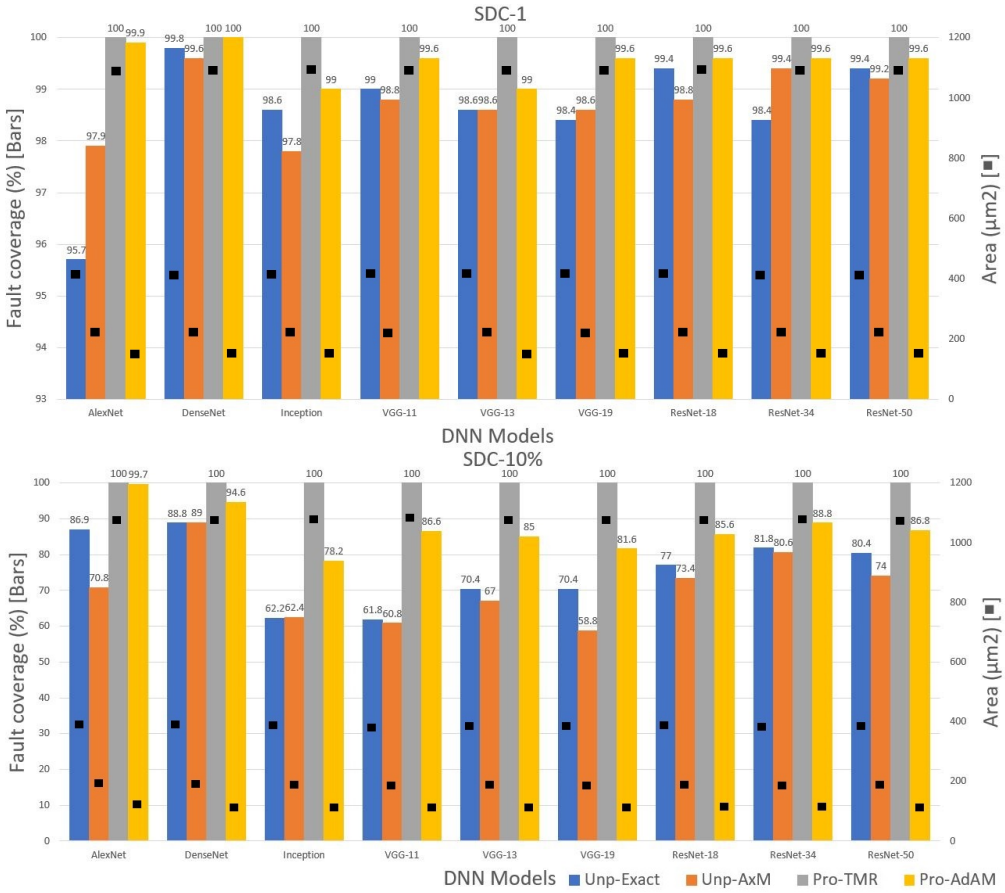


Fig. 7: Hardware efficiency (area) and fault resilience (fault coverage considering SDC-1 and SDC-10%) trade-offs in different benchmarks: AlexNet (MNIST), DenseNet(CIFAR), Inception(CIFAR), ResNet-18 (CIFAR), ResNet-34 (CIFAR), ResNet-50 (CIFAR), VGG-11 (CIFAR), VGG-13 (CIFAR), VGG-16 (CIFAR). Unp-Exact: unprotected exact multiplier, Unp-AxM: unprotected approximate multiplier, Pro-TMR: exact multiplier protected with TMR, Pro-AdAM: proposed multiplier

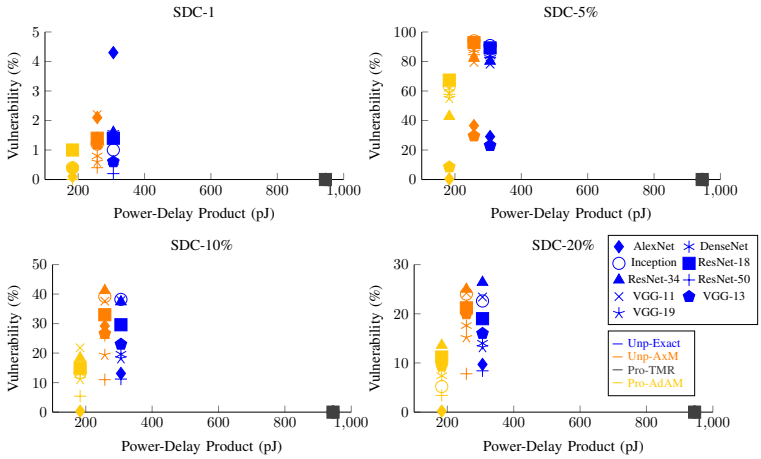


Fig. 8: PDP and vulnerability tradeoffs (considering different SDCs) in different benchmarks: AlexNet, DenseNet, Inception, ResNet-18, ResNet-34, ResNet-50, VGG-11, VGG-13, VGG-16. Unp-exact: unprotected exact multiplier, Unp-AxM: unprotected approximated multiplier, Pro-TMR: exact multiplier protected with TMR, Pro-AdAM: proposed multiplier

relies on an unconventional use of input Leading One Detector (LOD) values for fault detection by optimizing unutilized adder resources. A gate-level optimized LOD design is also proposed to improve the hardware performance as part of the adaptive multiplier. The proposed architecture uses a lightweight fault mitigation technique that sets the detected faulty bits to zero.

It is demonstrated that the proposed architecture enables a multiplication with a reliability level close to the multipliers protected by TMR while at the same time utilizing 2.74 $\times$ less area and with 39.06% less power-delay product compared to the exact multiplier.

VI. Acknowledgement

This work was supported in part by the Estonian Research Council grant PUT PRG1467 "CRASHLESS" and by Estonian-French PARROT project "EnTrustED".

References

[1] M. H. Ahmadilivani, M. Taheri, J. Raik, M. Daneshtalab, and M. Jenihhin, "A systematic literature review on hardware reliability assessment methods for deep neural networks," *ACM Computing Surveys*, vol. 56, no. 6, pp. 1–39, 2024.

[2] B. Rokh, A. Azarpeyvand, and A. Khanteymoori, "A comprehensive survey on model quantization for deep neural networks in image classification," *ACM Transactions on Intelligent Systems and Technology*, vol. 14, no. 6, pp. 1–50, 2023.

[3] M. Nourazar, V. Rashtchi, A. Azarpeyvand, and F. Merrikh-Bayat, "Code acceleration using memristor-based approximate matrix multiplier: Application to convolutional neural networks," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 12, pp. 2684–2695, 2018.

[4] M. Taheri, M. Riazati, M. H. Ahmadilivani, M. Jenihhin, M. Daneshtalab, J. Raik, M. Sjödin, and B. Lisper, "Deepaxe: A framework for exploration of approximation and reliability trade-offs in dnn accelerators," in *2023 24th International Symposium on Quality Electronic Design (ISQED)*. IEEE, 2023, pp. 1–8.

[5] M. H. Ahmadilivani, M. Barbareschi, S. Barone, A. Bosio, M. Daneshtalab, S. Della Torca, G. Gavarini, M. Jenihhin, J. Raik, A. Ruospo *et al.*, "Special session: Approximation and fault resiliency of dnn accelerators," in *2023 IEEE 41st VLSI Test Symposium (VTS)*. IEEE, 2023, pp. 1–10.

[6] P. Nayak, D. Zhang, and S. Chai, "Bit efficient quantization for deep neural networks," in *2019 Fifth Workshop on Energy Efficient Machine Learning and Cognitive Computing-NeurIPS Edition (EMC2-NIPS)*. IEEE, 2019, pp. 52–56.

[7] M. Taheri, N. Cherezova, M. S. Ansari, M. Jenihhin, A. Mahani, M. Daneshtalab, and J. Raik, "Exploration of activation fault reliability in quantized systolic array-based dnn accelerators," *arXiv preprint arXiv:2401.09509*, 2024.

[8] M. Taheri, N. Cherezova, S. Nazari, A. Rafiq, A. Azarpeyvand, T. Ghasempouri, M. Daneshtalab, J. Raik, and M. Jenihhin, "Adam: Adaptive fault-tolerant approximate multiplier for edge dnn accelerators," *In press, ETS 2024, arXiv:2403.02936*, 2024.

[9] M. Andjelkovic, O. Schrape, A. Breitenreiter, and M. Krstic, "Set and seu hardened clock gating cell," in *2023 38th Conference on Design of Circuits and Integrated Systems (DCIS)*, 2023, pp. 1–6.

[10] P. Rech, "Artificial neural networks for space and safety-critical applications: Reliability issues and potential solutions," *IEEE Transactions on Nuclear Science*, 2024.

[11] S. Narayanamoorthy, H. A. Moghaddam, Z. Liu, T. Park, and N. S. Kim, "Energy-efficient approximate multiplication for digital signal processing and classification applications," *IEEE transactions on very large scale integration (VLSI) systems*, vol. 23, no. 6, pp. 1180–1184, 2014.

[12] S. Hashemi, R. I. Bahar, and S. Reda, "Drum: A dynamic range unbiased multiplier for approximate applications," in *2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2015, pp. 418–425.

[13] M. Taheri, H. Zandevakili, and A. Mahani, "A high-performance memristor-based smith-waterman dna sequence alignment using fpni structure," *Journal of Applied Research in Electrical Engineering*, vol. 1, no. 1, pp. 59–68, 2022.

[14] S. Vahdat, M. Kamal, A. Afzali-Kusha, and M. Pedram, "Tosam: An energy-efficient truncation-and rounding-based scalable approximate multiplier," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 5, pp. 1161–1173, 2019.

[15] L. Sayadi, S. Timarchi, and A. Sheikh-Akbari, "Two efficient approximate unsigned multipliers by developing new configuration for

approximate 4: 2 compressors," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 70, no. 4, pp. 1649–1659, 2023.

[16] M. H. Haider and S.-B. Ko, "Booth encoding based energy efficient multipliers for deep learning systems," *IEEE Transactions on Circuits and Systems II: Express Briefs*, 2023.

[17] E. Farahmand, A. Mahani, B. Ghavami, M. A. Hanif, and M. Shafique, "scaletrim: Scalable truncation-based integer approximate multiplier with linearization and compensation," *arXiv preprint arXiv:2303.02495*, 2023.

[18] J. N. Mitchell, "Computer multiplication and division using binary logarithms," *IRE Transactions on Electronic Computers*, no. 4, pp. 512–517, 1962.

[19] G. Alsuhli, V. Sakellariou, H. Saleh, M. Al-Qutayri, B. Mohammad, and T. Stouraitis, *Number Systems for Deep Neural Network Architectures*. Springer, 2023.

[20] G. Li, S. K. S. Hari, M. Sullivan, T. Tsai, K. Pattabiraman, J. Emer, and S. W. Keckler, "Understanding error propagation in deep learning neural network (dnn) accelerators and applications," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2017, pp. 1–12.

[21] A. Siddique, K. Basu, and K. A. Hoque, "Exploring fault-energy trade-offs in approximate dnn hardware accelerators," in *2021 22nd International Symposium on Quality Electronic Design (ISQED)*. IEEE, 2021, pp. 343–348.

[22] S. Mittal, "A survey on modeling and improving reliability of dnn algorithms and accelerators," *Journal of Systems Architecture*, vol. 104, p. 101689, 2020.

[23] G. Ammes, G. B. Manske, P. F. Butzen, A. I. Reis, and R. P. Ribas, "Atmr design by construction based on two-level als," in *2023 36th SBC/SBMicro/IEEE/ACM Symposium on Integrated Circuits and Systems Design (SBCCI)*. IEEE, 2023, pp. 1–6.

[24] H. R. Mahdiani, S. M. Fakhraie, and C. Lucas, "Relaxed fault-tolerant hardware implementation of neural networks in the presence of multiple transient errors," *IEEE transactions on neural networks and learning systems*, vol. 23, no. 8, pp. 1215–1228, 2012.

[25] T. Arifeen, A. S. Hassan, and J.-A. Lee, "Approximate triple modular redundancy: A survey," *IEEE Access*, vol. 8, pp. 139 851–139 867, 2020.

[26] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra, "Weight uncertainty in neural network," in *International conference on machine learning*. PMLR, 2015, pp. 1613–1622.

[27] D. Danopoulos, G. Zervakis, K. Siozios, D. Soudris, and J. Henkel, "Adapt: Fast emulation of approximate dnn accelerators in pytorch," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2022.

[28] R. Leveugle, A. Calvez, P. Maistri, and P. Vanhauwaert, "Statistical fault injection: Quantified error and confidence," in *2009 Design, Automation & Test in Europe Conference & Exhibition*. IEEE, 2009, pp. 502–506.

[29] G. Li, S. K. S. Hari, M. Sullivan, T. Tsai, K. Pattabiraman, J. Emer, and S. W. Keckler, "Understanding error propagation in deep learning neural network (dnn) accelerators and applications," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2017, pp. 1–12.

Appendix 4

IV

M. Taheri, M. Daneshtalab, J. Raik, M. Jenihhin, S. Pappalardo, P. Jimenez, B. Deveautour, and A. Bosio, "Saffira: a framework for assessing the reliability of systolic-array-based dnn accelerators," in 2024 27th International Symposium on Design & Diagnostics of Electronic Circuits & Systems (DDECS), pp. 19–24, 2024.

SAFFIRA: a Framework for Assessing the Reliability of Systolic-Array-Based DNN Accelerators

Mahdi Taheri[1*], Masoud Daneshtalab[3,1], Jaan Raik[1], Maksim Jenihhin[1],
Salvatore Pappalardo[2*], Paul Jimenez[2], Bastien Deveautour[2], and Alberto Bosio[2]

[1]Tallinn University of Technology, Tallinn, Estonia
[2]Ecole Centrale de Lyon, Lyon, France
[3]Mälardalen University, Västerås, Sweden

2024 27th International Symposium on Design & Diagnostics of Electronic Circuits & Systems (DDECS) | 979-8-3503-5934-3/24/$31.00 ©2024 IEEE | DOI: 10.1109/DDECS60919.2024.10508925

***Abstract*—Systolic array has emerged as a prominent architecture for Deep Neural Network (DNN) hardware accelerators, providing high-throughput and low-latency performance essential for deploying DNNs across diverse applications. However, when used in safety-critical applications, reliability assessment is mandatory to guarantee the correct behavior of DNN accelerators. While fault injection stands out as a well-established practical and robust method for reliability assessment, it is still a very time-consuming process. This paper addresses the time efficiency issue by introducing a novel hierarchical software-based hardware-aware fault injection strategy tailored for systolic array-based DNN accelerators. The uniform Recurrent Equations system is used for software modeling of the systolic-array core of the DNN accelerators. The approach demonstrates a reduction of the fault injection time up to 3× compared to the state-of-the-art hybrid (software/hardware) hardware-aware fault injection frameworks and more than 2000× compared to RT-level fault injection frameworks — without compromising accuracy. Additionally, we propose and evaluate a new reliability metric through experimental assessment. The performance of the framework is studied on state-of-the-art DNN benchmarks.**

***Index Terms*—hardware accelerator, systolic array, deep neural networks, fault simulation, reliability, resilience assessment**

I. Introduction

Assessing the reliability of a Deep Neural Network (DNN) is not a trivial task: it depends on several factors, such as the training set, the data type, and the quality of the test set [1]. On top of that, we need to consider the hardware that performs the computations [2] since specific platforms have specific faults [3].

Many studies showed that hardware faults can greatly reduce the effectiveness of DNNs [4]. As a result, there is a surge in research efforts to evaluate and enhance the reliability of DNNs. An example of faulty hardware is given in Figure 1. This figure presents the possible fault locations in a DNN inference engine. This example shows the necessity of reliability assessment of DNNs. However, assessing DNN reliability is a challenging task [5].

There are three main methodologies on DNNs' reliability assessment as irradiation-based, platform-based and simulation-based [7] in which simulation-based Fault Injection (FI) is less expensive (in terms of equipment) and thus is also the most used in the research community [2]. The other advantages of FI are the possibility to model different fault scenarios precisely to assess their impact on DNNs without the need for extensive hardware resources and design time, and full control over network parameters and architecture.

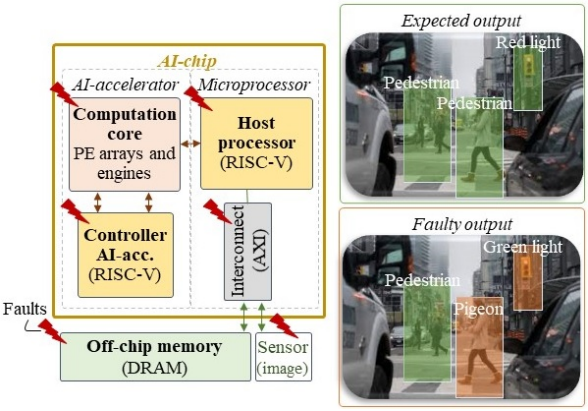


Fig. 1: DNN accelerator hardware reliability threats [6]

On the other hand, DNN hardware-accelerator simulation for FI is computationally expensive and typically demands a substantial amount of time to complete a single inference [8]. This paper introduces a novel simulation flow and FI tailored to significantly accelerate the injection process on systolic-array-based DNN hardware accelerators. The systolic-array core of the DNN accelerators is modeled using the Uniform Recurrent Equations (URE) system. The proposed injection flow has been implemented as an open-source tool named **SAFFIRA**, which stands for **Systolic Array simulator Framework for Fault Injection based Reliability Assessment**. Simulation-based FI is usually done either without considering the underlying hardware or through RTL (Register-Transfer Level) simulations known for their resource-intensive computations and time-consuming nature. SAFFIRA is based on a Systolic Array (SA) simulator, thus offering the advantage of being more precise than a hardware-agnostic tool, but much faster than traditional RTL-level simulations. Experimental results show a reduction of the fault injection time up to 3× compared

* These authors contributed equally

979-8-3503-5934-3/24/$31.00 ©2024 IEEE

to the state-of-the-art hybrid (software/hardware) hardware-aware fault injection frameworks and up to 2000× compared to RT-level fault injection frameworks — without compromising accuracy.

The key contributions of this paper are the following:

- introducing a hierarchical **methodology** for the hardware-accurate reliability assessment of SA using a novel simulation-based fault injection approach by modeling the systolic-arrays using Uniform Recurrent Equations (URE) system;
- presenting an **open-source tool** implementing the aforementioned methodology;
- introducing a new **metric** called **faulty distance** for reliability assessments of DNNs;
- evaluating the performance of the framework on state-of-the-art DNN benchmarks

The rest of this paper is organized as follows. Section II presents the related works. Section III presents the proposed fault injection flow for SA. Section IV shows the experimental setup and results. Section V concludes the paper.

II. Related Works

This section discusses previous works targeting DNNs reliability assessment by using simulation-based FI.

A. Hardware-Agnostic FI Tools

Tools in this category perform fault injection without taking into account the underlying hardware. Some of these are capable of performing FI directly in the DNNs models. In this category, PyTorchFI [9] and TensorFI [10] can inject faults into DNN models respectively implemented in PyTorch, Tensorflow, and Keras. All of these open-source frameworks can inject both permanent and transient faults into weights as well as activations given specific error rates such that it is possible to evaluate the accuracy loss.

Moreover, to further enhance the efficiency, additional FI tools have been introduced. For example, BinFI [11] is an extension of TensorFI that aims at identifying critical bits in DNNs. Another tool, namely LLTFI [12], is able to inject transient faults into specific instructions of DNN models in either PyTorch or TensorFlow.

B. Hardware-Aware FI Tools

These tools can perform FI in software, taking into account the relying hardware using some abstract models of the ‘DNN hardware accelerator.

In [13], the authors used an RTL model of a SA to perform their experiments. Reference [14] maps a DNN into the RTL implementation of the accelerator. They study the effect of transient faults in memory and datapath accurately. In these studies, FI is performed in software while all of its parameters are integrated with the corresponding hardware components. Authors in [15] implemented their DNN and the fault injector in software, inspired by an FPGA-based DNN accelerator. Moreover, in [16], DNN and FI are implemented in Keras, and the architecture of a SA accelerator is considered for a fault-tolerant design. Similarly, authors in [17] evaluate their proposed reliability improvement technique on memories in TensorFlow while injecting transient faults into the weights. PyTorch is used in [18] to implement the DNN, and transient faults are injected into activations (datapath or MAC units) and weights (memory) regarding the SA accelerator model. Reference [19] also uses PyTorch and injects faults by a custom framework called TorchFI to inject faults into the outputs of CONV and FC layers of the network.

The effect of permanent faults at PEs' outputs is studied in [20] where the model of the accelerator is adopted from implementing the DNN in an N2D2 framework [21]. Furthermore, authors in [22] use PyTorch and study permanent faults in MAC units of an accelerator while training to improve the reliability at inference. Authors in [23] developed a Keras-based accelerator simulator to study the effect of permanent faults on the on-chip memory of accelerators by injecting permanent faults into activations and weights. Weight remapping strategy in memory to decrease the effect of permanent faults is evaluated in [24] using Ares. SCALE-Sim [25], a systolic CNN accelerator simulator, is adopted in [26] to study permanent faults in PEs and computing arrays in systolic array-based accelerators.

Similar to the Hardware-Independent platform, faults are injected based on Bit Error Rate (BER), or fault rate, and experiments are repeated to reach 95% confidence level and 1% error margin [16]. In general, the main drawbacks in the existing reliability assessment methods for DNNs can be summarized as follows:

- There is no software FI framework in hardware-aware platforms. Hence, there is a potential for DNN accelerator simulators to be exploited or developed for the reliability assessment of DNNs;
- Several FI research works carry out accuracy loss and fault classification as an evaluation of reliability. Also, some works considered FIT (Failure In Time) [27]. However, there is still an urgent need to present DNN-specific metrics for reliability evaluation. In this work, we are introducing a new metric called faulty distance to provide a better understanding of the network resilience.

III. Proposed Methodology

The proposed methodology for the SAFFIRA framework is illustrated in Fig. 2. After providing the trained network parameters and architecture, in step one, the fault list is generated. Possible fault locations can be defined by the user or can be a random fault list generated based on the network parameters by the framework. Faults can be selected as transient or permanent faults targeting different activations of the DNN. Then, in step two, the fault injection campaign is performed at the systolic-array simulation environment in Python, and the rest of the network is executed at the high-level API (e.g. Pytorch) to speed up the process. In this step, switching between high-level API and systolic-array simulator (2-A) is done by a method called LoLif, which is described in subsection III-A2. Finally, the reliability of the network and the impact of the faults are reported at step three by different metrics.

SAFFIRA supports various data representations, including fixed-point, integer, and floating-point formats. This framework

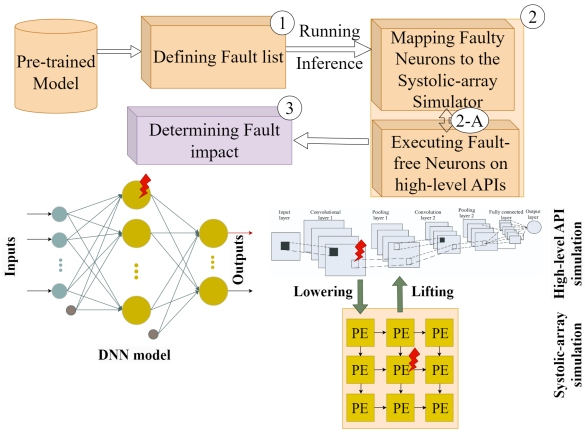


Fig. 2: SAFFIRA methodology

also supports various relevant mapping to systolic-array architecture scenarios (e.g. output stationary, weight stationary, etc.). These flexibilities allow researchers to adapt the framework to different applications and tailor the reliability assessment to specific hardware requirements.

A. Hardware simulations

SAFFIRA is a SA model based on the Uniform Recurrent Equations (URE) system. As described by [28], it is possible to generate a SA that solves the problem described by a URE system. In the case of SAFFIRA, the URE system is the one associated with matrix multiplication since it is the operation deployed on the SA for DNN execution. The following subsection presents the formal details needed to perform a simulation followed by performing fault injection in such a context, and finally, the strategies strictly related to DNNs are covered.

1) Mathematical formalism: A URE system is defined on top of an integer lattice L_n of points p in the n-dimensional Euclidean space E_n. The goal is to solve a system of equations associated with the variables $x_1(p), x_2(p), \cdots, x_m(p)$ for all points $p \in R$, where $R \subseteq L_n$ [28]. This system can be either uni-variate or multi-variate. Here, only uni-variate case is considered, thus the system would have the following form.

$$
\begin{aligned}
x_1(p) &= f[x_1(p-w_1), \cdots, x_m(p-w_p)],\\
x_2(p) &= x_2(p-w_2),\\
&\vdots\\
x_m(p) &= x_m(p-w_p).
\end{aligned}
$$

The points $p - w_{i_k}$ belong to L_n. The vectors w_k are constants independent of p and this is why they are said to have *uniform dependence*. Each equation $x_i(p)$ depends on the points $p - w_{i_k}$.

The authors of [28] showed a strategy to model a SA starting from the problem to solve. Specifically, the authors explain three steps:

1) find a URE system for the problem to solve,
2) find a timing function compatible with the dependencies of the URE system,
3) find an allocation function to map the URE onto a finite architecture.

The main idea is to project the space E_n twice: the first time, the resulting points will correspond to the spatial arrangement of each Processing Element (PE). The second projection determines iso-temporal planes, identifying operations that are computed during the same clock cycle but on different PEs; each plane corresponds to a different clock cycle. The space-projection matrix P and the temporal dimension vector π are used later.

2) Convolutions: The strategy explained above opens the possibility to implement a variety of algorithms as a systolic array. Based on the literature, it is possible to perform a convolution as a matrix multiplication [29]. The experiments shown below are performed using a systolic array to perform the matrix multiplication $C = A \times B$. The associated URE is the following.

$$
\begin{aligned}
c(i,j,k) &= c(i,j,k-1) + a(i,j-1,k) \times b(i-1,j,k)\\
a(i,j,k) &= a(i,j-1,k)\\
b(i,j,k) &= b(i-1,j,k)
\end{aligned}
$$

initial conditions

$$
\begin{aligned}
a(i,0,k) &= a_{ik}, \quad \forall i,k\\
b(0,j,k) &= b_{kj}, \quad \forall k,j\\
c(i,j,0) &= 0, \quad \forall i,j
\end{aligned}
$$

where $p = (i,j,k) \in R \subseteq L_n$, in which i, j and k assume values between 1 and $N1$, $N2$, $N3$ respectively. $N1$, $N2$ and $N3$ are problem parameters such that $A \in \mathbb{R}^{N_1,N_3}, B \in \mathbb{R}^{N_3,N_2}, C \in \mathbb{R}^{N_1,N_2}$.

When it comes to performing a convolution, the input matrices must be *reshaped* such that the result of the SA is a convolution. In this paper, this concept is called LoLif, which stands for Lowering and Lifting strategies. This idea is explained in [29]. If computing a convolution $C = A * B$ is needed, it can be implemented as a transformation lif of the matrix multiplication of transformed matrices $low_a(A) \times low_b(B)$. In formulas:

$$
C = lif(low_a(A) \times low_b(B)),
$$

where Lif, Low_a and Low_b are corresponding transformations, as shown in the example of Fig. 3

3) Simulation and Injections: In order to perform the simulation, it is sufficient to solve the system shown above. Nevertheless, this method gives the possibility of injecting faults in the values in a hardware-aware fashion. To achieve the injection, it is sufficient to change the values $a(p)$, $b(p)$, $c(p)$ for specific points p. The faulty values must then be propagated to the following PEs. Given that each point p is projected to the physical space $r = (x,y)$ using the physical space-projection matrix, $r = Pp$, it can be inferred that how the injected values are propagated through the different PEs. Specifically, for some dependence vector d for the different labeled variables a, b, c.

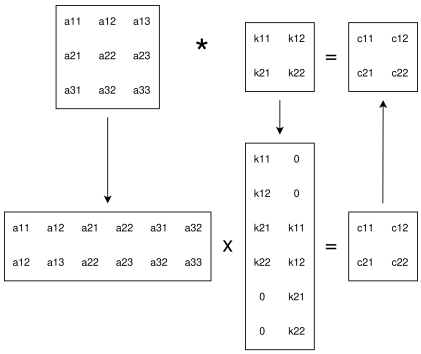


Fig. 3: LoLif example. Applied transformations are similar to im2col and im2row.

Looking at the system above, the following can be observed: $d_a = (0, 1, 0)$, $d_b = (1, 0, 0)$ and $d_c = (0, 0, 1)$. Afterwards, the propagation direction can be found using the same relationship shown before: $\delta x_i = Pd_i, i = \{a, b, c\}$. This means that the value of a in some PE in position s will be propagated to the PE in position $s + \delta x_a$. The same reasoning can be done for the time, supposing that a fault is propagated not only in space but also in time. We can compute $\delta t_i = \pi d_i$. For simplicity, $\pi = (1, 1, 1)$ is fixed to reduce the exploration space. In this case, the time dependency δt_i will always be 1: $\delta t_i = 1$.

Figure 4 shows an example. In this case, an injection in the element $s = (x, y, t)$ on the generic line i is done between times 0 and ∞. The injected elements are visible in the figure. Specifically, the fault will propagate in time, thus injecting also $s + \delta t_i$ and $s + 2\delta t_i$. In the same way, this fault will propagate in space, to the element cascading from s. Note that the value propagation only happens after each clock cycle. This means that the next injected element will be displaced also in time, thus injecting element $s + \delta x_i + \delta t_i$. In the same way, the latter will propagate to the following element on the following clock cycle, thus injecting element $s + 2\delta x_i + 2\delta t_i$ and so on.

The set of points belonging to the injection can be transposed back into the iteration space E_n using the pseudo-inverse P^{-1}. The set of points identified with this strategy will be subject to injection. Formally, injection is as a function h applied to a variable:

$$a(p) = h(a(p - w_a))$$

IV. Experiments and Results

Two different sets of experiments are performed using SAFFIRA. First, a fault injection based on the permanent-fault model is performed on two different quantization versions of the LeNet-5 network (8-bit and 16-bits integers). The second set of experience is performing fault injection based on the transient fault model in the three different benchmarks (AlexNet, VGG-16 and ResNet-18). All networks are fully quantized to INT data type, including all activations, weights, and biases. The base accuracies are reported in the table I

The SA model for these experiments is output stationary. This means that its physical-space projection matrix P is as follows:

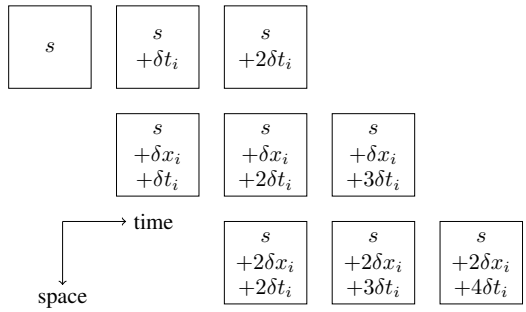


Fig. 4: When injecting element s, the fault is propagated in time (thus affecting elements $s + \delta t_i$ and $s + 2\delta t_i$) and in space (forwarding the faulty value to neighboring elements $s + \delta x_i + \delta t_i$, $s + 2\delta x_i + \delta t_i$ and so on).

TABLE I: Base accuracy of networks under test

DNN	accuracy (%)
8-bit LeNet-5 (MNIST)	93.8
16-bit LeNet-5 (MNIST)	95.4
AlexNet (CIFAR-10)	78.0
VGG-16 (CIFAR-10)	93.4
ResNet-18 (CIFAR-10)	93.8

$$P = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$$

Such a matrix corresponds to a rectangular SA with $N1 \times N2$ PEs. Please note that with this projection, the variable c (i.e. the partial sum) is a *stationary variable* since it is always available on the same PE regardless of the iteration. Whether a variable is stationary or not depends on the employed projection.

In all experiments, fault injection is repeated several times to reach an acceptable confidence level, based on [30]. This work provides an equation to reach 95% confidence level and 1% error margin.

A. Fault Classification

The DNN resilience is evaluated by comparing the output probability vector of the golden run (i.e. the DNN that behaves as expected, without faults) and the faulty run (i.e. the DNN that includes the fault). The Silent Data Corruption (SDC) rate is defined as the proportion of faults that caused misclassification in comparison with the golden model [31].

In addition, the targeted hardware reliability can be calculated by differentiating SDC rates of injected transient faults into defined classes and calculating Failures In Time (FIT) for the accelerator (*accel*) by its components (*comp*) with (IV-A) in which FIT_{raw} is provided by the manufacturer, $Size_{comp}$ is the total number of the component bits, and SDC_{comp} is obtained by FI.

$$FIT_{accel} = \sum_{comp} FIT_{raw} \times Size_{comp} \times SDC_{comp}$$

Finally, **faulty distance** is proposed. This metric can be used to evaluate the resilience of classifications DNNs. Supposing the golden probability vector is G, the faulty probability vector is

F and the function $ag(\cdot)$ corresponds to the argmax function, then the faulty distance function d_f is defined as follows.

$$d_f = (1 - \frac{G \cdot F}{||G|| \cdot ||F||}) \cdot (ag(F) - ag(G))$$

In this metric, cosine similarity is being used $cos\theta = \frac{G \cdot F}{||G|| \cdot ||F||}$. Cosine similarity serves as a metric for assessing the resemblance between two non-zero vectors within an inner product space. Representing the cosine of the angle between the vectors, this measure calculates similarity by normalizing their dot product. In our study, we utilize cosine similarity to evaluate the entirety of generated probabilities across various classes in both faulty and golden modes. The cosine similarity metric yields values within the range of -1 to 1. Proximity to 1 signifies a high degree of similarity between vectors. Therefore, the faulty distance metric gives 0 when the faulty output corresponds to the correct classification. The bigger the metric, the worse the misclassification is.

B. Results

Table II shows the results of the FI for permanent fault injection experiments on LeNet-5 with the different metrics. It can be seen that this network was highly susceptible to the injected permanent faults. Specifically, the SDC-1 and SDC-5 are very high: on average, about 82% of the time, the faulty inference misclassified the input; furthermore, about 93.5% of the inputs were completely missed since the correct label was below the fifth position. The SDC-10% and SDC-20% rates are very high as well: more than 95% of the inputs had the correct class with a probability much too low than expected. Average Faulty Distance (AFD) is also reported that shows the 16-bit network in this particular case, is more reliable compared to the 8-bit network in the presence of permanent faults in the systolic architecture.

These results show that the DNN used was not usable in a safety-critical environment. This result was expected since the network was not trained to withstand stuck-at faults like the ones injected.

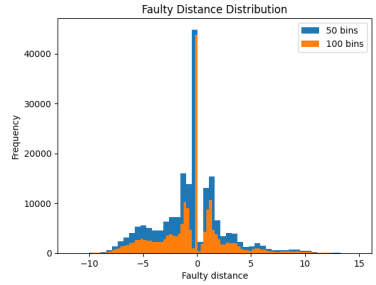
TABLE II: FI experiments results on two LeNet-5

Metric	16bit	8bit
SDC-1 (%)	77.84	87.70
SDC-5 (%)	93.05	94.49
FIT (failures/10^9 hours)	4.9e-4	5.0e-4
SDC-10% (%)	98.16	98.53
SDC-20% (%)	96.21	96.97
AFD	-0.04	-0.53

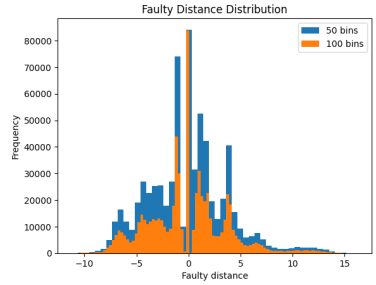
For the second experiment, only SDC and AFD are reported in table III.

C. Faulty Distance

In the previous subsection, only the average faulty distance was shown. Nevertheless, this metric can be looked through with more details when plotted as a histogram. Figure 5 shows the histograms (both with 50 and 100 bins) of the metrics per each experiment. It is possible to see a peak at 0, which corresponds to all the correctly classified inputs. The height



(a) Faulty Distance on the 8bit network



(b) Faulty Distance on the 16bit network

Fig. 5: Histogram plot of the Faulty distance values

of that column is precisely the same as the complement of the SDC-1 metric. On top of that, it is possible to see two different, yet similar, trends for the two networks. Figure 5a shows other three peaks: around $+1$, -1 and -5. This means that, although most of the inputs were mis-classified, the difference with the golden vector was not extremely big, in general. On the other hand, figure 5b shows many more peaks, this means that it is more difficult to predict how a fault will propagate in this case.

D. Computation Time

The experiments were performed on a server using python3 with an Intel Xeon Silver 4210, with a total number of 40 cores. SAFFIRA completes 500 inferences of two convolutional layers, with the same systolic array, in about 10 minutes with minimal optimization. This means a total of about 16.3 simulations per second. For comparison, by utilizing the framework presented in [32] to perform fault injection on the same networks as this work, on average, 5.8 simulations per second are executed. The mentioned framework is the state-of-the-art hybrid (software/hardware codesign) hardware-aware fault injection framework. Therefore, SAFFIRA provides about 2.8× speed up by performing the same analysis. Also, the same fault injection campaign is performed at the RT level using QuestaSim. The results show 0.007 simulation per second, which is 2100× slower than the proposed method in this work.

V. Conclusions

This paper presents a novel hierarchical fault injection strategy for systolic arrays, addressing the time efficiency issue by introducing a novel hierarchical software-based hardware-aware fault injection strategy tailored for systolic array-based DNN

TABLE III: Reliability analysis of different state-of-the-art DNN benchmarks

DNN	SDC-1	SDC-5	SDC-10%	SDC-20%	AVF
AlexNet (CIFAR-10)	4.3	29.1	13.1	9.7	7.1×10^{-2}
VGG-16 (CIFAR-10)	3.0	40.0	46.5	84.5	1.9×10^{-3}
ResNet-18 (CIFAR-10)	1.5	23.0	16.5	82	1.6×10^{-3}

implementations. The approach demonstrates a reduction of the fault injection time up to threefold compared to the state-of-the-art hybrid (software/hardware) hardware-aware fault injection frameworks and more than 2000× compared to RT-level fault injection frameworks — without compromising accuracy. Additionally, we propose and evaluate a new reliability metric through experimental assessment. The performance of the framework is studied on state-of-the-art DNN benchmarks.

VI. Acknowledgement

This work was supported in part by the Estonian Research Council grant PUT PRG1467 "CRASHLESS" and by Estonian-French PARROT project "EnTrustED".

References

[1] M. Taheri, "Dnn hardware reliability assessment and enhancement," in *27th IEEE European Test Symposium (ETS)*, 2022.

[2] M. H. Ahmadilivani *et al.*, "A systematic literature review on hardware reliability assessment methods for deep neural networks," *arXiv preprint arXiv:2305.05750*, 2023.

[3] A. Bosio, I. O'Connor, M. Traiola, J. Echavarria, J. Teich, M. A. Hanif, M. Shafique, S. Hamdioui, B. Deveautour, P. Girard, *et al.*, "Emerging computing devices: Challenges and opportunities for test and reliability," in *2021 IEEE European Test Symposium (ETS)*, pp. 1–10, IEEE, 2021.

[4] M. Taheri *et al.*, "Appraiser: Dnn fault resilience analysis employing approximation errors," in *2023 26th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS)*, pp. 124–127, IEEE, 2023.

[5] M. Taheri *et al.*, "Deepaxe: A framework for exploration of approximation and reliability trade-offs in dnn accelerators," in *2023 24th International Symposium on Quality Electronic Design (ISQED)*, pp. 1–8, IEEE, 2023.

[6] M. Taheri, N. Cherezova, M. S. Ansari, M. Jenihhin, A. Mahani, M. Daneshtalab, and J. Raik, "Exploration of activation fault reliability in quantized systolic array-based dnn accelerators," *arXiv preprint arXiv:2401.09509*, 2024.

[7] A. Ruospo *et al.*, "A survey on deep learning resilience assessment methodologies," *Computer*, vol. 56, no. 2, pp. 57–66, 2023.

[8] M. H. Ahmadilivani *et al.*, "Special session: Approximation and fault resiliency of dnn accelerators," in *2023 IEEE 41st VLSI Test Symposium (VTS)*, pp. 1–10, IEEE, 2023.

[9] A. Mahmoud *et al.*, "Pytorchfi: A runtime perturbation tool for dnns," in *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, pp. 25–31, IEEE, 2020.

[10] N. Narayanan *et al.*, "Fault injection for tensorflow applications," *IEEE Transactions on Dependable and Secure Computing*, 2022.

[11] Z. Chen *et al.*, "Binfi: an efficient fault injector for safety-critical machine learning systems," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–23, 2019.

[12] U. K. Agarwal, A. Chan, and K. Pattabiraman, "Lltfi: Framework agnostic fault injection for machine learning applications (tools and artifact track)," in *2022 IEEE 33rd International Symposium on Software Reliability Engineering (ISSRE)*, pp. 286–296, IEEE, 2022.

[13] S. Pappalardo *et al.*, "Resilience-performance tradeoff analysis of a deep neural network accelerator," in *2023 26th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS)*, pp. 181–186, IEEE, 2023.

[14] A. Azizimazreah *et al.*, "Tolerating soft errors in deep learning accelerators with reliable on-chip memory designs," in *2018 IEEE International Conference on Networking, Architecture and Storage (NAS)*, pp. 1–10, IEEE, 2018.

[15] W. Li *et al.*, "Soft error mitigation for deep convolution neural network on fpga accelerators," in *2020 2nd IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, pp. 1–5, IEEE, 2020.

[16] E. Ozen and A. Orailoglu, "Low-cost error detection in deep neural network accelerators with linear algorithmic checksums," *Journal of Electronic Testing*, vol. 36, no. 6, pp. 703–718, 2020.

[17] M. Jasemi, S. Hessabi, and N. Bagherzadeh, "Enhancing reliability of emerging memory technology for machine learning accelerators," *IEEE Transactions on Emerging Topics in Computing*, vol. 9, no. 4, pp. 2234–2240, 2020.

[18] E. Ozen and A. Orailoglu, "Boosting bit-error resilience of dnn accelerators through median feature selection," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 11, pp. 3250–3262, 2020.

[19] B. F. Goldstein *et al.*, "A lightweight error-resiliency mechanism for deep neural networks," in *2021 22nd International Symposium on Quality Electronic Design (ISQED)*, pp. 311–316, IEEE, 2021.

[20] S. Burel, A. Evans, and L. Anghel, "Mozart+: Masking outputs with zeros for improved architectural robustness and testing of dnn accelerators," *IEEE Transactions on Device and Materials Reliability*, vol. 22, no. 2, pp. 120–128, 2022.

[21] ""N2D2 CAD framework for DNNs"." https://github.com/cea-list/N2D2. [Online].

[22] L.-H. Hoang, M. A. Hanif, and M. Shafique, "Tre-map: Towards reducing the overheads of fault-aware retraining of deep neural networks by merging fault maps," in *2021 24th Euromicro Conference on Digital System Design (DSD)*, pp. 434–441, IEEE, 2021.

[23] Y.-Y. Tsai and J.-F. Li, "Evaluating the impact of fault-tolerance capability of deep neural networks caused by faults," in *2021 IEEE 34th International System-on-Chip Conference (SOCC)*, pp. 272–277, IEEE, 2021.

[24] T.-H. Nguyen *et al.*, "Low-cost and effective fault-tolerance enhancement techniques for emerging memories-based deep neural networks," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pp. 1075–1080, IEEE, 2021.

[25] A. Samajdar *et al.*, "Scale-sim: Systolic cnn accelerator simulator," *arXiv preprint arXiv:1811.02883*, 2018.

[26] Y. Zhao, K. Wang, and A. Louri, "Fsa: An efficient fault-tolerant systolic array-based dnn accelerator architecture," in *2022 IEEE 40th International Conference on Computer Design (ICCD)*, pp. 545–552, IEEE, 2022.

[27] G. Li, S. K. S. Hari, M. Sullivan, T. Tsai, K. Pattabiraman, J. Emer, and S. W. Keckler, "Understanding error propagation in deep learning neural network (dnn) accelerators and applications," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–12, 2017.

[28] P. Quinton, "Automatic synthesis of systolic arrays from uniform recurrent equations," *ACM SIGARCH Computer architecture news*, vol. 12, no. 3, pp. 208–214, 1984.

[29] S. Hadjis *et al.*, "Caffe con troll: Shallow ideas to speed up deep learning," in *Proceedings of the Fourth Workshop on Data analytics in the Cloud*, pp. 1–4, 2015.

[30] R. Leveugle, A. Calvez, P. Maistri, and P. Vanhauwaert, "Statistical fault injection: Quantified error and confidence," in *2009 Design, Automation & Test in Europe Conference & Exhibition*, pp. 502–506, IEEE, 2009.

[31] G. Li, S. K. S. Hari, M. Sullivan, T. Tsai, K. Pattabiraman, J. Emer, and S. W. Keckler, "Understanding error propagation in deep learning neural network (dnn) accelerators and applications," in *SC17*, 2017.

[32] S. Pappalardo *et al.*, "A fault injection framework for ai hardware accelerators," in *2023 IEEE 24th Latin American Test Symposium (LATS)*, IEEE, 2023.

Appendix 5

V

M. Taheri, et al., "Appraiser: Dnn fault resilience analysis employing approximation errors," in DDECS, pp. 124–127, 2023.

2023 26th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS) | 979-8-3503-3277-3/23/$31.00 ©2023 IEEE | DOI: 10.1109/DDECS57882.2023.10139468

APPRAISER: DNN Fault Resilience Analysis Employing Approximation Errors

Mahdi Taheri[1], Mohammad Hasan Ahmadilivani[1], Maksim Jenihhin[1], Masoud Daneshtalab[1,2], and Jaan Raik[1]

[1]Tallinn University of Technology, Tallinn, Estonia
[2]Mälardalen University, Västerås, Sweden
[1]mahdi.taheri@taltech.ee

***Abstract*—Nowadays, the extensive exploitation of Deep Neural Networks (DNNs) in safety-critical applications raises new reliability concerns. In practice, methods for fault injection by emulation in hardware are efficient and widely used to study the resilience of DNN architectures for mitigating reliability issues already at the early design stages. However, the state-of-the-art methods for fault injection by emulation incur a spectrum of time-, design- and control-complexity problems. To overcome these issues, a novel resiliency assessment method called APPRAISER is proposed that applies functional approximation for a non-conventional purpose and employs approximate computing errors for its interest. By adopting this concept in the resiliency assessment domain, APPRAISER provides thousands of times speed-up in the assessment process, while keeping high accuracy of the analysis. In this paper, APPRAISER is validated by comparing it with state-of-the-art approaches for fault injection by emulation in FPGA. By this, the feasibility of the idea is demonstrated, and a new perspective in resiliency evaluation for DNNs is opened.**

***Index Terms*—Deep Neural Networks, approximate computing, fault injection, reliability, resiliency assessment**

I. Introduction

In recent years, Deep Neural Networks (DNNs) surpassed human-level precision [1] that made them attractive for several safety-critical applications such as autonomous driving [2]–[4]. Faults that can be caused by soft errors, aging, etc., are the source of threatening the reliability of DNN inference hardware accelerators. Here, *soft errors*, are of particular concern for researchers in the industry and academia. It is a class of faults caused by ionized particles hitting transistors that can flip a logic value in a memory cell or a logic gate.

In today's applications, network parameters, e.g., weights, occupy most of the inference accelerator's areal footprint, making them natural targets for soft-errors-caused disturbances. Unlike other logic structures, DNNs are known to be relatively resilient to transient faults. However, in practice, such faults still may cause a significant accuracy drop in DNNs because of the large area and memory requirements for the state-of-the-art DNNs accelerators. Although numerous techniques have been proposed recently to evaluate the architectural fault resilience of DNNs, they are still rather costly. Throughout the literature, Fault Injection (FI) is the most commonly used method for resilience evaluation of DNNs.

Fault injection by emulation in hardware, usually in FPGAs, is widely adopted by the industry [5] because of its ability to evaluate real-scale DNN accelerator designs with significantly shorter run times compared to software-based simulation.

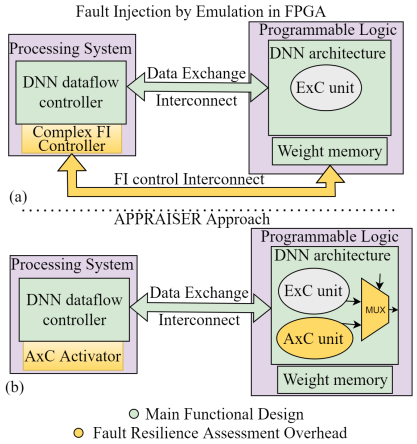


Fig. 1: DNN fault resiliency assessment methods: (a) Fault injection by emulation in FPGA; (b) APPRAISER approach using errors by AxC units.

However, the state-of-the-art approaches for fault injection by emulation in hardware imply iterative procedures for each injected fault, including numerous extra memory accesses, which make them time-consuming and imply complex implementation. Fig. 1(a) illustrates the execution overheads of the general flow of FI by emulation in hardware. In particular, such an iterative approach is breaking the pipeline and requires a complex FI Controller and an extra FI control interconnect [6]–[9]. Fig. 1 (b) illustrates the proposed approach APPRAISER, which allows reducing the fault resiliency assessment overheads. The ability to tolerate the impact of faults on the output accuracy is called *fault resiliency* and, in practice, it is one of the contributors to the final DNN accelerators' reliability [10].

In this paper, our contribution is a novel method of fault resiliency analysis for DNN architectures that applies functional approximation for a non-conventional purpose and harnesses approximate computing errors for its interest. To the best of our knowledge, for the first time, *Approximate Computing* (AxC) units are adopted to improve the processing time-, design-, and control-complexity for DNN fault resiliency analysis process.

APPRAISER provides a rapid exploration of different options of the network architecture, training, dataset, etc., to study the fault resilience of the DNNs. In particular, it enables efficient analysis of subsequent layers' resilience to faults in the weights of a compromised layer.

The new method has the following advantages:

979-8-3503-3277-3/23/$31.00 ©2023 IEEE

- It eliminates the need for designing and deploying an extra complex controller for the fault injection procedure. A simple approximate units enabling circuitry (AxC Activator) is employed instead.
- The inference pipeline process executes a batch of inputs with no need to break this process.
- The resilience assessment process is performed without an extra interconnect for weight sampling.
- The proposed approach is not iterative for each potential fault location, unlike the traditional fault injection. Thus, the analysis complexity is vastly reduced.

The rest of the paper is organized as follows. An analysis of Related Works in Section II is followed by the new methodology presented in Section III. The experimental results, along with their discussion, are presented in section IV. Finally, this work is concluded in Section V.

II. Related works

The extensive growth of the memory footprint size in today's practical DNN inference HW accelerators increases the chances of soft errors' occurrences causing prediction failures. Even a minor change in the DNN architecture may cause a notable difference in the DNNs' architectural fault resiliency [11]. Evaluating the resiliency of DNNs with FI by emulation in hardware is a practical method used today by the industry. There are several works emulating fault injection on FPGAs as a hardware platform.

Fiji-FIN [9] is one of such DNNs' resiliency evaluation frameworks. It considers the model's accuracy degradation as a metric to study the impact of soft errors on the network's parameters, such as weights and activation. Unfortunately, it implies severe effort for designing the fault injection campaigns. For each single fault injection, the execution of the inference should be halted for manipulating the DNN parameters, and it has to be resumed thereafter. It means that the classification time for a batch of inputs should be interrupted to apply fault injection between the classification process of two consecutive inputs.

A similar method is also used in [7], [8]. These works also propose injecting transient faults into on-chip memories of the design implemented on the FPGA. In these works, the bit stream file of the FPGA is obtained by a High-Level Synthesis (HLS) tool and imported to the FPGA. While the system is running, the faults are generated and injected by the embedded processor and the reliability is evaluated in comparison with the golden model.

In contrast to the works mentioned above, this paper proposes a novel non-iterative fault resilience analysis by exploiting the approximation errors instead of fault injection It enables keeping the inference pipeline process to be executed on a batch of inputs unbroken.

III. Proposed Method APPRAISER

The proposed approach for applying errors of approximate computing units for DNN fault resiliency assessment is outlined in Fig. 1(b). An AxC Activator unit on the Processing System (PS) side enables the AxC units to induce errors. These units are AxC multipliers in the multiply-and-accumulate units (MACs), in the targeted (mimicked to be compromised) layer of the DNN. This activator controls the multiplexers on the Programmable Logic (PL) side to switch between the exact implementation of the units (for the functional mode) and the approximated one (for the resiliency assessment mode). Then, the user runs the inference just once for the validation dataset and stores the results of the layers' outputs.

The flow of APPRAISER method is depicted in Fig. 2. Step 1 is the initialization that includes the selection of the compromised layer (e.g. one by one in the DNN structure), the validation testset (i.e. DNN inputs), and the assumed application-specific fault rate. In Step 2, suitable AxC units are selected. For example, in this work, we used AxC multipliers from the EvoApproxLib library [12]. Further, a set of ExC units are substituted with the AxC units in the network architecture (Step 3). The DNN inference is executed keeping the pipeline of the network and the *DNN output accuracy drop* is reported. It is used as the main DNN fault resilience analysis metric. The more the accuracy drops with the induced errors, the less fault-resilient the given DNN implementation is.

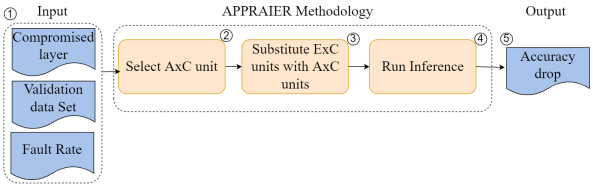


Fig. 2: APPRAISER Methodology

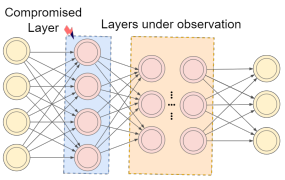


Fig. 3: APPRAISER assessment flow: Compromised layer in the presence of faults in weights vs. layers under resiliency test

In a traditional application of AxC, the approximation of hardware components is based on their inexact implementation that creates a functionally tolerable mismatch with the specification while providing gains in compute-efficiency. In practice, there is an error induced by approximation that can also be employed to mimic the error caused by a fault in the inputs of a logic circuit that is propagated to the output. Such approximation-induced errors affect their corresponding outputs, which are also connected to several other neurons in subsequent layers (as their activation) (Fig. 3). The characteristics of the approximation-induced errors can be assessed by several metrics, normalized error, number of flipped bits, and their impact on the neural network classification accuracy drop. In this study, we rely on the following simple set of metrics:

1) *normalized error*: calculated as the average error on the output of each layer by subtracting the neurons' outputs of that layer from the golden output and dividing all the error values to the maximum value;
2) *network accuracy and recall drop*: calculated by executing the network under different circumstances (faulty vs approximated) over the test set;

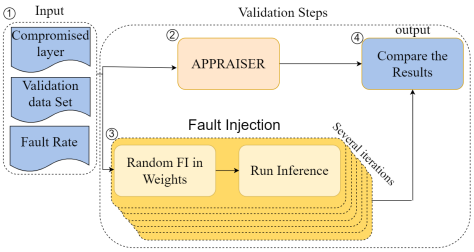


Fig. 4: APPRAISER evaluation flow

3) *bitflips in subsequent layers*: calculated by comparing all bits in the next layers' outputs with the golden model and counting the bits that do not match as flipped bits.

The main objective of APPRAISER is the study of the resiliency of DNN architecture layers to faults that might occur in the weights of a compromised layer. By using this method, the user can rapidly explore the options of network architecture, training, dataset, etc., in terms of fault resiliency analysis. Unlike some other frameworks (e.g., FijiFin [9], APPRAISER does not support assessing the reliability of the network to faults in the activations and DNN neurons and currently is only aimed at resiliency to faults in stored DNN weights. Other limitations are a lower diagnostic capability and implicit correspondence to traditional fault injection based metrics (e.g. in standards).

IV. Experimental Results

A. Evaluation Methodology

The flow to evaluate the proposed method is illustrated in Fig. 4. Here, Steps 1 and 2 repeat the APPRAISER method execution (Fig. 2). The list of candidate approximate multipliers from the EvoApproxLib library [12] was narrowed down with several relevant metrics adopted from EvoApproxLib with the main focus on two established features (Variance of Error Distance (Var-ED) and Root Mean Square (RMS-ED)) presented in [13]. These two metrics are the most critical approximation-induced errors' features for the performance of an AxC unit in DNNs. Based on these metrics, Mult8s_1KX2 (further referred to as *Mult1*) and Mult8s_1KRC (*Mult2*) multipliers are selected for the experiment. For the reference part, the fault resiliency evaluation is repeated on the original network instrumented for a state-of-the-art FI method [9] (Step 4). Two fault models are considered in this study:

- Injection of a *single bitflip* at a random location in all weight bits of the compromised layer for every input in the DNN validation test set,
- Injection of *double bitflips* in weights of the compromised layer for every input in the DNN validation test set.

For each fault model, the experiment is repeated for 1000 random faults per image are considered to reach the 95% FI confidence level according to the statistical fault injection approach [14] and in the end, the average accuracy of all repetitions is reported. Finally, the DNN accuracy drops as a result of applying approximation and fault injection along with normalized error, and the number of flipped bits are compared (Step 5).

B. Experimental Setup

To evaluate the feasibility of the proposed method, a simple Convolutional Neural Network (CNN) with two convolutional layers, two max-pooling, and one Fully-Connected (FC) layer was implemented and trained. The simulations were performed on an Intel® Core™ i7-6800K CPU @ 3.40GHz × 12, and the proposed method was implemented with Python 3. The hardware synthesis and implementation results are produced by the Xilinx Vivado HLS tool on a Xilinx Spartan-7 FPGA (xc7s100-fgga676-1) at 100 MHz operational frequency.

The CNN under study is trained on a dataset of 2000 images of animals (cats and dogs) and humans for binary classification. The accuracy of the network over the test set (including 450 images of animals and humans) is 93.34%. Bit truncation quantization is applied in network parameters during training and data precision is reduced to 8-bit.

C. Evaluation Results

The similarity of the fault resiliency analysis results by fault injection emulation and using the APPRAISER method is analyzed using the metrics identified in Section III.

Fig. 5 illustrates *normalized error* distribution in the output of the second convolutional layer (Conv2), in the presence of random double faults in the first convolution layer (dashed grey) vs errors induced by approximate multipliers (Mult1 solid orange, Mult2 solid blue) enabled in the first convolution layer respectively. Fig. 6 reports the result of applying FI and APPRAISER on the same convolutional layer and its impact on the second pooling layer of the network. These results demonstrate the similarity of the trends in error propagation by the proposed and the reference methods.

Table I reports fault resiliency assessment by the proposed and the reference methods using the *bitflips in subsequent layers* and the *DNN accuracy and recall drop* metrics. These results also demonstrate the strong similarity of the trends in error propagation by the proposed and the reference methods.

Table II demonstrates that although APPRAISER is more resource hungry, it is vastly faster than the reference fault injection by emulation method. It should be noted that the extra resources required by APPRAISER or FI are used only for the fault resiliency analysis phase and cleaned out from the final inference accelerator. In this example, the original CNN occupies 12% of the FPGA resources (LUTs). The CNN instrumented with APPRAISER occupies 29% of the FPGA resources and provides the accuracy/recall drop measurement for fault resiliency assessment in 131 ms, i.e. the same time as the original network execution time. On the other hand, the CNN instrumented with FI utilizes 23% of the FPGA resources and performs the measurement in 632,000 ms, i.e. thousands of times (specifically, 4,824 times in this example) slower than the proposed method. This gain is composed of three components: a) processing of a single image in the CNN instrumented with APPRAISER is 0.29 ms vs 1.40 ms in the CNN instrumented with FI; b) APPRAISER pipelines the processing through the layers while FI has to break the pipeline; c) FI needs numerous iterations for each image to inject the faults (single, double or multiple) at different locations, one combination at a time, while APPRAISER uses only one iteration for each image. Therefore, the time difference becomes even more drastic when comparing these methods for deeper networks (determining the number of layers in the inference execution pipeline) or DNNs with a larger memory for storing weights (determining the number of potential fault locations).

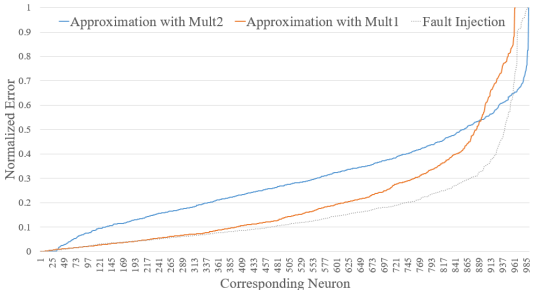


Fig. 5: Normalized output error of Conv2: Applying AxC and fault injection on the Conv1

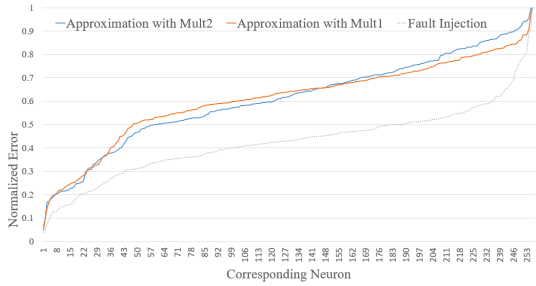


Fig. 6: Normalized output error of Pool2: Applying AxC and fault injection on the Conv1

TABLE I: Bitflips and Accuracy/Recall drop induced by APPRAISER vs the reference fault injection method

Affected/Measured Layers	Bitflips in subsequent layers					
	Injection of a single fault			Injection of a double fault		
	Fault Injection (reference) [%]	Approximation with MULT1 [%]	Approximation with MULT2 [%]	Fault Injection (reference)[%]	Approximation with MULT1 [%]	Approximation with MULT2 [%]
Conv1/Conv1	10.00	9.97	9.98	9.99	10.00	9.99
Conv1/Pool1	9.03	9.03	9.03	9.06	9.06	9.05
Conv1/Conv2	16.73	16.72	16.74	16.74	16.74	16.74
Conv1/Pool2	16.40	16.45	06.50	16.55	16.50	16.45
Conv1/FC	9.25	9.25	8.50	9.30	9.30	9.30
Conv2/Conv2	16.71	16.72	16.71	16.76	16.74	16.74
Conv2/Pool2	16.40	16.45	16.41	16.50	16.50	16.50
Conv2/FC	10.10	8.50	7.80	10.10	9.30	8.30
Affected Layer	DNN Accuracy/Recall drop					
Conv1	2.3/4.7	2.7/8.0	2.2/6.7	4.7/14.0	5.8/17.4	4.2/12.7
Conv2	1.8/6.0	1.6/5.0	2.7/8.0	9.1/26.4	9.1/26.4	8.9/26.7

TABLE II: Overheads of APPRAISER vs the reference fault injection method (Conv1 layer)

Network	Area LUT utilization	Analysis Control Circuitry	Interconnects	DNN execution time in FPGA
Base CNN	12%	N/A	Data Exchange Interconnect	131ms
Fault Resilience Assessment				
CNN instrumented with FI	**23%**	Complex FI Controller	(Data Exchange + FI) Interconnect	632,000ms
CNN instrumented with APPRAISER	∼29%	**Simple AxC Activator**	**Data Exchange Interconnect**	**131ms**

V. Conclusion

The state-of-the-art methods for fault injection by emulation incur a spectrum of time-, design- and control-complexity problems. To overcome these issues, a novel resiliency assessment method called APPRAISER is proposed that applies functional approximation for a non-conventional purpose and employs approximate computing errors for its interest. By adopting this concept in the resiliency assessment domain, APPRAISER provides thousands of times speed-up in the assessment process, while keeping high accuracy of the analysis. In this paper, APPRAISER is validated by comparing it with state-of-the-art approaches for fault injection by emulation in FPGA. By this, the feasibility of the idea is demonstrated, and a new perspective in resiliency evaluation for DNNs is opened.

References

[1] D. Silver *et al.*, "Mastering the game of go without human knowledge," *nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[2] M. Al-Qizwini, I. Barjasteh, H. Al-Qassab, and H. Radha, "Deep learning algorithm for autonomous driving using googlenet," in *2017 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2017, pp. 89–96.

[3] M. Taheri, "Dnn hardware reliability assessment and enhancement," *27th IEEE European Test Symposium (ETS)*., May 2022.

[4] S. Mozaffari, O. Y. Al-Jarrah, M. Dianati, P. Jennings, and A. Mouzakitis, "Deep learning-based vehicle behavior prediction for autonomous driving applications: A review," *IEEE T-ITS*, 2020.

[5] Y. Ibrahim, H. Wang, J. Liu, J. Wei, L. Chen, P. Rech, K. Adam, and G. Guo, "Soft errors in dnn accelerators: A comprehensive review," *Microelectronics Reliability*, vol. 115, p. 113969, 2020.

[6] M.-C. Hsueh, T. K. Tsai, and R. K. Iyer, "Fault injection techniques and tools," *Computer*, vol. 30, no. 4, pp. 75–82, 1997.

[7] N. Khoshavi, C. Broyles, and Y. Bi, "Compression or corruption? a study on the effects of transient faults on bnn inference accelerators," in *2020 21st International Symposium on Quality Electronic Design (ISQED)*. IEEE, 2020, pp. 99–104.

[8] N. Khoshavi, A. Roohi, C. Broyles, S. Sargolzaei, Y. Bi, and D. Z. Pan, "Shieldenn: Online accelerated framework for fault-tolerant deep neural network architectures," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2020, pp. 1–6.

[9] N. Khoshavi, C. Broyles, Y. Bi, and A. Roohi, "Fiji-fin: A fault injection framework on quantized neural network inference accelerator," in *2020 19th IEEE International Conference on Machine Learning and Applications (ICMLA)*. IEEE, 2020, pp. 1139–1144.

[10] J. D. Booth, "Algorithm-based fault tolerance at scale," 2022.

[11] M. Taheri, M. Riazati, M. H. Ahmadilivani, M. Jenihhin, M. Daneshtalab, J. Raik, M. Sjõdin, and B. Lisper, "Deepaxe: A framework for exploration of approximation and reliability trade-offs in dnn accelerators," in *24th International Symposium on Quality Electronic Design*. https://doi.org/10.48550/arXiv.2303.08226, 2023.

[12] V. Mrazek, R. Hrbacek, Z. Vasicek, and L. Sekanina, "Evoapprox8b: Library of approximate adders and multipliers for circuit design and benchmarking of approximation methods," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, March 2017, pp. 258–261.

[13] M. S. Ansari, V. Mrazek, B. F. Cockburn, L. Sekanina, Z. Vasicek, and J. Han, "Improving the accuracy and hardware efficiency of neural networks using approximate multipliers," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 2, pp. 317–328, 2019.

[14] R. Leveugle, A. Calvez, P. Maistri, and P. Vanhauwaert, "Statistical fault injection: Quantified error and confidence," in *2009 Design, Automation & Test in Europe Conference & Exhibition*. IEEE, 2009, pp. 502–506.

Appendix 6

VI

M. Taheri, M. Riazati, M. H. Ahmadilivani, M. Jenihhin, M. Daneshtalab, J. Raik, M. Sjödin, and B. Lisper, "Deepaxe: A framework for exploration of approximation and reliability trade-offs in dnn accelerators," in 2023 24th International Symposium on Quality Electronic Design (ISQED), pp. 1–8, 2023.

2023 24th International Symposium on Quality Electronic Design (ISQED) | 979-8-3503-3475-3/23/$31.00 ©2023 IEEE | DOI: 10.1109/ISQED57927.2023.10129353

DeepAxe: A Framework for Exploration of Approximation and Reliability Trade-offs in DNN Accelerators

Mahdi Taheri[1*], Mohammad Riazati[2*], Mohammad Hasan Ahmadilivani[1],
Maksim Jenihhin[1], Masoud Daneshtalab[1,2], Jaan Raik[1], Mikael Sjödin[2], and Björn Lisper[2]

[1]Tallinn University of Technology, Tallinn, Estonia
[2]Mälardalen University, Västerås, Sweden
[1]mahdi.taheri@taltech.ee

***Abstract*—While the role of Deep Neural Networks (DNNs) in a wide range of safety-critical applications is expanding, emerging DNNs experience massive growth in terms of computation power. It raises the necessity of improving the reliability of DNN accelerators yet reducing the computational burden on the hardware platforms, i.e. reducing the energy consumption and execution time as well as increasing the efficiency of DNN accelerators. Therefore, the trade-off between hardware performance, i.e. area, power and delay, and the reliability of the DNN accelerator implementation becomes critical and requires tools for analysis.**

In this paper, we propose a framework DeepAxe for design space exploration for FPGA-based implementation of DNNs by considering the trilateral impact of applying functional approximation on accuracy, reliability and hardware performance. The framework enables selective approximation of reliability-critical DNNs, providing a set of Pareto-optimal DNN implementation design space points for the target resource utilization requirements. The design flow starts with a pre-trained network in Keras, uses an innovative high-level synthesis environment DeepHLS and results in a set of Pareto-optimal design space points as a guide for the designer. The framework is demonstrated on a case study of custom and state-of-the-art DNNs and datasets.

***Index Terms*—deep neural networks, approximate computing, fault simulation, reliability, resiliency assessment**

I. Introduction

In the past decades, Deep Neural Networks (DNNs) demonstrated a significant improvement in accuracy by adopting intense parameterized models [1]. As a consequence, the size of these models has drastically increased imposing challenges in deploying them on resource-constrained platforms [2]. FPGAs are a widely used solution for flexible and efficient DNN accelerator implementations and have shown superior hardware performance in terms of latency and power [3].

In practice, deployment of a DNN accelerator for the safety- and mission-critical applications (e.g., autonomous driving) requires addressing the trade-off between different design parameters of *hardware performance*, e.g., area, power, delay, and *reliability*. A compromise between conflicting requirements can be achieved by simplifying the implementation to sacrifice the precision of results but benefiting from lower resource utilization, energy consumption, and higher system efficiency. *Approximation Computing (AxC)* is one of such concepts in hardware design [4].

Moreover, the assessment of the reliability of DNN accelerators is a challenging issue by itself. Reliability of DNNs concerns DNN accelerators' ability to execute correctly in the presence of faults [5] originating from either the environment (e.g., soft errors, electromagnetic effects, temperature variations) or from inside of the chip (e.g., manufacturing defects, process variations, aging effects) [6]. The ability to tolerate the impact of faults on the output accuracy is called *fault resiliency* and, in practice, it is one of the contributors to the DNN accelerators' reliability [7]. DNNs are known to be inherently fault-resilient due to the high number of learning process iterations and also several parallel neurons with multiple computation units. Nevertheless, faults may impact the output accuracy of DNNs drastically [8], and in case of resource-constrained critical applications, DNNs' fault resiliency is required to be evaluated and guaranteed [9] [10]. The complexity of such evaluation motivates an *automated tool-chain* with AxC and resiliency analysis to support *Design Space Exploration (DSE)* for DNN accelerators already at the early design stage, i.e. starting from a high-level description.

High-Level Synthesis (HLS) tools bridge high-level programming and hardware implementation and allow overcoming the complexity of the process and reducing the design time. Recently, DNN-tailored HLS tools were proposed, e.g., CNN2gate [11], fpgaConvNet [11] and DeepHLS [12]. Such tools are capable of providing a synthesizable C implementation of DNNs for FPGAs from a high-level description in a language such as e.g., Keras.

This paper presents a novel framework and a fully automated tool-chain DeepAxe to provide a design space exploration for FPGA-based implementation of DNN accelerators by analyzing approximation and soft-error reliability trade-offs. To the best of our knowledge, this is the first framework that holistically considers both the transient fault resiliency and hardware performance of DNN accelerators as design parameters. DeepAxe is empowered by techniques for quantiz-

* These authors contributed equally

ing the networks and providing the capability of substituting the exact computing (ExC) units of the network with AxC units and identifying the optimal design points for selective approximation. DeepAxe uses the Keras description of a DNN as the input and is capable of providing an FPGA-ready approximated and transient-fault-resilient inference implementation of the network based on the design parameters selected based on the DSE results. The main contributions in this work are as follows:

- A methodology for selective approximation of reliability-critical DNNs providing a set of Pareto-optimal DNN implementation design space points for the target resource utilization requirements.
- A framework DeepAxe for holistic exploration of approximation and reliability trade-offs in DNN accelerator FPGA-based implementation that enables assessing the trilateral impact of approximation on accuracy, reliability, and hardware performance.
- Integration of the fully automated DeepAxe tool-chain into the DeepHLS environment.
- Demonstration and validation of the framework on representative custom and state-of-the-art DNNs and datasets.

The rest of the paper is organized as follows. Related works are discussed in Section II, the DeepAxe methodology and framework are presented in Section III, the experimental setup and results are provided in Section IV, and finally, the work is concluded in Section V.

II. Related Works

The advantages of implementing and deploying DNNs on FPGAs are advocated in several recent works. The existing FPGA-based tool-chains to map Convolutional Neural Networks (CNNs) are presented in the surveys [13]–[16]. The FINN framework [17] is released by Xilinx for the exploration of quantized CNNs' inference on FPGAs that also provides customized data-flow architectures for each network. Research works [3] and [18] provide Register-Transfer Level (RTL) models using conventional synthesis tools, e.g., Vivado HLS, where the outputs can be directly synthesized on an FPGA. Heterogeneous systems are also another design strategy in the automated tool-chains that propose hardware-software co-design [18]–[20]. In these designs, computational units, e.g., addition, or multiplication, are mainly implemented on Processing Logic (PL) that is controlled by a control unit in a CPU using a dedicated framework, e.g., OpenCL [21].

Using Fixed-point (FxP) data type instead of Floating Point (FP) is becoming more popular due to the lesser resource utilization while keeping the output accuracy degradation at an acceptable level [22]–[24]. Throughout the literature, comprehensive simulations exist that prove that merely an 8-bit data type for MAC operations in DNN execution is sufficient to provide a practical accuracy along with favorable resource utilization [25], [26]. In this work, we considered 8-bit as the base data type for the simulations and implementations.

A number of works in the literature explore the reliability of the DNNs [27], [28]. Some works examine the impact of different fault models on the basis of a number of layers in DNNs and different data types [29]. Studying the significant impact of transient faults vs permanent faults is also done by [30]. The fault analysis of exact DNNs has drawn a lot of attention in the state-of-the-art research [31], and only recently, researchers have started to investigate also the reliability of approximated DNN accelerators (AxDNNs) [10]. A somewhat expected conclusion in [28] is that the error induced by approximation, along with the faults in the DNN structure, are not evenly propagated. The impact of a fault may differ based on different parameters, like fault type, fault location, the approximation error resiliency for each layer, etc. To the best of our knowledge, none of these works explored the impact of using different combinations of approximated layers of a DNN in the presence of transient faults on the reliability, accuracy and delay/resource utilization of the target DNN accelerator.

The approach proposed in this paper goes beyond the state of the art by establishing a fully automated tool for enabling efficient AxC in FPGA-based DNN accelerators aimed at reliability-critical applications. The proposed DeepAxe framework is integrated into DeepHLS environment [12], which is capable of providing completely synthesizable code for efficient FPGA implementations. In particular, this work extends DeepHLS with fault simulation, resiliency analysis and also the use of AxC. The new features allow providing the designers a guideline to choose optimal configurations based on specific requirements for latency, accuracy, resource utilization, and fault resiliency.

III. Proposed Methodology

Fig. 1 illustrates the methodology flow established in the DeepAxe tool-chain for reliability and hardware performance analysis of approximated DNN hardware accelerators. DeepAxe is a framework taking the DNNs' *Pre-trained Keras model* description as the input. Then, DeepAxe feeds the extracted model parameters through the flow to apply the initialization needed before creating the C code. The design, training and test of the DNNs are performed in Python, the *Preprocessing* step is seamlessly integrated into the same environment and is responsible for extracting the required data for the next step.

DeepAxe also supports quantizing the network down to 8-bit INT as a part of the preprocessing step. For this purpose, a full quantization is implemented, targeting all activations, weights and biases. The framework first takes the description of the network in Keras, and then uses the TFlite library to generate a training-aware quantized network. The user can replace their preferred Keras-based quantization library to the tool-chain for this step. The main output of this step is the quantized network's parameters (i.e., weight/bias) and also the files containing the memory dump of the test data. Specifically, the *Keras to C* step implies converting all the above-mentioned parameters to multidimensional arrays in C format. The output accuracy of the generated network is also provided at this step and is kept as a baseline for the further steps of the methodology.

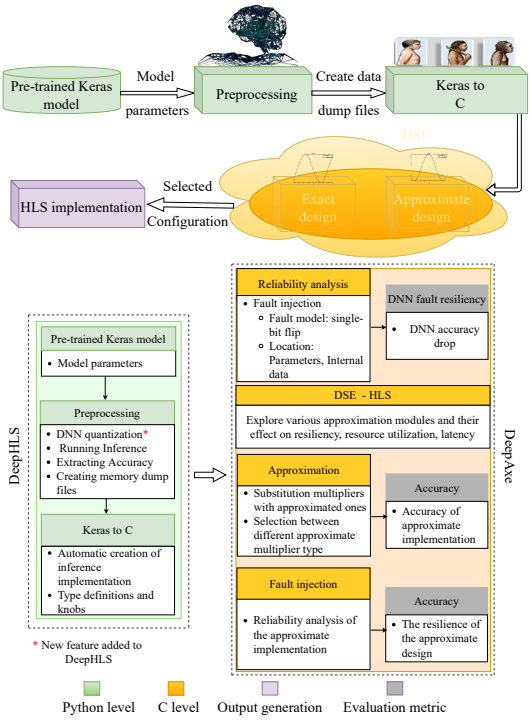


Fig. 1: DeepAxe methodology flow

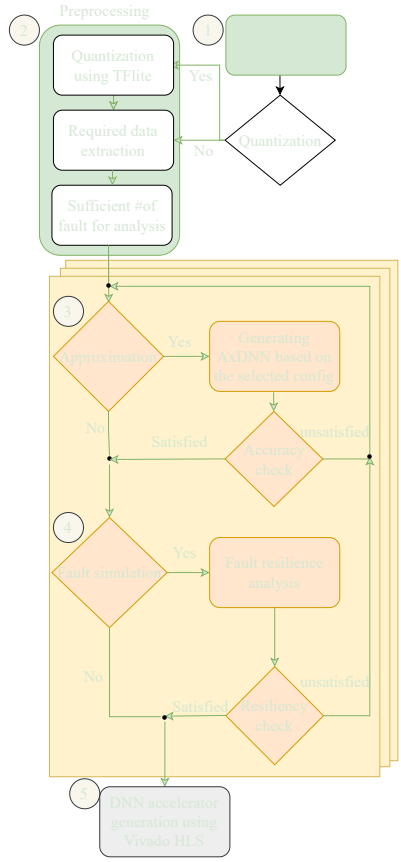


Fig. 2: DeepAxe flowchart

Reliability analysis relies on a fault injection (FI) in C, assuming the single bit-flip faults in the network's activation layers for resiliency assessment. While the multiple-bit fault model is more accurate, it requires a prohibitively large number of fault combinations to be considered ($3^n - 1$ combinations, where n is the number of bits). Fortunately, it has been shown that high fault coverage obtained using the single-bit model results in a high fault coverage of multiple-bit faults [32]. Therefore, a vast majority of practical FI and test methods are based on the single-bit fault assumption.

The reliability analysis step applies the accuracy drop comparison of the network-under-test as the assessment metric. *Approximate design* (see the yellow region in Fig. 1) refers to the selective approximation of DNNs by layers provided by DeepAxe. It instruments the user with the flexibility of choosing between a) different AxC models provided by any library of approximate computing units, such as AxC multipliers in EvoApproxLib, and b) the subset of layers, for setting up different configurations of the network. As an example, in a network with n computing layers (containing both convolutional and fully connected layers), the user has 2^n combinations for exploring the exact and approximate implementations for each layer individually.

After choosing the preferred approximation configuration, the designer can go through the fault injector provided for the resiliency evaluation of the AxDNN. Eventually, the final design can be fed to the *HLS implementation* step for DNN hardware accelerator generation process by the HLS tool.

To illustrate the DeepAxe methodology, the flowchart provided in Fig. 2 shows the step-by-step process from the beginning to the end of DeepAxe tool-chain. After providing the Keras description of the network in Step 1, the user can decide if they need to quantize the network. Then, the preprocessing step can be performed, enabling the user to apply a pre-analysis on the network to extract a sufficient number of faults for the reliability assessment, considering the number of its neurons.

Steps 3 and 4 in Fig. 2 show an iterative process to examine different approximated DNN combinations and, accordingly, their fault resiliency analysis to build the DSE. By enabling the fault simulation process in Step 4, the user can follow the impact of their chosen AxC model and also the approximation configuration on the resiliency of the network compared to the

other AxC model/configurations and also to the exact model. Finally, the selected design and its configuration are fed into the HLS tool for implementation.

It is noteworthy that all steps in the yellow box of Fig. 1 can be iterative, and the user can repeat these steps to find the optimal point based on their requirements. For instance, the user might decide to analyze an assumed approximation configuration, i.e. AxC model for the multiplier and also the layers to approximate. If, after applying approximation, the accuracy check does not satisfy the user, they can try another approximation configuration. Once the requirements are satisfied, it is possible to proceed to the fault vulnerability analysis. If, after applying the fault injection, the resiliency of the network is also satisfying, the next step is generating the DNN accelerator based on the selected configuration.

IV. Experimental Results

A. Experimental Setup

First, all DNNs are implemented, trained and tested in Keras. The required data for further steps of DeepAxe are also generated in the same environment. In the DeepAxe flowchart (Fig. 2), the green parts, including steps 1 and 2, refer to the steps of the framework implemented in this high-level environment. Both a three-layer MLP and LeNet-5, trained on the MNIST dataset, and AlexNet, trained on the CIFAR-10 dataset, are representative DNNs and efficient to perform the validation of the proposed methodology and framework. All networks use ReLu as an activation function. All networks are quantized down to 8-bit INT data type, including all activations, weights, and biases, by using the TFlite [33] library in Python. The yellow parts in Fig. 2 are implemented in C. Simulations are performed on 2 x Intel Xeon Gold 6148 2.40 GHz (40 cores, 80 threads per node) with 96GB RAM. To speed up the simulation process, DeepAxe supports multi-thread parallelism, and users can benefit from this feature based on the number of cores their CPU provides.

All implementations in C are synthesizable by DeepHLS. The approximate multipliers in the C implementation of the network (referring to step 3 in Fig. 2) are adopted from the C codes provided by EvoApproxLib library [34]. In this paper, three 8-bit INT approximate multipliers are picked from EvoApproxLib with different error, area, and power characteristics reported in Table I. The error parameters reported in this table are as follows:

- MAE - Mean Absolute Error (Mean Error Magnitude)
- WCE - Worst-Case Absolute Error (Error Magnitude / Error Significance)
- MRE - Mean Relative Error (Mean Relative Error Distance)
- EP - Error Probability (Error Rate)

Power (power consumption in mW) and area (area on the chip in μm^2) are also reported as the design parameters in the last two columns of the table. To show the hardware characteristics of the output AxDNN, the Lookup Table (LUT) and Flip Flop (FF) utilization, as well as the number of required clock cycles for a one-time execution of the output AxDNN accelerator, are reported as the results based on the reports produced by Xilinx Vivado HLS tool on a Xilinx Spartan-7 FPGA with part number xc7s100-fgga676-1 and 100 MHz frequency.

TABLE I: Exact and approximate multipliers used in this paper and their parameters

Circuit name	MAE	WCE	MRE	EP	Power	Area
Exact multiplier	0.00	0.00	0.00	0.00	0.425	729.8
mul8s_1KVP	0.051	0.21	2.73	74.80	0.363	635.0
mul8s_1KV9	0.0064	0.026	0.90	68.75	0.410	685.2
mul8s_1KV8	0.0018	0.0076	0.28	50.00	0.422	711.0

TABLE II: Networks trained and quantized down to 8-bit INT for evaluation of this work

Network	Dataset	Accuracy 8-bit quantized network
3-layer MLP	MNIST	80.40%
LeNet-5	MNIST	85.80%
AlexNet	CIFAR-10	78.50%

B. Fault simulator

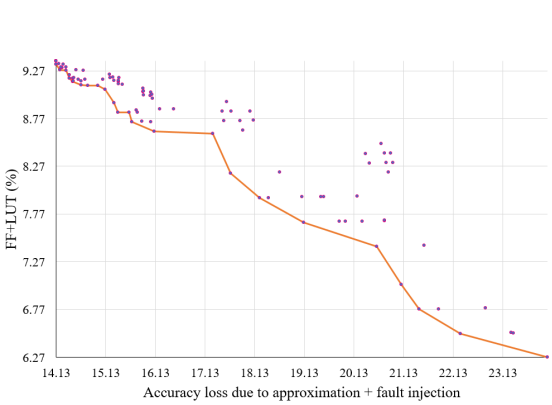
The fault simulator that is used in step 4 in Fig. 2 is implemented in the automated tool-flow of DeepAxe in a way that users can select the sufficient number of faults they need for their resiliency analysis. AxDNNs generated by step 3 in Fig. 2 are validated by means of fault injection over the test set.

Random Fault Injection. According to the adopted fault model, a random single bit-flip is injected into a random neuron in a random layer of the network, and the whole test set is fed to the network to obtain the accuracy of the network. This process is repeated several times to reach an acceptable confidence level which depends on the number of neurons and data representation bit length based on [35].

To find the required number of repetitions for the fault simulation experiments, [35] provides an equation to reach 95% confidence level and 1% error margin. However, it can pessimistically obtain a larger number, and the execution time of the iterative fault simulation experiments would be very long. Therefore, we have performed a fault simulation for each neural network to find a smaller number of experiments in a way that the difference of the average accuracy is less than 0.1% in comparison with the average accuracy of the network achieved using the statistical fault injection approach [35]. As a result, we have selected for injection 600, 800, and 1000 random single bit-flip faults for 3-layer MLP, LeNet-5, and AlexNet fault simulation, respectively.

C. Validation Results

The proposed methodology is validated on three networks, i.e. a 3-layer MLP, LeNet-5 and AlexNet, trained on two representative datasets MNIST and Cifar-10. Each network is fully quantized down to 8-bit INT as a part of the preprocessing step of the methodology. The accuracy results for the quantized networks are reported in Table II. Further, all possible combinations of approximate layers in the network are tested for



(a)

Accuracy loss (%)	#of [FF+LUT] (%)	Pareto points
24.02	6.27	mul8s_1KVP (1-1--111)
22.27	6.51	mul8s_1KVP (1-1--011)
21.44	6.77	mul8s_1KVP (1-1--010)
21.08	7.03	mul8s_1KVP (1-1--000)
20.59	7.43	mul8s_1KVP (0-1--111)
19.12	7.68	mul8s_1KVP (0-1--011)
18.23	7.93	mul8s_1KVP (0-1--001)
17.64	8.19	mul8s_1KVP (0-1--000)
17.28	8.61	mul8s_1KVP (0-0--111)
16.10	8.63	mul8s_1KV9 (0-1--111)
15.66	8.73	mul8s_1KV9 (0-1--101)
15.60	8.83	mul8s_1KV9 (0-1--100)
15.37	8.83	mul8s_1KV9 (0-1--001)
15.30	8.93	mul8s_1KV9 (0-1--000)
15.12	9.07	mul8s_1KV9 (0-0--111)
14.97	9.11	mul8s_1KVP (0-0--010)
14.77	9.11	mul8s_1KVP (0-0--001)
14.63	9.12	mul8s_1KV8 (0-1--111)

Worst error

Best error

Best area

Worst area

(b)

Fig. 3: (a) Resource utilization of the approximate implementation vs. accuracy drop when the approximate implementation is fault-simulated (b) Approximation configuration of each point on the Pareto frontier

TABLE III: The impact of approximation configuration and fault injection for MLP, LeNet-5, and AlexNet.

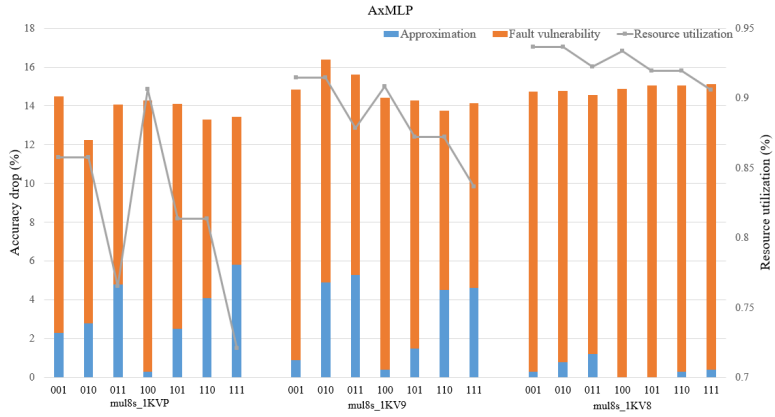
DNN dataset	Multiplier	Layer configuration	Base accuracy (%)	Accuracy drop (%) [Exact network - AxDNN]	AxDNN accuracy drop (%) [AxDNN - FI on AxDNN]	Latency (#of clk cycles)	Resource utilization (%) #of[FF + LUT] / Total #of[FF + LUT]
MLP MNIST	mul8s_1KVP	111	80.40	5.8	7.62	206644	0.72
	mul8s_1KVP	101		2.5	11.62	272180	0.81
	mul8s_1KV9	101		1.5	12.78	274740	0.87
	mul8s_1KV9	100		0.4	14.03	274740	0.90
	mul8s_1KV8	001		0.3	14.72	285010	0.95
LeNet-5 MNIST	mul8s_1KVP	1-1–111	85.80	10.6	2.82	164864	6.27
	mul8s_1KVP	1-1–011		8.8	4.67	195584	6.51
	mul8s_1KV9	0-1–111		1.7	12.70	206408	7.93
	mul8s_1KV9	0-1–101		1.0	13.66	206504	8.19
	mul8s_1KV8	0-1–111		0.7	13.23	175784	9.12
AlexNet CIFAR-10	mul8s_1KVP	0-0-11-0–011	78.50	16.0	9.12	19933514	11.75
	mul8s_1KVP	0-0-11-0–100		17.0	10.41	20324170	11.84
	mul8s_1KVP	0-0-00-0–001		2.0	11.10	20467530	12.35
	mul8s_1KV9	0-1-11-1–111		18.5	9.58	19799882	11.04
	mul8s_1KV9	0-1-11-1–110		17.5	11.80	19945802	11.93
	mul8s_1KV9	0-0-00-0–001		3.0	12.60	20470090	12.45
	mul8s_1KV8	1-1-11-1–110		6.5	10.90	20470090	12.18
	mul8s_1KV8	0-1-11-1–111		6.0	11.70	20470090	12.19
	mul8s_1KV8	0-1-11-1–110		4.5	12.00	20470090	12.21
	mul8s_1KV8	0-0-11-0–011		3.5	12.00	20470090	12.35
	mul8s_1KV8	0-0-11-0–100		2.5	12.15	20470090	12.33
	mul8s_1KV8	0-0-00-0–001		0.0	12.64	20470090	12.43

selective approximation. For each experiment, three different multipliers reported in Table I are examined separately for efficiency to substitute the original exact multipliers.

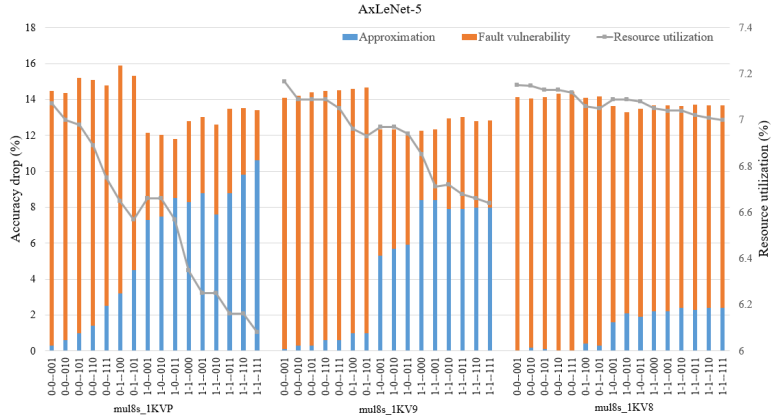
The fault injection procedure is performed for all different configurations, and the accuracy drop, due to approximation and fault injection, is profiled. Further, the HLS synthesis results of all configurations are generated, and the resource utilization in the number of FF, LUTs as well as the number of clock cycles required for processing one image for each network, are collected. A Pareto frontier for resource utilization and accuracy drop due to applying FI on different approximation configurations is plotted, and the results for LeNet-5 are reported in Fig. 3(a).

Fig. 3(b) shows the points on the Pareto frontier. The first column is the accuracy drop due to performing fault injection on that particular AxDNN configuration, the second column is resource utilization of the AxDNN in percentage, and finally, the last column is the selected approximate multiplier (AxM) and order of layers in ad-hoc (ones means that particular layer is approximated and dashes represent the non-computational layers like maxpooling). The coloured rows are some extreme and mid-range points of the Pareto chart. The same experiment is repeated for MLP and AlexNet networks, and the results for some extreme and mid-range points of their pareto charts are presented in Table III.

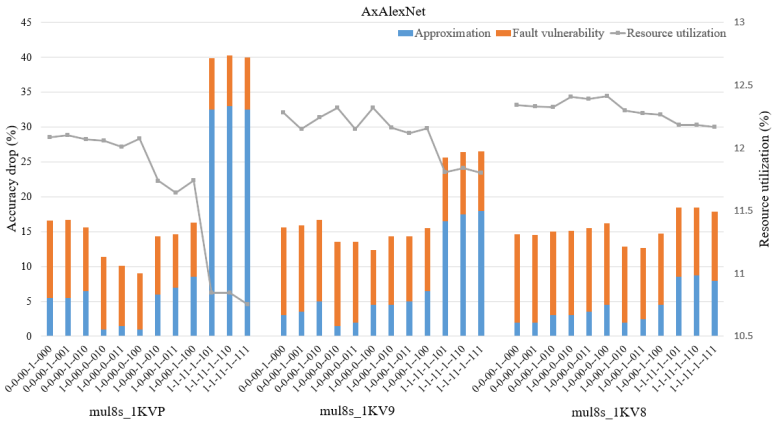
It can be observed from this table that, generally, by approximating more layers, the latency and resource utilization are less. It is also noteworthy that the fault vulnerability of the network, which can be defined as the accuracy drop of the AxDNN due to applying FI, also becomes less. Fault



(a)



(b)



(c)

Fig. 4: Reports of accuracy drop (due to approximation for different configurations), fault vulnerability, and resource utilization of (a) 3-layer MLP network, (b) LeNet-5 and (c) AlexNet

TABLE IV: Case study: the impact of full approximation on three different MLP architectures

Network MNIST dataset	Exact network accuracy (%)	Normalized resource utilization (%) [exact network]	AxM	Accuracy drop (%)	Fault vulnerability	Normalized latency	Normalized resource utilization (%)
7-layer MLP	98.80	100	mul8s_1KV8	0.2	2.45	1.00	96
			mul8s_1KV9	1.4	1.03	1.00	90
			mul8s_1KVP	0.9	1.33	0.75	76
5-layer MLP	86.30	69	mul8s_1KV8	0.0	3.33	1.00	96
			mul8s_1KV9	1.9	2.12	1.00	89
			mul8s_1KVP	3.1	3.84	0.78	76
3-layer MLP	80.40	36	mul8s_1KV8	0.4	14.14	1.00	95
			mul8s_1KV9	4.6	7.62	1.00	88
			mul8s_1KVP	5.8	9.54	0.76	74

vulnerability is opposite to fault resiliency and means the more the accuracy of an AxDNN drops due to applying FI, the more vulnerable the network is against faults. Generally, by increasing the level of approximation, the network shows better resiliency to faults. Still, there are several configurations that do not follow this trend and a tailored analysis using a framework such as DeepAxe is necessary for higher confidence.

Fig. 4 depicts the impact of different approximation units on the case-study DNNs' accuracy, resource utilization and fault vulnerability. For each network, three approximation units are chosen. For approximating the networks, the same configurations are picked to observe the impact of different AxM on the networks. Then all approximation units are applied, and the accuracy drop, fault vulnerability and resource utilization are reported. The correlation between the AxM error metrics reported in Table I, their area overhead, and the accuracy drop of the AxDNN impacted by AxMs lead us toward a conclusion that the network accuracy is generally impacted by a) the level of approximation and the configuration of the layers that are substituted by AxM; b) the error metrics of the AxM that is used as a substitution of ExC unit.

D. Approximate multipliers case-study

As a case study, three MLP networks with different architectures on the basis of a number of layers are selected. The base accuracy for each quantized network is 98.80% for the network with 7 layers, 86.30% for a network containing 5 layers and 80.40% for 3-layer MLP network. The results for full approximation of the MLP networks with each case-study approximate multiplier (AxM) are reported in Table IV. All the values in the table are normalized to the corresponding values of the ExC networks.

For the 7-layer MLP, it is shown that the multiplier mult8s_KVP is the best option for full approximation, in the sense that the accuracy of the network drops only 0.9%, and yet, latency and resource utilization of the network are better than for the other two multipliers. Therefore, based on the application of the network, if the designer can sacrifice the accuracy for 0.9%, they can gain 25% improvement in network latency and 24% improvement in resource utilization of the implemented network on FPGA.

The situation is different for the 5-layer MLP network. Based on the results of Table IV, the best multiplier can be mult8s_KV9 since the accuracy does not drop dramatically and yet, it gains a better resiliency than the other two multipliers. Similarly, in the 3-layer MLP, the best candidate for full approximation of the network is mult8s_KV9 multiplier since it shows the best resiliency with a little accuracy drop and still, provides 12% improvement in resource utilization compared to the exact design.

In summary, this case study shows the importance of exploring different AxMs for optimal implementation, i.e. not to compromise the accuracy of the network and, at the same time, to improve the network resiliency and hardware performance of the target design.

V. Conclusion

In this paper, we proposed a framework DeepAxe for design space exploration for FPGA-based implementation of DNNs by considering the trilateral impact of applying functional approximation on accuracy, reliability and hardware performance. The framework enables selective approximation of reliability-critical DNNs, providing a set of Pareto-optimal DNN implementation design space points for the target resource utilization requirements The design flow starts with a pre-trained network in Keras, uses an innovative high-level synthesis environment DeepHLS and results in a set of Pareto-optimal design space points as a guide for the designer. The framework is demonstrated on a case study of custom and state-of-the-art DNNs and datasets.

VI. Acknowledgement

This work was supported in part by the European Union through European Social Fund in the frames of the "Information and Communication Technologies (ICT) programme" ("ITA-IoIT" topic), by the Estonian Research Council grant PUT PRG1467 CRASHLESS", by Estonian-French PARROT project "EnTrustED", by the Swedish Innovation Agency VINNOVA project "SafeDeep", and Swedish Knowledge Foundation project "HERO".

References

[1] M. Taheri, "Dnn hardware reliability assessment and enhancement," *27th IEEE European Test Symposium (ETS).*, May 2022.

[2] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer, "A survey of quantization methods for efficient neural network inference," *arXiv preprint arXiv:2103.13630*, 2021.

[3] M. Riazati, M. Daneshtalab, M. Sjödin, and B. Lisper, "Autodeephls: Deep neural network high-level synthesis using fixed-point precision," in *2022 IEEE 4th International Conference on Artificial Intelligence Circuits and Systems (AICAS)*. IEEE, 2022, pp. 122–125.

[4] P. Choudhary, L. Bhargava, V. Singh, and A. K. Suhag, "Approximate computing: Evolutionary methods for functional approximation of digital circuits," *Materials Today: Proceedings*, 2022.

[5] Y. Ibrahim, H. Wang, J. Liu, J. Wei, L. Chen, P. Rech, K. Adam, and G. Guo, "Soft errors in dnn accelerators: A comprehensive review," *Microelectronics Reliability*, vol. 115, p. 113969, 2020.

[6] M. Shafique, M. Naseer, T. Theocharides, C. Kyrkou, O. Mutlu, L. Orosa, and J. Choi, "Robust machine learning systems: Challenges, current trends, perspectives, and the road ahead," *IEEE Design & Test*, vol. 37, no. 2, pp. 30–57, 2020.

[7] J. D. Booth, "Algorithm-based fault tolerance at scale," 2022.

[8] A. Bosio, P. Bernardi, A. Ruospo, and E. Sanchez, "A reliability analysis of a deep neural network," in *2019 IEEE Latin American Test Symposium (LATS)*. IEEE, 2019, pp. 1–6.

[9] N. Cavagnero, F. D. Santos, M. Ciccone, G. Averta, T. Tommasi, and P. Rech, "Fault-aware design and training to enhance dnns reliability with zero-overhead," *arXiv preprint arXiv:2205.14420*, 2022.

[10] A. Siddique, K. Basu, and K. A. Hoque, "Exploring fault-energy trade-offs in approximate dnn hardware accelerators," in *2021 22nd International Symposium on Quality Electronic Design (ISQED)*. IEEE, 2021, pp. 343–348.

[11] A. Ghaffari and Y. Savaria, "Cnn2gate: Toward designing a general framework for implementation of convolutional neural networks on fpga," *arXiv preprint arXiv:2004.04641*, 2020.

[12] M. Riazati, M. Daneshtalab, M. Sjödin, and B. Lisper, "Deephls: A complete toolchain for automatic synthesis of deep neural networks to fpga," in *2020 27th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, 2020, pp. 1–4.

[13] S. I. Venieris, A. Kouris, and C.-S. Bouganis, "Toolflows for mapping convolutional neural networks on fpgas: A survey and future directions," *arXiv preprint arXiv:1803.05900*, 2018.

[14] K. Guo, S. Zeng, J. Yu, Y. Wang, and H. Yang, "[dl] a survey of fpga-based neural network inference accelerators," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 12, no. 1, pp. 1–26, 2019.

[15] K. Abdelouahab, M. Pelcat, J. Serot, and F. Berry, "Accelerating cnn inference on fpgas: A survey," *arXiv preprint arXiv:1806.01683*, 2018.

[16] R. S. Molina, V. Gil-Costa, M. L. Crespo, and G. Ramponi, "High-level synthesis hardware design for fpga-based accelerators: Models, methodologies, and frameworks," *IEEE Access*, vol. 10, pp. 90 429–90 455, 2022.

[17] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre, and K. Vissers, "Finn: A framework for fast, scalable binarized neural network inference," in *Proceedings of the 2017 ACM/SIGDA international symposium on field-programmable gate arrays*, 2017, pp. 65–74.

[18] S. I. Venieris and C.-S. Bouganis, "fpgaconvnet: Mapping regular and irregular convolutional neural networks on fpgas," *IEEE transactions on neural networks and learning systems*, vol. 30, no. 2, pp. 326–342, 2018.

[19] A. Ghaffari and Y. Savaria, "Cnn2gate: Toward designing a general framework for implementation of convolutional neural networks on fpga," *arXiv preprint arXiv:2004.04641*, 2020.

[20] P. G. Mousouliotis and L. P. Petrou, "Cnn-grinder: from algorithmic to high-level synthesis descriptions of cnns for low-end-low-cost fpga socs," *Microprocessors and Microsystems*, vol. 73, p. 102990, 2020.

[21] J. E. Stone, D. Gohara, and G. Shi, "Opencl: A parallel programming standard for heterogeneous computing systems," *Computing in science & engineering*, vol. 12, no. 3, p. 66, 2010.

[22] S. I. Venieris and C.-S. Bouganis, "fpgaconvnet: A framework for mapping convolutional neural networks on fpgas," in *2016 IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2016, pp. 40–47.

[23] K. Guo, L. Sui, J. Qiu, S. Yao, S. Han, Y. Wang, and H. Yang, "Angel-eye: A complete design flow for mapping cnn onto customized hardware," in *2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 2016, pp. 24–29.

[24] H. Sharma, J. Park, D. Mahajan, E. Amaro, J. K. Kim, C. Shao, A. Mishra, and H. Esmaeilzadeh, "From high-level deep neural models to fpgas," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2016, pp. 1–12.

[25] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th annual international symposium on computer architecture*, 2017, pp. 1–12.

[26] S. S. Sarwar, S. Venkataramani, A. Ankit, A. Raghunathan, and K. Roy, "Energy-efficient neural computing with approximate multipliers," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 14, no. 2, pp. 1–23, 2018.

[27] S. Hong, P. Frigo, Y. Kaya, C. Giuffrida, and T. Dumitraş, "Terminal brain damage: Exposing the graceless degradation in deep neural networks under hardware fault attacks," in *28th USENIX Security Symposium (USENIX Security 19)*, 2019, pp. 497–514.

[28] S. Kundu, A. Soyyiğit, K. A. Hoque, and K. Basu, "High-level modeling of manufacturing faults in deep neural network accelerators," in *2020 IEEE 26th International Symposium on On-Line Testing and Robust System Design (IOLTS)*. IEEE, 2020, pp. 1–4.

[29] G. Li, S. K. S. Hari, M. Sullivan, T. Tsai, K. Pattabiraman, J. Emer, and S. W. Keckler, "Understanding error propagation in deep learning neural network (dnn) accelerators and applications," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2017, pp. 1–12.

[30] J. J. Zhang, T. Gu, K. Basu, and S. Garg, "Analyzing and mitigating the impact of permanent faults on a systolic array based neural network accelerator," in *2018 IEEE 36th VLSI Test Symposium (VTS)*. IEEE, 2018, pp. 1–6.

[31] M. H. Ahmadilivani, M. Taheri, J. Raik, M. Daneshtalab, and M. Jenihhin, "Deepvigor: Vulnerability value ranges and factors for dnns reliability assessment," in *28th IEEE European Test Symposium*. In press, 2023.

[32] M. Bushnell and V. Agrawal, *Essentials of electronic testing for digital, memory and mixed-signal VLSI circuits*. Springer Science & Business Media, 2004, vol. 17.

[33] R. David, J. Duke, A. Jain, V. Janapa Reddi, N. Jeffries, J. Li, N. Kreeger, I. Nappier, M. Natraj, T. Wang *et al.*, "Tensorflow lite micro: Embedded machine learning for tinyml systems," *Proceedings of Machine Learning and Systems*, vol. 3, pp. 800–811, 2021.

[34] V. Mrazek, R. Hrbacek, Z. Vasicek, and L. Sekanina, "Evoapprox8b: Library of approximate adders and multipliers for circuit design and benchmarking of approximation methods," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, March 2017, pp. 258–261.

[35] R. Leveugle, A. Calvez, P. Maistri, and P. Vanhauwaert, "Statistical fault injection: Quantified error and confidence," in *2009 Design, Automation & Test in Europe Conference & Exhibition*. IEEE, 2009, pp. 502–506.

Appendix 7

VII

S. Nazari, M. Taheri, A. Azarpeyvand, M. Jenihhin, M. Daneshtalab, et. al. "FORTUNE: A Negative Memory Overhead Hardware-Agnostic Fault TOleRance TechniqUe in DN Ns," In 2024 33rd IEEE Asian Test Symposium (ATS 2024).

The 33rd Asian Test Symposium (ATS 2024)

FORTUNE: A Negative Memory Overhead Hardware-Agnostic Fault TOleRance TechniqUe in DNNs

Samira Nazari[1*], Mahdi Taheri[2,3*], Ali Azarpeyvand[1,2], Mohsen Afsharchi[1],
Tara Ghasempouri[2], Christian Herglotz[3], Masoud Daneshtalab[2,4], and Maksim Jenihhin[2]

[1]University of Zanjan, Zanjan, Iran
[2]Tallinn University of Technology, Tallinn, Estonia
[3]Brandenburgische Technische Universität Cottbus, Cottbus, Germany
[4]Mälardalen University, Västerås, Sweden

***Abstract*—This paper presents FORTUNE, a hardware-agnostic fault tolerance technique for DNNs that leverages quantization to enhance reliability without significant performance overhead. Unlike conventional methods like Triple Modular Redundancy (TMR), which are computationally expensive, the proposed approach uses memory savings from quantization to protect the critical Most Significant Bit, improving fault tolerance in Deep Neural Networks (DNNs). Memory utilization has been reduced by 37.5% across all networks, with vulnerability in AlexNet reduced by 56% compared to the 8-bit version and 84% compared to the unprotected 3-bit version. These improvements come with only a minor increase in execution time of less than 3%. Using AlexNet as an example demonstrates how our approach effectively enhances memory utilization and resilience while causing only a minimal increase in execution time.**

***Index Terms*—deep neural networks, parallel processing, memory overhead, reliability, DNN accelerator**

I. Introduction

As real-time data processing and AI demand grow in embedded systems, Quantized Deep Neural Networks (QNNs) have become vital for deploying models efficiently in resource-constrained environments [1], spanning applications from image classification to safety-critical applications like autonomous driving [2], [3]. QNNs perform complex computations with reduced precision, minimizing computational and memory footprints, and achieving significant energy savings, crucial for energy-constrained platforms [4].

However, quantization introduces challenges in neural network accuracy, especially in critical applications like autonomous systems or medical diagnostics, where precision is non-negotiable [5], [6]. Post-training quantization (PTQ) methods address these challenges, preserving model integrity while reducing bit-widths to enhance efficiency [7]. Yet, QNNs' reliance on extensive memory resources makes them vulnerable to faults, particularly as transistors miniaturize.

979-8-3315-2916-1/24/$31.00 ©2024 IEEE
*Authors contributed equally

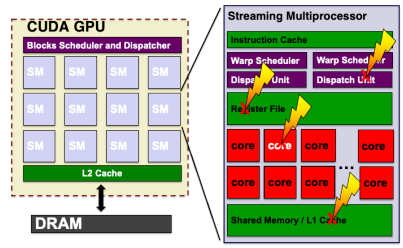


Fig. 1: Fault locations in critical points of a parallel architecture, e.g. the GPU-based [18]

Ensuring fault tolerance is essential, as even minor errors can significantly reduce accuracy [8].

Figure 1 highlights fault locations in critical GPU architecture points. Faults can arise from temperature fluctuations, radiation, aging circuits, and electromagnetic interference [9]. Enhancing fault tolerance in QNNs is thus crucial for safe deployment in safety-critical systems [3], [10], [11].

Traditional fault-tolerant techniques like Triple Modular Redundancy (TMR) introduce significant computational overhead [3]. Selective hardening approaches focus on protecting parameters or neurons with greater impact on the network's output [12]. These methods, however, are mainly applicable to FPGA and ASIC platforms where hardware modifications are possible [13], [14]. For general-purpose CPUs, GPUs, or fixed accelerators, hardware modifications are often infeasible [15].

Other methods, such as Error Correction Codes (ECC) [16], introduce significant memory and computational overhead. Activation restriction techniques mitigate error propagation but are ineffective at high error rates [17].

To address the extensive memory overhead and performance degradation challenges, we present FORTUNE, a model-level hardware agnostic methodology that explores different quantization levels of a DNN model, focusing on reliability, accuracy, memory overhead, and performance aspects. The

proposed methodology introduces a novel fault tolerance technique that uses memory savings from quantization to the Most Significant Bit (MSB) within the same memory elements, maintaining QNN reliability even in the presence of memory faults. A comprehensive GPU-based framework implements this methodology. Specifically, the proposed fault tolerance methodology integrates into the framework's iterative optimization loop, allowing users to define thresholds that balance model accuracy, reliability, and performance. The extensive experiments show that, without this protection, QNNs suffer significant accuracy degradation in fault-prone environments. The results underscore the importance of the proposed technique to minimize memory utilization and optimize protected QNN execution time on DNN accelerators. In this work, without loss of generality, we reported the results for GPU as an example.

The key contributions of this paper are:

- **Negative Overhead Fault Tolerance Technique**: Proposed a protection technique leveraging quantization to triplicate MSB, ensuring robustness against faults.
- **Design Space Exploration Framework**: Developed an open-source framework to assess the impact of quantization on QNN reliability, accuracy, memory utilization, and executing time for design space exploration. https://github.com/nilay1400/DNN-Quantization
- **Introduction of P_{drop} and Reliability-Aware Performance (RAP) metrics**: Introduced P_{drop}, the probability of accuracy drop over a device's lifetime with various BERs, and RAP, a metric for evaluating trade-offs in fault-prone and resource-constrained environments.
- **Validation**: Evaluated the proposed technique and framework on state-of-the-art DNN benchmarks.

The remainder of this paper is structured as follows: Section II presents the proposed methodology. Section III discusses experimental results and the impact on resilience, memory utilization and execution time. Finally, Section IV concludes the paper.

II. Proposed Methodology

Figure 2 illustrates the methodology for quantizing a DNN model, evaluating its reliability in fault-prone environments, and applying the proposed protection. The steps of this methodology are explained by the algorithm 1 and also in detail in this section. The goal is to achieve a quantized model that not only meets predefined accuracy and reliability thresholds but also provides insights into memory utilization and execution time. The algorithm uses a trained FP32 DNN model as input, along with three key parameters: an accuracy threshold a, a reliability threshold b, and a quantization range $[m, n]$ (Algorithm 1 - line 4-6). The accuracy threshold a represents the minimum acceptable accuracy of the model after quantization in the fault-free model, while the reliability threshold b specifies the maximum permissible drop in accuracy due to potential faults. The quantization range $[m, n]$ defines the range of bit widths to be explored during the quantization process.

Algorithm 1 FORTUNE Algorithm

1: **Input:** Trained FP32 DNN model, accuracy threshold a, reliability threshold b, quantization range $[m, n]$
2: **Output:** High-performance Reliable QNN
3: Load the trained DNN
4: SET accuracy_threshold TO a
5: SET reliability_threshold TO b
6: SET quantization range TO $[m, n]$
7: $bit\_width = \frac{n+m}{2}$
8: process = True
9: **while** process **do**
10: quantized_weights← QuantizeWeights(model.weights, bit_width)
11: golden_accuracy←EvaluateAccuracy(model, quantized_weights)
12: **if** $accuracy > a$ AND $accuracy\_drop < b$ **then**
13: Inject faults to weights of the model with different BERs
14: accuracy ← EvaluateAccuracy(model, quantized_weights)
15: accuracy_drop ← golden_accuracy − accuracy
16: Report Memory Utilization
17: Report Execution Time
18: $bit\_width \leftarrow bit\_width - \frac{n-bit\_width}{2}$
19: **else**
20: $bit\_width \leftarrow bit\_width + \frac{n-bit\_width}{2}$
21: **end if**
22: **if** $bit\_width > n$ **then**
23: process = False
24: **end if**
25: **end while**

Step 1: Quantization. The algorithm iterates over each bit width within the specified quantization range. To perform a binary search, the initial bit width considered is the midpoint of the quantization range. Based on the results obtained at this midpoint, the next bit width for weights is selected from either the upper or lower half of the range, depending on whether the desired accuracy and reliability criteria are met (Algorithm 1 - lines 7, 18 and 20). This process is repeated iteratively, narrowing the search range until the optimal bit width is identified. For each selected bit width, the model weights are quantized through linear quantization (Algorithm 1 - line 10). Linear quantization is a widely used technique in model compression, particularly in the context of QNNs. It reduces the precision of weights by mapping a large range of values (typically 32-bit floating-point) to a smaller, fixed range represented by fewer bits. This process involves two main steps: scaling and rounding. The first step in linear quantization is to define the range $[x_{min}, x_{max}]$ within which all the values of the tensor will be mapped.

$$x_{min} = min(x) \tag{1}$$

$$x_{max} = max(x) \tag{2}$$

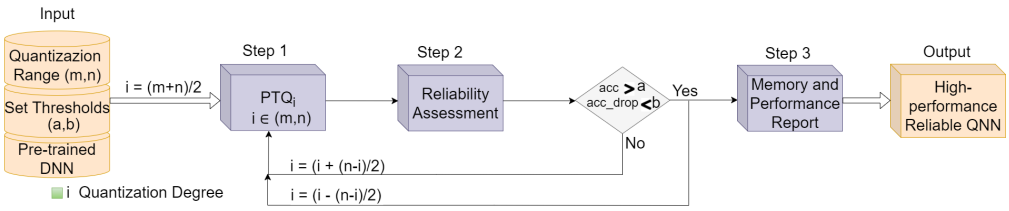


Fig. 2: Proposed methodology for quantizing a DNN model, evaluating its reliability and applying the protection.

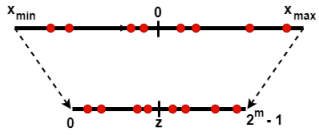


Fig. 3: Affine linear quantization

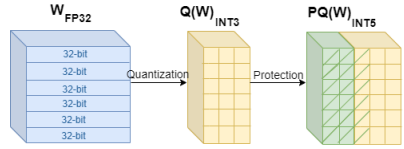


Fig. 4: An example of protected 3-bit weights

Next, the scaling factor s is calculated. This factor determines how the original floating-point values are scaled down to fit within the quantized range. For a given bit width m, the quantized range is $[0, 2^m - 1]$. The scaling factor s is computed as:

$$s = \frac{x_{max} - x_{min}}{2^m - 1} \tag{3}$$

As Figure 3 indicates, the tensor values are then quantized by mapping each value x_i to an integer value q_i within the range $[0, 2^m - 1]$ using the scaling factor s and a zero-point z:

$$z = -\frac{x_{min}}{s} \tag{4}$$

$$q_i = round(\frac{x_i}{s} + z) \tag{5}$$

Here, $round(.)$ denotes rounding to the nearest integer. The result is an integer value q_i that lies within the quantized range. Therefore, weights of the model are converted into unsigned m-bit integers through affine linear quantization.

Then, the accuracy of the model with quantized weights, referred to as the "golden accuracy," is evaluated. This accuracy serves as a baseline for further reliability testing (Algorithm 1 - line 11). If the golden accuracy falls below the predefined threshold a, the algorithm skips further evaluation for this bit width and proceeds to the next (Algorithm 1 - lines 19, 20).

Step 2: Reliability Evaluation and Enhancement. Then, the algorithm proceeds to evaluate its reliability under fault conditions. Faults are incrementally injected into the quantized model's weights, simulating different Bit Error Rates (BERs). After each fault injection, the model's accuracy is re-evaluated, and the accuracy drop is calculated as the difference between the golden accuracy and the current accuracy (Algorithm 1 - lines 13-15).

In the proposed approach, only the MSB bit is protected by replicating it in two redundant bits. During inference, these redundant bits, along with the protected bit, undergo a majority voting process to determine the final value of the protected bit.

TABLE I: Benchmark NNs base accuracies (%)

Type	VGG11	ResNet18	AlexNet	Inception
8-bit	92.41	93.06	95.32	93.70
5-bit	92.19	92.81	95.46	93.24
4-bit	92.18	92.66	94.91	92.03
3-bit	89.83	90.84	93.26	80.71

This method ensures that even with potential faults, the most critical bit remains reliable.

Figure4 illustrates an example of 3-bit weights with a protected bit. In this example, the FP32 weights are first converted into unsigned 3-bit integer values (shown in orange), and the MSB is replicated into two redundant bits (shown in green). Similarly, in 5-bit weights with one protected bit, two redundant bits are added. More generally, in i-bit quantization with one protection bits, two redundant bits are added, and one comparison is performed during inference.

Step 3: Memory Utilization and Execution Time. In this step, the algorithm computes the memory utilization and execution time associated with the given bit width (Algorithm 1 - lines 16, 17). These values provide insights into the trade-offs between quantization, reliability, and resource utilization. The algorithm repeats the above steps for all bit widths within the specified range, ultimately outputting a set of reliable and quantized networks. For each network, detailed reports on execution time and memory utilization are generated, allowing for an informed selection of the optimal quantization configuration.

P_{drop}* and *RAP

Accuracy drop is evaluated independently of any physical effects that faults might have on memory. To account for these effects on accuracy drop, we define P_{drop} as the probability of experiencing the accuracy drop during the device's lifetime:

$$P_{drop} = N^2 \times W^2 \times T/t \times P_{single} \times BER \times acc\_drop \tag{6}$$

where N is the number of parameters, W is their bit width, T is device life time, t is test time interval, P_{single} is the

TABLE II: Memory Utilization, Execution Time, Vulnerability (accuracy drop due to fault injection) and P_{drop} in DNNs.

Model	Type	Memory Utilization	Executuin Time (%)	Vulnerability (%) {BER}				P_{drop} {BER}			
				1.00E-5	3.00E-5	1.00E-4	3.00E-4	1.00E-5	3.00E-5	1.00E-4	3.00E-4
VGG11	8-bit	225,064,448	100	5.62	22.30	62.40	74.52	50.21E-3	150.63E-3	502.10E-3	1506.30E-3
	P-5-bit	196,931,392	141.72	1.11	5.23	28.33	68.73	35.45E-3	106.35E-3	354.50E-3	1063.52E-3
	P-4-bit	168,798,336	141.72	1.45	6.01	28.19	67.99	25.76E-3	77.30E-3	257.66E-3	773.00E-3
	P-3-bit	140,665,280	141.72	1.49	5.35	34.20	67.13	17.66E-3	53.00E-3	176.68E-3	530.05E-3
ResNet18	8-bit	66,991,616	100	0.18	1.85	14.64	32.29	1.92E-3	5.78E-3	19.27E-3	57.82E-3
	P-5-bit	58,617,664	119.54	0.16	0.81	3.54	18.77	0.85E-3	2.57E-3	8.58E-3	25.74E-3
	P-4-bit	50,243,712	119.54	0.24	0.66	2.63	18.34	0.61E-3	1.84E-3	6.15E-3	18.47E-3
	P-3-bit	41,869,760	119.54	0.25	0.79	3.74	21.89	0.51E-3	1.53E-3	5.10E-3	15.31E-3
AlexNet	8-bit	466,316,032	100	0.00	0.49	3.14	30.71	88.82E-3	266.47E-3	888.23E-3	2664.70E-3
	P-5-bit	408,026,528	102.94	0.14	0.04	0.48	4.73	10.47E-3	31.43E-3	104.78E-3	314.46E-3
	P-4-bit	349,737,024	102.94	0.19	0.10	0.52	5.30	8.62E-3	25.87E-3	86.24E-3	258.72E-3
	P-3-bit	291,447,520	102.94	0.27	0.58	2.62	13.58	15.34E-3	46.03E-3	153.44E-3	460.33E-3
Inception	8-bit	173,011,456	100	0.09	0.16	0.45	1.43	0.57E-3	1.71E-3	5.71E-3	17.15E-3
	P-5-bit	151,234,496	291.73	0.00	0.02	0.02	0.19	0.06E-3	0.18E-3	0.60E-3	1.80E-3
	P-4-bit	129,758,592	291.73	0.01	0.05	0.08	0.14	0.03E-3	0.09E-3	0.33E-3	0.99E-3

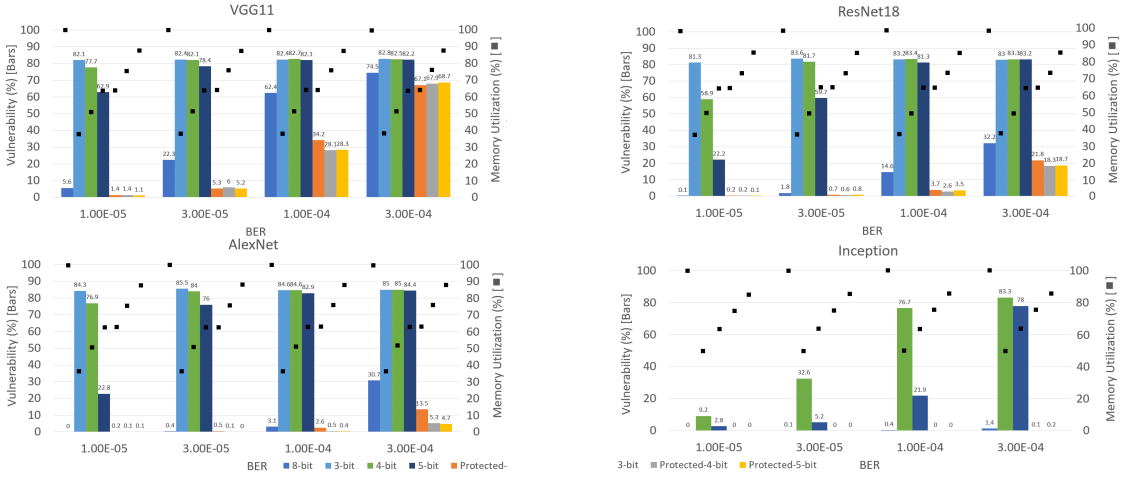


Fig. 5: Vulnerability (accuracy drop due to fault injection) and memory utilization trade-offs in different benchmarks: VGG-11 (CIFAR), ResNet-18 (CIFAR), AlexNet (FashionMNIST) and Inception(CIFAR).

probability of one bit flip during t and acc_drop is the reported accuracy drop in the BER. This metric is based on the probability of a one-bit flip stated in [19]. As the definition suggests, the more resilient the networks are, the smaller the value of P_{drop} becomes.

To account for performance along with accuracy drop and memory footprint, we define the Reliability-Aware Performance (RAP) metric as:

$$RAP = acc\_drop \times mem\_ovh \times perf\_ovh \qquad (7)$$

where acc_drop represents the accuracy drop, $perf\_ovh$ refers to the execution time overhead, and mem_ovh denotes the memory utilization overhead. Smaller RAP values indicate more reliable networks with lower memory and performance overhead.

III. Experimental Results

A. Experimental Setup

To evaluate the reliability and performance of the proposed method, we study four different neural networks. AlexNet is trained on the Fashion MNIST dataset, while VGG11, ResNet-18, and Inception are trained on the CIFAR-10 dataset. Quantization and reliability evaluations are conducted for all networks, resulting in the introduction of different reliable versions for each model. Then, the memory utilization and execution time associated with FORTUNE is assessed. To further quantify the effectiveness of the results, P_{drop} and RAP values are reported for each network to highlight the trade-offs involved.

B. Quantization and reliability evaluation

To facilitate comparison across different networks, the results for four different bit widths are presented in this section.

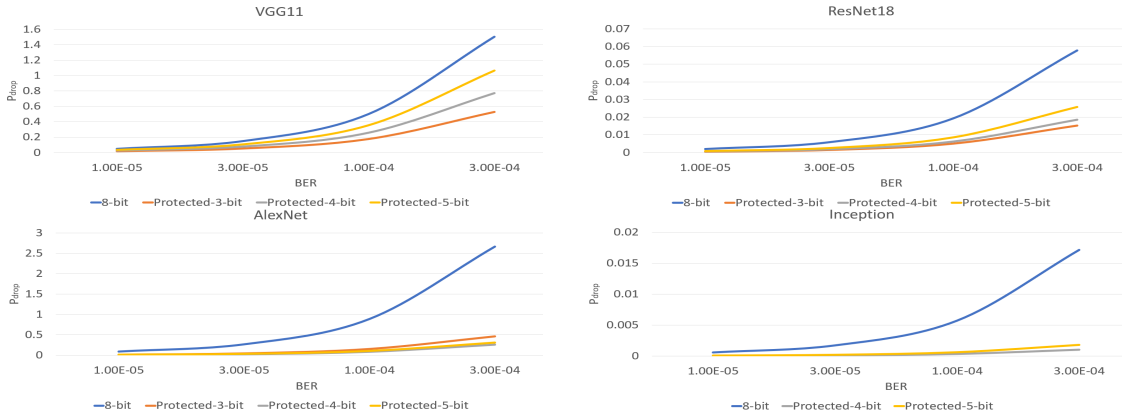


Fig. 6: P_{drop} in VGG-11 (CIFAR), ResNet-18 (CIFAR), AlexNet (FashionMNIST) and Inception(CIFAR).

The unprotected 8-bit version is used as the baseline network. As previously mentioned, the weights are quantized using affine linear quantization, and no retraining is performed. The accuracy of the different quantized networks is presented in Table I. Notably, the accuracy of Inception in its 3-bit version is 80.71%, representing an accuracy reduction of over 10%.

To evaluate reliability, a random fault injection is conducted across all weights in the DNNs under study. The number of injected faults is determined using a BER ranging from 10^{-5} to 3×10^{-4}, covering a comprehensive range of potential errors [8]. Fault injection is repeated several times to reach an acceptable confidence level, following the approach in [20]. This reference provides an equation to reach a 95% confidence level and 1% error margin. For each bit width and BER, the resulting drop in accuracy (with respect to the corresponding fault-free model) is reported in Table II as Vulnerability of the networks. Although the protected versions of AlexNet and ResNet-18 do not exhibit significant differences in Vulnerability values at lower BERs, a considerable difference becomes apparent at higher BERs. Conversely, VGG11 and Inception experience less vulnerability across all protected versions and BER levels. Additionally, the unprotected versions of the quantized networks show worst vulnerabilities, as illustrated in Figure 5.

C. Reliability, memory and performance trade-off

Memory utilization is defined as the combination of the number of parameters and their respective bit widths, while execution time refers to the execution time of each network under study. Memory utilizationn and execution time are reported in Table II, where execution time is normalized with respect to the unprotected 8-bit model. Figure 5 illustrates the trade-off between vulnerability and memory utilization for all protected and unprotected versions of the quantized networks. As the BER increases, all protected models exhibit lower vulnerability compared to the unprotected 8-bit model, while also utilizing less memory. For example, while the unprotected 5-bit model has the same memory utilization as the protected 3-bit model, the protected model demonstrates significantly lower vulnerability. To clarify, all quantized models (5-bit, 4-bit, and 3-bit) are significantly more vulnerable than their protected counterparts. Moreover, the protected versions still maintain lower memory utilization compared to the base 8-bit model.

P_{drop} values for the DNNs under study are reported in Table II and Figure 6. All final networks are considered more resilient when evaluated based on P_{drop}. This indicates that, when factoring in both memory footprint and vulnerability, all protected quantized networks exhibit greater resilience throughout their lifetime.

RAP values are reported in Figure 7. Except for VGG11 at the highest BER, all other networks demonstrate smaller RAP values in protected quantized versions, indicating better trade-off between reliability, memory utilization, and execution time. Smaller RAP values indicate that the networks introduced by FORTUNE are more resilient, while also requiring less memory and execution time compared to the base 8-bit models.

IV. Conclusion

This paper introduces FORTUNE, a novel framework designed to enhance fault tolerance in DNNs through quantization, offering a balanced trade-off between reliability, memory usage, and execution time. By leveraging memory savings to protect the critical Most Significant Bit, FORTUNE improves fault tolerance without the high computational costs of conventional methods like TMR. As examples, we demonstrated memory reductions of 37.5% across networks, with vulnerability in AlexNet reduced by 56% compared to the 8-bit model and 84% compared to the unprotected 3-bit model. These improvements were achieved with less than a 3% increase in execution time. FORTUNE proposes a flexible framework that allows for the identification of the most suitable network configuration, optimizing the trade-off between reliability, memory efficiency, and execution time based on specific application requirements.

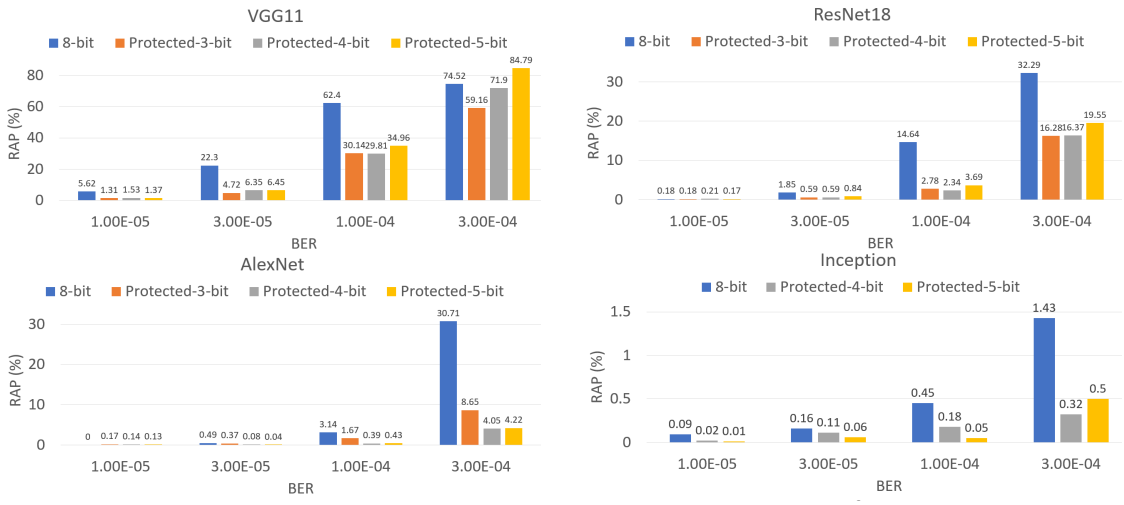


Fig. 7: RAP in VGG-11 (CIFAR), ResNet-18 (CIFAR), AlexNet (FashionMNIST) and Inception(CIFAR) .

V. Acknowledgement

This work was supported in part by the Estonian Research Council grant PUT PRG1467 "CRASHLESS", EU Grant Project 101160182 "TAICHIP" and by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – Project-ID "458578717".

References

[1] B. Rokh, A. Azarpeyvand, and A. Khanteymoori, "A comprehensive survey on model quantization for deep neural networks in image classification," *ACM Transactions on Intelligent Systems and Technology*, vol. 14, no. 6, pp. 1–50, 2023.

[2] M. Taheri, "Dnn hardware reliability assessment and enhancement," *27th IEEE European Test Symposium (ETS).*, May 2022.

[3] M. H. Ahmadilivani, M. Taheri, J. Raik, M. Daneshtalab, and M. Jenihhin, "A systematic literature review on hardware reliability assessment methods for deep neural networks," *ACM Computing Surveys*, vol. 56, no. 6, pp. 1–39, 2024.

[4] M. Taheri, M. Riazati, M. H. Ahmadilivani, M. Jenihhin, M. Daneshtalab, J. Raik, M. Sjödin, and B. Lisper, "Deepaxe: A framework for exploration of approximation and reliability trade-offs in dnn accelerators," in *2023 24th International Symposium on Quality Electronic Design (ISQED)*. IEEE, 2023, pp. 1–8.

[5] M. Taheri, N. Cherezova, M. S. Ansari, M. Jenihhin, A. Mahani, M. Daneshtalab, and J. Raik, "Exploration of activation fault reliability in quantized systolic array-based dnn accelerators," in *2024 25th International Symposium on Quality Electronic Design (ISQED)*. IEEE, 2024, pp. 1–8.

[6] M. H. Ahmadilivani, M. Barbareschi, S. Barone, A. Bosio, M. Daneshtalab, S. D. Torca, G. Gavarini, M. Jenihhin, J. Raik, A. Ruospo, E. Sanchez, and M. Taheri, "Special session: Approximation and fault resiliency of dnn accelerators," in *2023 IEEE 41st VLSI Test Symposium (VTS)*, 2023, pp. 1–10.

[7] Z. Yuan, J. Liu, J. Wu, D. Yang, Q. Wu, G. Sun, W. Liu, X. Wang, and B. Wu, "Benchmarking the reliability of post-training quantization: a particular focus on worst-case performance," *arXiv preprint arXiv:2303.13003*, 2023.

[8] M. H. Ahmadilivani *et al.*, "Enhancing fault resilience of qnns by selective neuron splitting," in *2023 IEEE 5th AICAS*, 2023, pp. 1–5.

[9] M. Taheri, M. H. Ahmadilivani, M. Jenihhin, M. Daneshtalab, and J. Raik, "Appraiser: Dnn fault resilience analysis employing approximation errors," in *2023 26th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS)*, 2023, pp. 124–127.

[10] M. H. Ahmadilivani *et al.*, "Special session: Reliability assessment recipes for dnn accelerators," in *2024 VTS*. IEEE, 2024, pp. 1–6.

[11] M. Taheri, M. Daneshtalab, J. Raik, M. Jenihhin, S. Pappalardo, P. Jimenez, B. Deveautour, and A. Bosio, "Saffira: a framework for assessing the reliability of systolic-array-based dnn accelerators," in *2024 27th International Symposium on Design & Diagnostics of Electronic Circuits & Systems (DDECS)*. IEEE, 2024, pp. 19–24.

[12] A. Ruospo and E. Sanchez, "On the reliability assessment of artificial neural networks running on ai-oriented mpsocs," *Applied Sciences*, vol. 11, no. 14, p. 6455, 2021.

[13] M. Taheri, N. Cherezova, S. Nazari, A. Rafiq, A. Azarpeyvand, T. Ghasempouri, M. Daneshtalab, J. Raik, and M. Jenihhin, "Adam: Adaptive fault-tolerant approximate multiplier for edge dnn accelerators," in *2024 IEEE European Test Symposium (ETS)*, 2024, pp. 1–4.

[14] M. Taheri, N. Cherezova, S. Nazari, A. Azarpeyvand, T. Ghasempouri, M. Daneshtalab, J. Raik, and M. Jenihhin, "Adam: Adaptive approximate multiplier for fault tolerance in dnn accelerators," *Authorea Preprints*, 2024.

[15] M. Nourazar, V. Rashtchi, A. Azarpeyvand, and F. Merrikh-Bayat, "Code acceleration using memristor-based approximate matrix multiplier: Application to convolutional neural networks," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 12, pp. 2684–2695, 2018.

[16] S. Lee and J. Yang, "Value-aware parity insertion ecc for fault-tolerant deep neural network," in *2022 DATE*, 2022, pp. 724–729.

[17] B. Ghavami *et al.*, "Fitact: Error resilient deep neural networks via fine-grained post-trainable activation functions," in *2022 DATE*, 2022, pp. 1239–1244.

[18] R. G. Alía, A. Coronetti, K. Bilko, M. Cecchetto, G. Datzmann, S. Fiore, and S. Girard, "Heavy ion energy deposition and see intercomparison within the radnext irradiation facility network," *IEEE Transactions on Nuclear Science*, vol. 70, no. 8, pp. 1596–1605, 2023.

[19] Z. Yan, Y. Shi, W. Liao, M. Hashimoto, X. Zhou, and C. Zhuo, "When single event upset meets deep neural networks: Observations, explorations, and remedies," in *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2020, pp. 163–168.

[20] R. Leveugle, A. Calvez, P. Maistri, and P. Vanhauwaert, "Statistical fault injection: Quantified error and confidence," in *2009 Design, Automation & Test in Europe Conference & Exhibition*. IEEE, 2009, pp. 502–506.

Appendix 8

VIII

M. H. Ahmadilivani, M. Taheri, J. Raik, M. Daneshtalab, and M. Jenihhin, "Deep-Vigor: Vulnerability value ranges and factors for DNNs' reliability assessment," in 2023 IEEE European Test Symposium (ETS), pp. 1–6, 2023.

DeepVigor: VulnerabIlity Value RanGes and FactORs for DNNs' Reliability Assessment

Mohammad Hasan Ahmadilivani[1], Mahdi Taheri[1], Jaan Raik[1], Masoud Daneshtalab[1,2], and Maksim Jenihhin[1]

[1]Tallinn University of Technology, Tallinn, Estonia
[2]Mälardalen University, Västerås, Sweden
[1]{mohammad.ahmadilivani, mahdi.taheri, jaan.raik, maksim.jenihhin}@taltech.ee
[2]masoud.daneshtalab@mdu.se

2023 IEEE European Test Symposium (ETS) | 979-8-3503-3634-4/23/$31.00 ©2023 IEEE | DOI: 10.1109/ETS56758.2023.10174133

***Abstract*—Deep Neural Networks (DNNs) and their accelerators are being deployed ever more frequently in safety-critical applications leading to increasing reliability concerns. A traditional and accurate method for assessing DNNs' reliability has been resorting to fault injection, which, however, suffers from prohibitive time complexity. While analytical and hybrid fault injection-/analytical-based methods have been proposed, they are either inaccurate or specific to particular accelerator architectures.**

In this work, we propose a novel accurate, fine-grain, metric-oriented, and accelerator-agnostic method called DeepVigor that provides vulnerability value ranges for DNN neurons' outputs. An outcome of DeepVigor is an analytical model representing vulnerable and non-vulnerable ranges for each neuron that can be exploited to develop different techniques for improving DNNs' reliability. Moreover, DeepVigor provides reliability assessment metrics based on vulnerability factors for bits, neurons, and layers using the vulnerability ranges.

The proposed method is not only faster than fault injection but also provides extensive and accurate information about the reliability of DNNs, independent from the accelerator. The experimental evaluations in the paper indicate that the proposed vulnerability ranges are 99.9% to 100% accurate even when evaluated on previously unseen test data. Also, it is shown that the obtained vulnerability factors represent the criticality of bits, neurons, and layers proficiently. DeepVigor is implemented in the PyTorch framework and validated on complex DNN benchmarks.

I. Introduction

Deep Neural Networks (DNNs) have recently emerged to be exploited in a wide range of applications. DNN accelerators have also penetrated into safety-critical applications e.g., autonomous vehicles [1], [2]. Therefore, several concerns are raised regarding developing and utilizing DNN accelerators in the realm of safety-critical applications, one of them being the reliability.

Reliability of DNNs concerns their accelerators' ability to perform correctly in the presence of faults [3] originating from either the environment (e.g., soft errors, electromagnetic effects, temperature variations) or inside of the chip (e.g., manufacturing defects, process variations, aging effects) [1], [4]. As shown in Fig. 1, faults may occur in different locations of accelerators either in memory or logic components and they influence the parameters (e.g., weights and bias) and intermediate results (layers' activations) of neural networks that can decrease their accuracy drastically [5], [6]. By technology miniaturization, the effect of Single Event Transient (SET) and Single Event Upset (SEU) faults in devices is increasing thereby jeopardizing the reliability of modern digital systems [7].

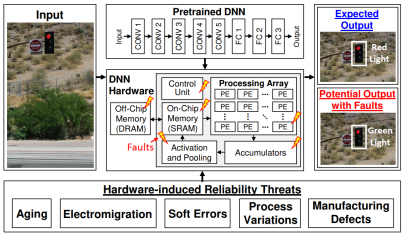


Fig. 1: Hardware reliability threats in DNN accelerators and their impact on the output [1].

Recently, several works have been published on the assessment and improvement of the reliability of a variety of DNNs as well as on different levels of system hierarchy [3], [4], [8]. Reliability assessment is the process of modeling the target DNN accelerator and measuring its reliability with respect to the corresponding quantitative evaluation metrics. Reliability assessment is the underlying procedure for improving reliability since it presents how the system could be influenced by threats as well as which locations of the system are more vulnerable to them. Therefore, it is the very first and principal phase of a reliable design process.

Throughout the literature, reliability assessment methods for DNNs are mainly categorized into two major classes: fault injection (FI) and resilience analysis. The majority of the works assess the reliability of DNNs relying on FI, which provides realistic results on the impact of different fault models on the system's execution and is performed directly on the target platform (accelerator's software [9] or RTL model [10], FPGA [11], GPU [12]). FI outputs different evaluations for DNNs' reliability by accuracy loss, vulnerability factors, or fault classification [11], [13], [14]. Moreover, fine-grain evaluations for finding critical bits can be performed by exhaustive FI or an optimized method in [15].

Nevertheless, FI methods are prohibitively time-consuming and carry a high complexity due to the need to inject an enormous amount of faults into a huge number of DNN parameters as

The work is supported in part by the European Union through European Social Fund in the frames of the "Information and Communication Technologies (ICT) programme" ("ITA-IoIT" topic), by the Estonian Research Council grant PUT PRG1467 "CRASHLES" and by Estonian-French PARROT project "EnTrustED".

979-8-3503-3634-4/23/$31.00 ©2023 IEEE

well as time instances to reach an acceptable confidence level [16], [17]. The more fine-grain evaluation is required the more sophisticated experiments should be performed. In addition, most faults in a FI experiment on DNNs are masked [18] and are thus unnecessarily examined. Furthermore, the outcome of such assessment is application/platform specific which can not be generalized for other platforms [19].

Resilience analysis methods cope with the drawbacks of FI. They analyze the function of DNNs mathematically and have the potential to evaluate their reliability with arbitrary metrics. Therefore, resilience analysis methods can provide a deeper insight into the reliability evaluations of DNNs with lower complexity. Moreover, they can be conducted in different fault-tolerant designs on various platforms [20].

Layer-wise Relevance Propagation (LRP) algorithm is leveraged in [21]–[24] to obtain the contribution of neurons to the output to express their criticality and apply protections to improve the reliability of DNN accelerators. The sensitivity of DNN's filters is obtained by Taylor expansion with given error rates in [25] for designing an error-resilient and energy-efficient accelerator.

The conducted resilience analyses in these works are not able to provide reliability measurement metrics and detailed vulnerability evaluations. Moreover, they combine the criticality scores of neurons over individual outputs of the DNNs, thus resulting in missing important information about the resilience of DNNs as a whole. Mahmoud et al. [20] proposed different heuristics for vulnerability estimation of feature maps without FI. These estimations which are more coarse grain than the LRP-based methods, lead to hardening the accelerators, however, the accuracy of the vulnerability estimation methods is remarkably lower than that of fault-injection methods.

The aforementioned papers on resilience analysis methods have focused mainly on finding the most critical neurons/weights in a DNN to protect them against faults in a fault-tolerant design. In addition, they do not explain sufficiently how a fault propagates through the network and influence its outputs. Fidelity framework [26] is proposed to take advantage of both FI and analyzing DNN accelerators to provide reliability metrics. However, it requires detailed information of the accelerator architecture/implementation. To the best of our knowledge, there is no accelerator-agnostic resilience analysis method for DNNs that can compete with FI in terms of reliability evaluation to be less time-consuming, and accurate with fine-grain metrics enabling different reliability improvement techniques.

In this research work, we introduce the concept of neurons' vulnerability ranges expressing whether or not a fault at the output of neurons would misclassify the network. Thus, it enables a comprehensive reliability study with a novel resilience analysis method called DeepVigor where the vulnerability factors of layers, neurons, and bits in a DNN are obtained. The contributions in this work are:

- Proposing DeepVigor, a novel accurate, metric-oriented, and accelerator-agnostic resilience analysis method for DNNs reliability assessment faster than fault injection;
- Introducing and acquiring vulnerability ranges for all neurons in DNNs, assisted by a fault propagation analysis, providing accurate categorization of critical/non-critical faults;
- Providing fine-grain vulnerability factors as reliability evaluation metrics for layers, neurons, and bits in DNNs, compared with and validated by fault injection.

The remainder of the paper is organized as follows: the resilience analysis method is presented in Section II, and the experimental setup and results are provided in Section III. The applicability of the method is discussed in Section IV, and the work is concluded in Section V.

II. DNN Reliability Assessment with DeepVigor

A. Fault Model

In this work, the fault propagation analysis is performed at the outputs of DNN neurons. However, they will cover a vast majority of internal faults of the neurons occurring inside the MAC units and also a large portion of faults in the weights and neurons' input activations. It is assumed that only one neuron has an erroneous output per execution due to faults which is a common assumption in the literature [15].

For validation by FI, the single-bit fault model has been applied. While the multiple-bit fault model is more accurate, it requires a prohibitively large number of fault combinations to be considered ($3^n - 1$ combinations, where n is the number of bits). Fortunately, it has been shown that high fault coverage obtained using the single-bit model results in a high fault coverage of multiple-bit faults [27]. Therefore, a vast majority of practical FI and test methods are based on the single-bit fault assumption. Single bitflip faults are injected randomly at neurons' outputs and once per execution.

B. Fault Propagation Analysis

Fig. 2 depicts an overview of the rationale behind the DeepVigor method. A tiny neural network with few layers and neurons with given inputs, golden (fault-free) activation values (inside of neurons), and weights (on the arrows) is shown. The golden classification output is *class1*. A fault changes the neuron's output by δ which is the difference between the golden and faulty activation values. This δ that can have either a negative or a positive value will be propagated to the output layer and may change the classification result. The fault propagation will make a difference on each output class as Δ_1 and Δ_2. Misclassification happens when the value of the output neuron *class2* gets higher than that of neuron *class1*.

Thus, the propagation of the fault can be traced from the neuron to the output and a problem for misclassification can be expressed as shown in Fig. 2. By solving the problem of misclassification condition in the output, the value for δ is obtained as a vulnerability threshold that expresses how much a fault should influence the neuron to misclassify the network. Therefore, a vulnerability value range for the neuron is acquired. In this example, the range $(-\infty, -5.39)$ is a vulnerable range and $[-5.39, +\infty)$ is non-vulnerable range. This idea is generalized for a DNN including multiple output classes and other corresponding functions in this paper.

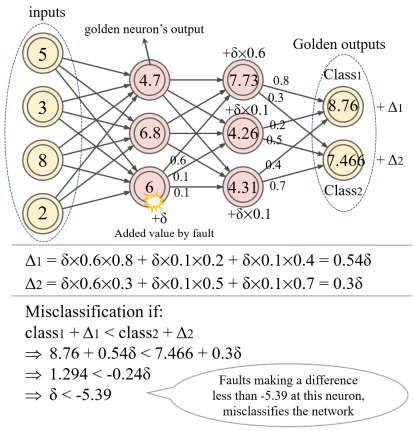


Fig. 2: An example of fault propagation analysis model and finding the vulnerability value ranges for a neuron with a given input.

C. The DeepVigor Method

The steps of the proposed DNNs' resilience analysis method (DeepVigor) and its validation are illustrated in Fig. 3. As shown, an analysis is performed on a set of data (i.e., set1, training set) and outputs the vulnerability value ranges as well as the vulnerability factors. Furthermore, FI is performed on the same and different data (i.e., set2, test set) to validate the outcomes of the analysis. The steps of DeepVigor are as follows:

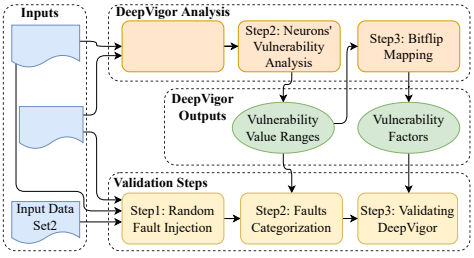


Fig. 3: Steps of the DeepVigor method for DNNs' reliability assessment and its validation.

Step1 - Gradient-based Initialization: In the first step, a neuron is examined whether or not to be processed for the vulnerability analysis. For this purpose, assuming a neural network consisting of L layers with N output classes in $C = \{c_1, c_2, ..., c_N\}$. Neuron k at layer l is selected to be examined. The neuron's output is corrupted by adding a sample positive or negative value as ϵ_k^l to its output and the feed-forward of the network is executed over a batch of input data. A loss function $\mathcal{L}$ is defined in Equation (1) as:

$$\mathcal{L} = sigmoid(\sum_{j=0}^{N}(\mathcal{E}_{c_t} - \mathcal{E}_{c_i})) \tag{1}$$

where c_t is the golden top class and $\mathcal{E}_{c_t}$ and $\mathcal{E}_{c_i}$ are the erroneous output values corresponding to the respective classes. The loss function computes the summation of differences between the value of the golden top class and the other outputs in the corrupted network and applies a $sigmoid$ function. The *golden top class* is what the fault-free DNN outputs as its classification whether or not it is correctly classified.

$\mathcal{L}$ represents the impact of the neuron's erroneous output on the golden top class of the network. When the gradient of $\mathcal{L}$ w.r.t. the corrupted neuron's output for one input is zero, it means that any error at this neuron's output does not change the output classification. Considering a batch of inputs, if the gradients are zero for a portion of inputs larger than a threshold, the neuron is disregarded for the vulnerability analysis. In case most of the gradients are not zero, a range for searching the vulnerability value is initialized.

Considering ϵ_k^l is a positive value for one input, in case the gradient is positive, there is a minimum value $0 < \delta_k^l < \epsilon_k^l$ for the neuron that if error δ_k^l is added to its output (by a fault at its inputs or the output value itself) the network's golden classification would change. But if the gradient is negative, then δ_k^l should be searched through the values larger than ϵ_k^l. A similar scenario is valid for negative values of ϵ_k^l.

Step2 - Neurons' Vulnerability Analysis: In this step, the vulnerability ranges of neurons under analysis are obtained. Let $R_{NV}(l,k,x) = [r_{lower}, r_{upper}]$ be a *Range of Non-vulnerable Values* for a k-th neuron at layer l with input data x. The bounds of range R for x are calculated as follows:

$$\begin{cases} r_{upper} = min(\delta_k^l), \delta_k^l > 0, \mathcal{E}_{c_t} < \mathcal{E}_{c_i}, i \neq t \\ r_{lower} = max(\delta_k^l), \delta_k^l < 0, \mathcal{E}_{c_t} < \mathcal{E}_{c_i}, i \neq t \end{cases} \tag{2}$$

where c_t and c_i are the golden top class and any other output class, respectively, and $\mathcal{E}_{c_t}$ and $\mathcal{E}_{c_i}$ are the erroneous output values corresponding to the respective classes.

Equation (2) finds the maximum negative and minimum positive values induced at the corresponding neuron that do not lead to misclassifying the input data from the golden classification. Further, a *Range of Vulnerable Values* $R_{VV}(l,k,x)$ for a k-th neuron at layer l with input data x is equal to $R_{VV} = (-\infty, r_{lower}) \cup (r_{upper}, \infty)$.

Note, the equation is applied for a single input data. In the case of a data set X containing T input data x_j the R_{NV} and R_{VV} will get refined and will be equal to intersections of their respective ranges over all inputs x_j as follows:

$$\begin{cases} R_{NV}(l,k) = \bigcap_{j=1}^{T} R_{NV}(l,k,x_j) \\ R_{VV}(l,k) = \bigcap_{j=1}^{T} R_{VV}(l,k,x_j) \end{cases} \tag{3}$$

The outcome of solving the equations for each neuron and merging the results over all inputs will be the vulnerability value ranges for each class separately, each range specifies the impact of a fault on changing the neuron value whether it influences the network classification result or not. Fig. 4 depicts different cases for vulnerability ranges over all numbers. Three vulnerability ranges are identified as follows:

- **Non-vulnerable range:** If a fault lay an effect on the neuron output in this range, no misclassification happens (hachured-green sections in Fig. 4);

- **Vulnerable range:** If a fault makes a difference at the output of the neuron in this range, the output will be misclassified (cross hachured-red sections in Fig. 4);
- **Semi-vulnerable range:** If a fault causes the neuron value to move as an amount in this range, this fault *may* cause a misclassification (dashed-grey sections in Fig. 4). Cases *d-f* in Fig. 4 happen when the portion of zero gradients in *step1* is less than the *threshold* and more than $1 - threshold$.

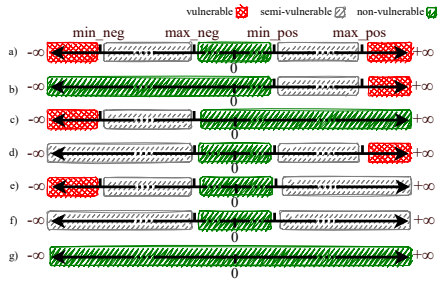


Fig. 4: Different possible cases of vulnerability ranges for each class in a neuron.

Step3 - Bitflip Mapping: In this step, DeepVigor maps the neurons' bitflipped values over input data on the vulnerability value ranges to indicate fine-grain vulnerability factors as metrics for the DNNs' reliability. For this purpose, the inputs used in *step2* and obtained vulnerability value ranges are fed to the network and in each bit of each neuron, bitflips are performed. In each bitflip, the difference in the new value of the target neuron is calculated and compared with the corresponding vulnerability range.

Based on the range of what the bitflip maps, the bit is considered vulnerable or non-vulnerable, respectively. By this analysis, the number of vulnerable bits of the neurons is obtained over the inputs. Hence, vulnerability factors of each layer (LVF), neuron (NVF), or bit (BVF) of the DNN can be defined as equations (4), (5), and (6), respectively. Vulnerability factors express the probability of misclassifying the network in case of the occurrence of a bitflip at the target element.

$$LVF = \frac{\#vulnerable\ bits\ in\ layer}{\#inputs\ \times \#layer's\ neurons \times word\ length} \times 100 \quad (4)$$

$$NVF = \frac{\#vulnerable\ bits\ in\ neuron}{\#inputs\ \times word\ length} \times 100 \quad (5)$$

$$BVF = \frac{\#vulnerable\ times\ for\ bit}{\#inputs} \times 100 \quad (6)$$

D. Validating DeepVigor By Fault Injection

As illustrated in Fig. 3, DeepVigor results are validated by means of FI over the input data and categorizing faults based on the vulnerability value ranges. The steps of the validation process of DeepVigor are as follows:

Step1 - Random Fault Injection: According to the adopted fault model, when one input is fed to the network, a random single bitflip is injected into a random neuron in a layer. This process is repeated several times for one input depending on the number of neurons and word length of data to reach a 95% confidence level and 1% error margin based on [28]. The required number of faults is obtained by Equation (7) where $N = word\ length \times \#layer's\ neurons$ that represents the total number of bits in the output of a layer.

$$\#layer's\ random\ faults = \frac{N}{1 + (0.01^2 \times \frac{N-1}{1.96^2 \times 0.5^2})} \quad (7)$$

Step2 - Fault Categorization: Once a fault is injected, a difference is produced in the output of the neuron in comparison with the golden model. In this step, the produced difference by a fault at the neuron's output is compared with the obtained vulnerability ranges, and faults are categorized as:

- **Non-critical fault:** The produced difference is in the non-vulnerable range.
- **Critical fault:** The produced difference is in the vulnerable range.

Step3 - Validating DeepVigor: To validate DeepVigor by FI, injected faults are propagated to the output and the network classification output is examined. The accuracy of the method is defined based on the two metrics as follows:

- **True non-critical faults:** Percentage of faults that are categorized as non-critical and do not change the classification at the output;
- **True critical faults:** Percentage of faults that are categorized as critical and change the classification at the output.

Another metric for validating the outputs of DeepVigor is the correlation between LVF and DNN's accuracy loss. This correlation shows that the obtained vulnerability factors from DeepVigor represent the criticality of the components properly. Since other vulnerability factors (NVF and BVF) are calculated using the same vulnerability ranges, by validating LVF, they will be also liable metrics for the resilience analysis, consequently.

III. Experimental Results

A. Experimental Setup

All DNNs, steps of DeepVigor, and its validation are implemented in PyTorch and run on NVIDIA 3090 GPU. To explore different DNN structures, six representative DNNs trained on three datasets are examined for the experimental results. We have experimented with two 5-layer MLPs (one with Sigmoid and one with ReLU) trained on MNIST, two LeNet-5 with 3 convolutional (CONV) layers, 2 max-pooling (POOL) layers, and 2 fully-connected (FC) layers trained on MNIST and CIFAR-10, AlexNet with 5 CONV, 3 POOLs, 2 batch normalization (BN) and 3 FCs trained on CIFAR-10, and VGG-16 with 13 CONV, 13 BNs, 5 POOLs and 2 FCs trained on CIFAR-100. The respective networks' accuracy on the corresponding test sets are 94.64%, 90.55%, 90.4%, 66.15%, 72.73%, and 69.41%.

Data representation in this work is 32-bit floating point IEEE-754 and the $word\ length$ in equations (4)-(7) is 32 bits. For

TABLE I: Accuracy of DeepVigor by fault injection on the same input data as the analysis.

DNN	True non-critical faults	True critical faults
MLP-sigmoid-mnist	99.985%∼100%	100%
MLP-relu-mnist	99.991%∼100%	100%
LeNet-mnist	99.992%∼100%	100%
LeNet-cifar10	99.956%∼100%	100%
AlexNet-cifar10	99.973%∼100%	99.955%∼100%
VGG16-cifar100	99.950%∼100%	99.972%∼100%

validation, a layer-wise statistical random FI is performed that satisfies a 95% confidence level and 1% error margin.

In the first step of DeepVigor ϵ_k^l is considered $-/+10000$ for range initialization and the whole search range is $[-5 \times 10^5, 5 \times 10^5]$. Finding δ_k^l in all networks by a logarithmic search is performed for negative and positive numbers separately, considering a 0.05 difference from the main value. Also, based on empirical explorations the threshold of neurons' zero-gradients for inputs is considered 98% for all experiments. Corresponding experiments are performed on the whole sets of training (as the input data set1) and test (as the input data set2) data.

B. Results and Validation

We analyze all neurons of the representative DNNs with training sets as the input data set1 by DeepVigor and obtain the vulnerability ranges. In the fault categorization step, faults are categorized into critical and non-critical classes with an accuracy close to 100%. Throughout the results from FI experiments, DeepVigor identified 66.63% to 99.42% of faults as non-critical over different layers of analyzed networks.

For validation, Table I presents the range of obtained accuracy values of the method through all layers of DNNs in terms of true non-critical and critical faults. It is observed that the accuracy of the method for categorizing non-critical faults is 99.950% to 100% and for critical faults ranging from 99.955% to 100% for the same data set.

The minor error seen in the results is due to: 1) Considered error in finding vulnerability values, 2) FI results in "NaN" values in 32-bit floating point IEEE-754 while the computations are being done on a GPU. We have categorized them as critical faults, 3) the effect of few inputs with non-zero gradients in *step1* as described in II-C.

We have also experimented with FI on the test sets (input data set2) to see the validity of the analysis on different sets reported in Table II. As it can be seen, similar high accuracy values to input data set1 are obtained.

TABLE II: Accuracy of DeepVigor by fault injection on a different input data from the analysis.

DNN	True non-critical faults	True critical faults
MLP-sig-mnist	99.985%∼99.996%	99.911%∼100%
MLP-relu-mnist	99.976%∼100%	100%
LeNet-mnist	99.992%∼100%	100%
LeNet-cifar10	99.952%∼100%	99.970%∼100%
AlexNet-cifar10	99.951%∼99.997%	99.948%∼99.998%
VGG16-cifar100	99.950%∼99.983%	99.972%∼99.998%

To validate the vulnerability factors, Fig. 5 illustrates the correlation between LVF and accuracy loss for a layer-wise FI on AlexNet. As demonstrated, there is a close relationship between the LVF obtained from DeepVigor and accuracy loss in FI, either the input sets are similar or different. This correlation is observed similarly in the results for all experimented DNNs. Therefore, LVF represents the vulnerability of layers competently.

DeepVigor also provides *NVF* and *BVF* metrics as vulnerability factors for neurons and bits, respectively. As a representative example, Fig. 6 depicts *NVF* for layer *conv3* of LeNet5-mnist and LeNet5-cifar10 that the more vulnerable neurons can be identified. In this figure, the number of neurons is sorted in each DNN separately, in the ascending order of NVF. Also, *BVF* for all neurons in DNNs is obtained and the results show that the most significant bit of exponents is the most vulnerable bit in most cases.

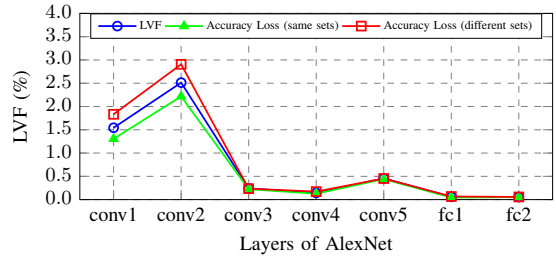


Fig. 5: Correlation between LVF and accuracy loss.

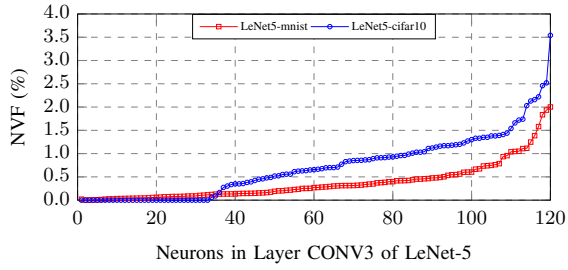


Fig. 6: NVF of neurons in CONV3 for LeNet5-mnist and LeNet-cifar10.

C. Run-Time Analysis

DeepVigor enables a fine-grain reliability evaluation for DNNs faster than exhaustive FI. In our experiments, *step1* of DeepVigor have removed up to 48% of neurons' vulnerability analysis to be processed in *step2*. Moreover, the range initialization in *step1* has accelerated the search for finding the vulnerability values for 50% to 99% of neurons in *step2* among the DNNs. Based on our experiments, a complete vulnerability range (as in Fig. 4) for one neuron can be obtained by 9.1 times feed-forward execution per neuron on average. While an exhaustive FI experiment runs the feed-forward by the number of bits (32 in our case) per neuron. Therefore, DeepVigor requires 3.5 times fewer feed-forwards translating into a similar amount of speed-up in run-time.

The run-time of DeepVigor depends on:

- Backpropagation execution by the number of neurons *step1* (one for positive and one negative numbers per neuron);
- Feed-forward execution by the number of searches for finding a positive or negative δ_k^l per neuron, in which the

best case is 0 searches (in case of zero gradients), the moderate case is 14 searches (in case of limited range initialization), and the worst case is 22 searches;
- Vulnerability analysis of the neurons in the last layer is performed by simplified mathematics similar to Fig. 2 and requires no iterative feed-forward or searching process through a wide range of numbers;
- Bitflip mapping is merely performing a bitflip at each neuron and a comparison with the obtained vulnerability ranges.

IV. Discussion

DeepVigor method is validated in the previous section, and it is shown how it can evaluate the reliability of DNNs proficiently with shorter run-times than FI. Vulnerability ranges enable a fine-grain and accurate resilience evaluation for neural networks. They are not limited to representing the single bitflip fault model and the outcome of the analysis is valid for an erroneous output for the neurons covering several fault models. This method enables an accelerator-agnostic analysis for DNNs and results can be applied to different accelerators.

The outputs of DeepVigor provide different possibilities for exploiting techniques of reliability improvement, including:

- Selective bits/neurons/layers hardening in accelerators based on the obtained BVF/NVF/LVF metrics;
- Fault-aware mapping for neurons on the processing elements of accelerators as in [21], [23];
- Applying range restriction for neurons' or layers' outputs for preventing faults propagation as in [9], [29], [30].

V. Conclusions

In this work, a novel resilience analysis method for DNNs reliability assessment named DeepVigor is proposed. The output of this method is the vulnerability value ranges for all neurons through the DNNs which result in vulnerability factors for all layers, neurons, and bits of the DNN, separately. The method is validated extensively by fault injection and its feasibility to categorize non-critical and critical faults on complex DNNs with 99.9% to 100% accuracy is demonstrated. Moreover, vulnerability factors obtained by the proposed analysis provide fine-grain criticality metrics for DNNs' components leading to different reliability improvement techniques. The DeepVigor method is very proficient in the evaluation and explanation of the reliability of DNNs with shorter run-times than fault injection.

References

[1] A. Bosio *et al.*, "Emerging computing devices: Challenges and opportunities for test and reliability," in *2021 IEEE ETS*. IEEE, 2021, pp. 1–10.

[2] H. Forsberg *et al.*, "Challenges in using neural networks in safety-critical applications," in *2020 AIAA/IEEE 39th Digital Avionics Systems Conference (DASC)*. IEEE, 2020, pp. 1–7.

[3] Y. Ibrahim *et al.*, "Soft errors in dnn accelerators: A comprehensive review," *Microelectronics Reliability*, vol. 115, p. 113969, 2020.

[4] M. Shafique *et al.*, "Robust machine learning systems: Challenges, current trends, perspectives, and the road ahead," *IEEE Design & Test*, vol. 37, no. 2, pp. 30–57, 2020.

[5] W. Li *et al.*, "Soft error mitigation for deep convolution neural network on fpga accelerators," in *2020 2nd IEEE AICAS*. IEEE, 2020, pp. 1–5.

[6] M. A. Neggaz *et al.*, "Are cnns reliable enough for critical applications? an exploratory study," *IEEE Design & Test*, vol. 37, no. 2, pp. 76–83, 2019.

[7] A. Azizimazreah *et al.*, "Tolerating soft errors in deep learning accelerators with reliable on-chip memory designs," in *2018 IEEE International Conference on Networking, Architecture and Storage (NAS)*. IEEE, 2018, pp. 1–10.

[8] S. Mittal, "A survey on modeling and improving reliability of dnn algorithms and accelerators," *Journal of Systems Architecture*, vol. 104, p. 101689, 2020.

[9] Z. Chen *et al.*, "A low-cost fault corrector for deep neural networks through range restriction," in *2021 51st IEEE/IFIP DSN*. IEEE, 2021, pp. 1–13.

[10] D. Xu *et al.*, "A hybrid computing architecture for fault-tolerant deep learning accelerators," in *2020 IEEE 38th International Conference on Computer Design (ICCD)*. IEEE, 2020, pp. 478–485.

[11] D. Xu *et al.*, "Reliability evaluation and analysis of fpga-based neural network acceleration system," *IEEE TVLSI*, vol. 29, no. 3, pp. 472–484, 2021.

[12] P. M. Basso *et al.*, "Impact of tensor cores and mixed precision on the reliability of matrix multiplication in gpus," *IEEE Transactions on Nuclear Science*, vol. 67, no. 7, pp. 1560–1565, 2020.

[13] F. F. dos Santos *et al.*, "Analyzing and increasing the reliability of convolutional neural networks on gpus," *IEEE Transactions on Reliability*, vol. 68, no. 2, pp. 663–677, 2018.

[14] N. Khoshavi *et al.*, "Shieldenn: Online accelerated framework for fault-tolerant deep neural network architectures," in *2020 57th ACM/IEEE DAC*. IEEE, 2020, pp. 1–6.

[15] Z. Chen *et al.*, "Binfi: An efficient fault injector for safety-critical machine learning systems," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2019, pp. 1–23.

[16] M. Taheri, M. H. Ahmadilivani *et al.*, "Deepaxe: A framework for exploration of approximation and reliability trade-offs in dnn accelerators," in *2023 ISQED*. In press, 2023.

[17] M. Taheri, M. H. Ahmadilivani *et al.*, "Appraiser: Dnn fault resilience analysis employing approximation errors," in *2023 DDECS*. In press, 2023.

[18] A. Bosio *et al.*, "A reliability analysis of a deep neural network," in *2019 IEEE LATS*. IEEE, 2019, pp. 1–6.

[19] A. Ruospo *et al.*, "Pros and cons of fault injection approaches for the reliability assessment of deep neural networks," in *2021 IEEE LATS*. IEEE, 2021, pp. 1–5.

[20] A. Mahmoud *et al.*, "Hardnn: Feature map vulnerability evaluation in cnns," *arXiv preprint arXiv:2002.09786*, 2020.

[21] C. Schorn *et al.*, "Accurate neuron resilience prediction for a flexible reliability management in neural network accelerators," in *2018 DATE*. IEEE, 2018, pp. 979–984.

[22] C. Schorn *et al.*, "An efficient bit-flip resilience optimization method for deep neural networks," in *2019 DATE*. IEEE, 2019, pp. 1507–1512.

[23] A. Ruospo and E. Sanchez, "On the reliability assessment of artificial neural networks running on ai-oriented mpsocs," *Applied Sciences*, vol. 11, no. 14, p. 6455, 2021.

[24] M. Abdullah Hanif and M. Shafique, "Salvagednn: salvaging deep neural network accelerators with permanent faults through saliency-driven fault-aware mapping," *Philosophical Transactions of the Royal Society A*, vol. 378, no. 2164, 2020.

[25] W. Choi *et al.*, "Sensitivity based error resilient techniques for energy efficient deep neural network accelerators," in *2019 DAC*, 2019, pp. 1–6.

[26] Y. He *et al.*, "Fidelity: Efficient resilience analysis framework for deep learning accelerators," in *2020 53rd IEEE/ACM MICRO*. IEEE, 2020, pp. 270–281.

[27] M. Bushnell and V. Agrawal, *Essentials of electronic testing for digital, memory and mixed-signal VLSI circuits*. Springer Science & Business Media, 2004, vol. 17.

[28] R. Leveugle *et al.*, "Statistical fault injection: Quantified error and confidence," in *2009 DATE*. IEEE, 2009, pp. 502–506.

[29] L.-H. Hoang *et al.*, "Ft-clipact: Resilience analysis of deep neural networks and improving their fault tolerance using clipped activation," in *2020 DATE*. IEEE, 2020, pp. 1241–1246.

[30] B. Ghavami *et al.*, "Fitact: Error resilient deep neural networks via fine-grained post-trainable activation functions," in *2022 DATE*. IEEE, 2022, pp. 1239–1244.

Appendix 9

IX

M. H. Ahmadilivani, M. Taheri, J. Raik, M. Daneshtalab, and M. Jenihhin, "A systematic literature review on hardware reliability assessment methods for deep neural networks," ACM Computing Surveys, vol. 56, no. 6, pp. 1–39, 2024.

A Systematic Literature Review on Hardware Reliability Assessment Methods for Deep Neural Networks

MOHAMMAD HASAN AHMADILIVANI, MAHDI TAHERI, and JAAN RAIK, Tallinn University of Technology, Estonia
MASOUD DANESHTALAB, Mälardalen University, Sweden and Tallinn University of Technology, Estonia
MAKSIM JENIHHIN, Tallinn University of Technology, Estonia

Artificial Intelligence (AI) and, in particular, Machine Learning (ML), have emerged to be utilized in various applications due to their capability to learn how to solve complex problems. Over the past decade, rapid advances in ML have presented Deep Neural Networks (DNNs) consisting of a large number of neurons and layers. DNN Hardware Accelerators (DHAs) are leveraged to deploy DNNs in the target applications. Safety-critical applications, where hardware faults/errors would result in catastrophic consequences, also benefit from DHAs. Therefore, the reliability of DNNs is an essential subject of research.

In recent years, several studies have been published accordingly to assess the reliability of DNNs. In this regard, various reliability assessment methods have been proposed on a variety of platforms and applications. Hence, there is a need to summarize the state-of-the-art to identify the gaps in the study of the reliability of DNNs. In this work, we conduct a Systematic Literature Review (SLR) on the reliability assessment methods of DNNs to collect relevant research works as much as possible, present a categorization of them, and address the open challenges.

Through this SLR, three kinds of methods for reliability assessment of DNNs are identified, including Fault Injection (FI), Analytical, and Hybrid methods. Since the majority of works assess the DNN reliability by FI, we characterize different approaches and platforms of the FI method comprehensively. Moreover, Analytical and Hybrid methods are propounded. Thus, different reliability assessment methods for DNNs have been elaborated on their conducted DNN platforms and reliability evaluation metrics. Finally, we highlight the advantages and disadvantages of the identified methods and address the open challenges in the research area. We have concluded that Analytical and Hybrid methods are light-weight yet sufficiently accurate and have the potential to be extended in future research and to be utilized in establishing novel DNN reliability assessment frameworks.

CCS Concepts: • **General and reference** → **Surveys and overviews**; • **Hardware** → **Hardware reliability**; • **Computer systems organization** → **Neural networks**; **Reliability;**

This work was supported in part by the European Union through the European Social Fund in the frames of the "Information and Communication Technologies (ICT) programme" ("ITA-IoIT" topic), the Estonian Research Council grant PRG1467 "CRASHLESS," the Estonian-French science and technology cooperation programme PARROT project "EnTrustED," and by the Swedish Innovation Agency VINNOVA project SafeDeep.

Authors' addresses: M. H. Ahmadilivani, M. Taheri, J. Raik, and M. Jenihhin, Tallinn University of Technology, Tallinn, Estonia; e-mails: {mohammad.ahmadilivani, mahdi.taheri, jaan.raik, maksim.jenihhin}@taltech.ee; M. Daneshtalab, Mälardalen University, Västerås, Sweden and Tallinn University of Technology, Tallinn, Estonia; e-mail: masoud.daneshtalab@taltech.ee.

0360-0300/2024/01-ART141
https://doi.org/10.1145/3638242

Additional Key Words and Phrases: Reliability assessment, deep neural networks, DNN hardware accelerator, fault injection

ACM Reference format:
Mohammad Hasan Ahmadilivani, Mahdi Taheri, Jaan Raik, Masoud Daneshtalab, and Maksim Jenihhin. 2024. A Systematic Literature Review on Hardware Reliability Assessment Methods for Deep Neural Networks. *ACM Comput. Surv.* 56, 6, Article 141 (January 2024), 39 pages.
https://doi.org/10.1145/3638242

1 INTRODUCTION

Deep Neural Networks (DNNs) are nowadays extensively applied to a wide variety of applications due to their impressive ability to approximate complex functions (e.g., classification and regression tasks) via learning. Since powerful processing systems have evolved in the recent decade, DNNs have emerged to be deeper and more efficient as well as employed in an ever broader extent of domains. Meanwhile, using **DNN Hardware Accelerators (DHAs)** in safety-critical applications, including autonomous driving, raises reliability concerns [1, 2]. In compliance with ISO 26262 functional safety standard for road vehicles, the evaluated **FIT (Failures In Time)** rates of hardware components must be less than 10 (meaning 10 failures in 1 billion hours) to pass the highest reliability level [3], which requires diligent design.

DNNs are deployed in their target application by different DHA platforms, including **Field-Programmable Gate Arrays (FPGAs)**, **Application-Specific Integrated Circuits (ASICs)**, and **Graphics Processing Units (GPUs)** [4]. Depending on the DHA and the application's environment, different fault types may present a threat to the reliability of the component [5]. Figure 1 illustrates the reliability threats (described in Section 2.3) in an example DHA. In this figure, different fault types originating from several reasons could occur in any of the DHA's components that may lead to a disastrous misclassification, e.g., once a red light is detected as a green light. Faults are originated from hardware, however, they can also be modeled at software platforms for the ease of study. Accordingly, the reliability of DNNs is tightly coupled with the reliability of DHAs as faults are coming from hardware. It is worth highlighting that the reliability in this article does not relate to the reliability in software engineering or security issues, e.g., adversarial attacks.

It has been shown in several studies that the functionality of DNNs in terms of accuracy is remarkably degraded in the presence of faults [6–10]. Recently, numerous research works have been published on the assessment and enhancement of DNNs' reliability. However, due to the extent of the DNNs domain, these works approach the problem of the reliability of DNNs from various perspectives. We are faced with several applications of DNNs as well as a variety of DNN algorithms for different tasks. Therefore, it will lead to distinct platforms and reliability threats, which hinders unifying and generalizing the methods of reliability assessment and enhancement of DNNs.

Throughout the literature, various methods of DNN reliability assessment and enhancement are presented. Some review papers have been published on the topic of DNNs reliability enhancement methods [4, 5, 11–14]. These works aim to formulate the reliability problem in DNNs, categorize available reliability improvement methods in this domain, and overview the fault injection methods for reliability assessment. The analysis in Reference [14] is the first review on the subject of fault tolerance in DNNs and describes different fault models and reliability improvement methods in DNNs. However, the topic was still not as mature as it is today, and numerous works have been published afterwards. Subsequent works such as References [4, 5, 11] provide extensive reviews on the reliability improvement methods for DNNs and characterize taxonomies of different methods. Nevertheless, they do not consider the assessment and evaluation methods of the reliability for

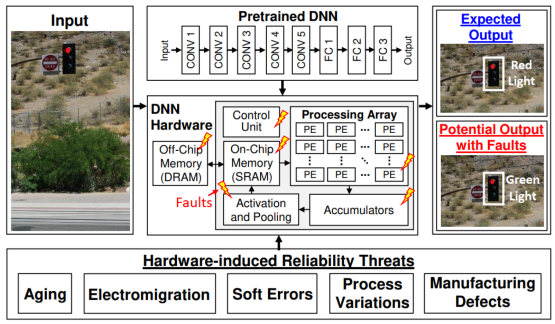


Fig. 1. Hardware-induced reliability threats in an example DHA and their possible impact on the output [1].

DNNs. Other surveys [12, 13] have reviewed fault injection methods for DNNs reliability assessment, with the former work focused merely on fault criticality assessment and the latter including only a few papers in the survey. In this article, we present the first **Systematic Literature Review (SLR)** dedicated to all methods of reliability assessment of DNNs.

Reliability assessment of DNNs is a process for evaluating the reliability of a DNN that is being executed either as a software model or by a hardware platform. However, the assessment method for reliability may vary, depending on the platform. In this regard, it is necessary to comprehend and distinguish the different methods used to assess the reliability of DNNs across platforms. This article establishes a thorough picture of the reliability assessment methods for DNNs and systematically reviews the relevant literature. To achieve this, we carry out the SLR methodology [15, 16] to present this survey. The primary focus of this review is to investigate the methods of reliability assessment for DNNs, generalize and characterize the methods, and identify the open challenges in the domain.

To the best of our knowledge, this survey represents the first comprehensive literature review on reliability assessment methods for DNNs. We cover all published papers from 2017 to 2022 that could be found through a systematic search. The main contributions of this article are:

— Reviewing the literature of the reliability assessment methods of DNNs, systematically;
— Analyzing the trends of published papers over different years and methods;
— Characterizing and categorizing the reliability assessment methods for DNNs;
— Identifying fault injection methods based on the DNN platforms;
— Introducing analytical and hybrid reliability assessment methods along with fault injection;
— Addressing the open challenges in the research area and recommendations for future research directions.

The structure of the article is as follows: Section 2 presents the background on DNNs and reliability concepts; Section 3 explains the methodology of this survey and addresses the research questions; Section 4 reviews the study briefly, presents the statistics of the publications, and depicts the top-level taxonomy of reliability assessment methods for DNNs. In Section 5, the details of the reliability assessment methods are explained. Section 6 includes pros and cons of methods and open challenges of the study domain. Section 7 provides the conclusions of this survey.

2 PRELIMINARIES

2.1 Deep Neural Networks

Deep Learning (DL) is a sub-domain of **Machine Learning (ML)**, which is the study of making computers learn to solve problems without being directly programmed [17]. Regarding the

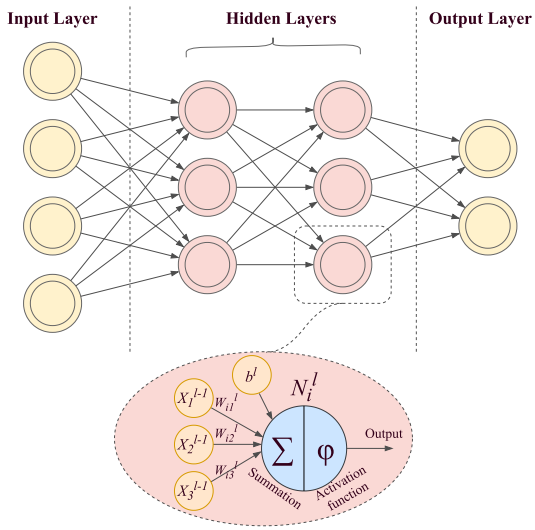


Fig. 2. Abstract view of a simple neural network with the detail of a neuron.

impressive ability of DNNs in learning, they are applicable in a vast variety of domains, such as image and video processing, data mining, robotics, autonomous cars, gaming, and so on.

DNNs are inspired by the human brain, and they have two major phases: training and inference. In the training phase, which is an iterative process and performed once, the hyper-parameters (e.g., weights and biases) of the neural network are updated on a determined dataset. A loss function is adopted in the training phase that measures the difference between the expected and the estimated output of DNN to achieve higher accuracy. Accuracy expresses the proportion of the DNN outputs coinciding with the expected output. However, in the inference phase, representing the DNN deployment, the network is run several times with the parameters obtained during the training phase [17].

DNNs are constructed of the units of neurons. Each neuron receives some activation inputs and multiplies them by the corresponding weights. Then, it conveys the summation of the weighted activations to its output. A set of neurons builds up a layer that may have other additional functions, e.g., activation function (ReLu, sigmoid, etc.), batch normalization, (max or average) pooling, and so on [17]. Equation (1) represents the function of the ith neuron in layer l (denoted as N_i^l) with input activations from the previous layer l-1 with n outputs (denoted as X^{l-1}), where W and b represent weights and bias, respectively.

$$N_i^l = \phi\left(\sum_{j=0}^{n} X_j^{l-1} \times W_{ij}^l + b^l\right) \tag{1}$$

An abstract view of a neuron and a neural network is depicted in Figure 2. As shown, inputs are fed into the network through the input layer. The middle layers, called hidden layers, determine the depth of the network and conduct the function of the DNN. The output layer is where the network decides. It produces some probabilities of the possible outputs, i.e., output confidence score, and the class with the highest value is the top-ranked output.

DNNs have various architectures each suitable for specific applications. Nevertheless, it is worth mentioning some terms that are used in this article. **Convolutional Neural Networks (CNNs)** are extensively used in classification, object detection, and semantic segmentation tasks and consist of multiple **convolutional (CONV)** and **fully connected (FC)** layers. CONV layers have

a set of **two-dimensional (2D)** weights, called filters, that extract a specific feature from the input of the layer. A channel is a set of **input feature maps (ifmap)** that is convolved with filters resulting in the **output feature maps (ofmap)** [17].

In the research area of CNNs, there are some models of networks that are most frequently used. For instance, LeNet-5 [18], AlexNet [19], GoogLeNet [20], VGG [21], and ResNet [22] are introduced for image classification, and YOLO [23] is designed for object detection. In addition, prominent datasets that are mostly used for training networks on image classification tasks are MNIST [24], CIFAR [25], and ImageNet [26]; and on object detection are KITTI [27] and PASCAL VOC [28].

In addition, due to the large number of parameters and calculations in DNNs, **Quantized Neural Networks (QNNs)** [29] and **Binarized Neural Networks (BNNs)** [30] are introduced to reduce the complexity, memory usage, and energy consumption of DNNs. These DNNs are the quantized versions of existing DNNs that reduce the bit-width of their parameters and calculations with an acceptable accuracy loss.

2.2 DNN Platforms

2.2.1 Software Frameworks. DNN software frameworks and libraries in high-level programming languages have been developed to ease the process of designing, training, and testing DNNs. These frameworks are widely used due to their high abstraction level of modeling and short design time. Some of well-known software frameworks that are being used for training the DNNs are: TensorFlow [31], Keras [32], PyTorch [33], DarkNet [34], and Tiny-DNN [35]. All these frameworks are capable of using both CPU and GPU to accelerate the training process.

2.2.2 DNN Hardware Accelerators (DHAs). DHAs are used for the training as well as the inference phase of DNNs. They are called accelerators due to their dedicated design employing parallelism for reducing the execution time of the DNN, either in training or inference. DHAs can be generally categorized into four classes: FPGAs, ASICs, GPUs, and multi-core processors [36, 37].

According to the literature review of DHAs in Reference [37], FPGAs are used more frequently than other DHA platforms in terms of implementing DNNs, due to their availability and design flexibility for different applications [38]. FPGAs are programmed via their configuration bits that determine the functionality of the FPGA. The system of FPGA-based DNN accelerators usually consists of a host CPU and an FPGA part with corresponding interconnections between them. In this design model, the DNN is implemented on the FPGA part and the CPU controls the accelerator with software, while each part is integrated with memories [38]. A typical structure of FPGA-based DNN accelerator is depicted in Figure 3, which is based on HW/SW co-design, which means separating the implementation of DNNs on the integrated CPU (the software) and FPGA (the hardware) that are communicating with one another [39]. **High-Level Synthesis (HLS)** tools, which can synthesize high-level programming languages to RTL, are also used for developing FPGA-based DNN accelerators [38].

ASIC-based DNN accelerators are more efficient than FPGAs in terms of performance and power consumption but less flexible in terms of applications and require a long design time [40]. There are two general types of architectures for ASIC-based DHA platforms: spatial and temporal [17]. Figure 4 depicts an example of a spatial architecture model that is constructed of 2D arrays of **Processing Elements (PEs)** flowing data horizontally and vertically from input/weight buffers to output buffers. PEs perform **Multiply-Accumulate (MAC)** operations on inputs and weights representing a neuron operation in the DNN. Off-chip memories are required to store the parameters of DNNs and save the intermediate results from PEs. **Tensor Processing Unit (TPU)**, produced by Google, one of the most applicable ASIC-based DNN accelerators, is based on this type of architecture [41].

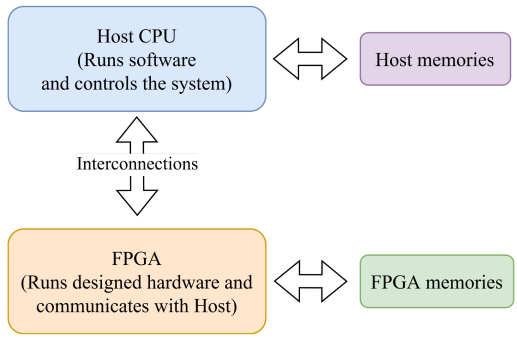


Fig. 3. Typical structure of an FPGA-based DNN accelerator [38].

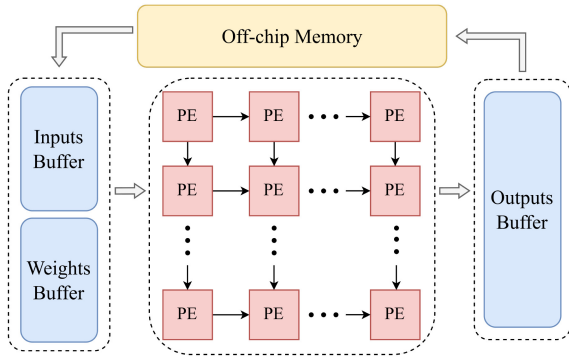


Fig. 4. An example of spatial architecture for ASIC-based DNN accelerators [42].

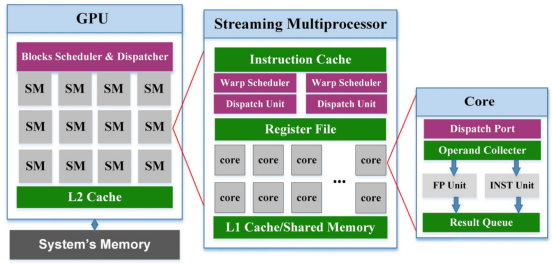


Fig. 5. General architecture of CUDA-based GPUs [44].

GPUs are a powerful platform for training and inferring deep networks and are vastly used in safety-critical applications [43]. GPUs include up to thousands of parallel cores, which make them efficient for DNN algorithms, especially in the training phase [40]. GPUs are designed to run several threads of a program and are also exploited to accelerate running DNNs [37]. The general architecture of GPUs is depicted in Figure 5. There are numerous **Streaming Multiprocessors (SMs)** in the GPU, each having several cores with a shared register file and caches, while a scheduler and dispatchers control the tasks among and within SMs and cores [44].

Multi-core processors, e.g., ARM processors, deploy DNNs mostly for edge processing and **Internet of Things (IoT)** applications [45–47]. They facilitate DNNs with parallel computing and low power consumption and provide a wider range of applications for DNNs.

2.3 Reliability, Threats, Fault Models, and Evaluation

Terms of robustness, reliability, and resilience are mostly used in the research pertaining to the reliability of DNNs. These terms are often used interchangeably and ambiguously. In the following, we present the definitions of these three terms as applied in the current literature review:

- **Reliability** concerns DNN accelerators' ability to perform correctly in the presence of faults, which may occur during the deployment caused by physical effects either from the environment (e.g., soft errors, electromagnetic effects) or from within the device (e.g., manufacturing defects, aging effects, process variations).
- **Robustness** refers to the property of DNNs expressing that the network is able to continue functioning with high integrity despite the alteration of inputs or parameters due to noise or malicious intent.
- **Resilience** is the feature of DNN to tolerate faults in terms of output accuracy.

In this work, we are concerned about the reliability of DNNs, which refers to the ability of accelerators to continue functioning correctly in a specified period of time with the presence of faults. Reliability in this article does not relate to the reliability and test in software engineering or security issues, e.g., adversarial attacks in which an attacker perturbs the inputs or parameters.

Faults are the sources of threatening the reliability of DNN accelerators (see Figure 1) that can be caused by several reasons, e.g., soft errors, aging, process variation, and so on [1]. Soft errors are transient faults induced by radiation that are caused by striking charged particles to transistors [48]. Aging is the time-dependent effect of the increasing threshold voltage of transistors due to physical phenomena that will lead to timing errors and permanent faults [49]. Process variations are alterations of transistor's attributes in the process of chip fabrication. As a consequence, voltage scaling may result in faults at the outputs of transistors during their operation [50].

Faults as reliability threats are generally modeled as *permanent* and *transient* faults [5, 11, 14]. Permanent faults result from process variations, manufacturing defects, aging, and so on, and they stay constant and stable during the runtime. However, transient faults are caused by soft errors, electromagnetic effects, voltage and temperature variations, and so on, and they show up for a short period of time. Nevertheless, once a faulty value from a component is read by another component and the propagated value does not coincide with the expected one, an *error* happens. Therefore, a fault is an erroneous state of hardware or software, and an error is a manifestation of it at the output. *Failure* or system malfunction is the corruption or abnormal operation of the system, which is caused by errors [14, 51, 52].

Faults may have different impacts on the output of DNNs and can be classified based on their effects. A fault may be masked or corrected if detected or result in different outputs compared to the fault-free execution (golden model), in which case, the fault is propagated and observed at the output. Faults observed at the output of the system can be classified in two categories: **Silent Data Corruption (SDC)** and **Detected Unrecoverable Errors (DUE)**, depending on whether a fault is undetected (SDC) or detected (DUE) [11, 53]. Figure 6 illustrates this general fault classification scheme regarding the output of systems adopted from Reference [51].

Reliability assessment is the process in which the target system or platform is modeled or presented, and by means of simulations, experiments, or analysis, the reliability is measured and evaluated. Reliability assessment is a challenging process, and several methods can be adopted for modeling and evaluating reliability. In general, evaluating the reliability of a system can be performed by three approaches: **Fault Injection (FI)** methods, analytical methods, and hybrid methods [54]. FI methods are exploited to inject a model of faults into the system implemented either in software or hardware, while the system is in simulation or being executed. Analytical

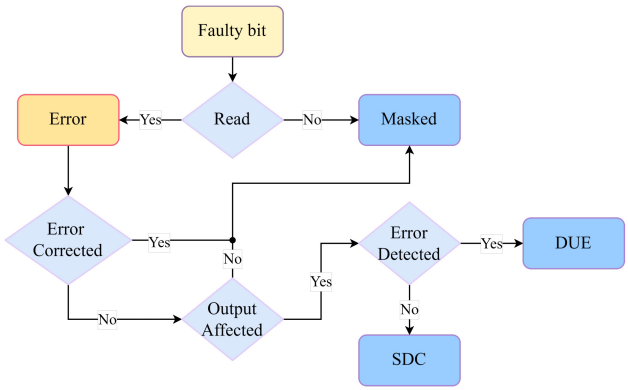


Fig. 6. The adopted fault classification based on the output point of view, as in Reference [51].

methods attempt to model the function of the system and its reliability with mathematical equations, depending on the target architecture. In hybrid methods, an analytical model is adopted alongside an FI to evaluate the reliability. Generally, FI methods are more realistic than analytical and hybrid methods; however, FI is a time-consuming process with a high computational complexity [55].

In the reliability assessment using FI, it is necessary to determine the target platform, potential fault locations (logic or memory), and the fault type (transient or permanent). Transient faults in logic show up in one clock cycle, while in the memory, they flip a bit that will remain until the end of the execution. Permanent faults are modeled as stuck-at-0 (sa-0), or stuck-at-1 (sa-1), and they exist during the whole execution. According to the selected fault model, perturbation of the model is performed, the system is run, and the outputs are gathered. The output of faulty execution should be compared with the one of the golden-model to measure the impact of faults on the system.

FI allows calculating reliability metrics, e.g., **Failures-In-Time (FIT), Architectural Vulnerability Factor (AVF)**, SDC rate, **Soft Error Rate (SER)**, cross-section, and so on. FIT is the number of failures in 10^9 hours, AVF is the probability of fault propagation from a component to other components in a design, SDC rate refers to the ratio of the outputs affected by faults, SER refers to the ratio of soft error occurrence, and cross-section is the proportion of observed errors over all collided particles. These quantitative evaluation metrics are usually tightly coupled to each other, yet follow a different purpose to express the reliability of a system.

Exhaustive fault injection into all bits of a platform at every clock cycle requires an extensive simulation. Therefore, to determine how many faults could be injected into the system to be representative statistically, a confidence level with an error margin is presented [56]. It provides a fault rate or **Bit Error Rate (BER)** for an FI experiment. The number of FI experiments' repetitions regarding the number of possible bit and clock cycle combinations to support the number of injected faults determines the execution space for the FI task.

3 REVIEW METHODOLOGY

Systematic Literature Review (SLR) is a standard methodology for reviewing the literature in a recursive process and minimizing bias in the study [15, 16, 37]. Hence, the SLR methodology is adopted in this survey. The methodology determines:

— Specifying the **Research Questions (RQs)**,
— Specifying the search method for finding and filtering the related papers,

— Extracting corresponding data from the found papers based on the RQs,
— Synthesizing and analyzing the extracted data.

Therefore, based on the aforementioned steps of SLR, the RQs that we attempt to answer are:

— **RQ1:** What is the distribution of the research works in the domain of reliability assessment? (To obtain the trend of publications in this domain).
— **RQ2:** What are the existing methods of reliability assessment for DNNs? (To comprehend the entire variety of methods in this domain).
— **RQ3:** How could the existing methods be characterized and categorized in terms of reliability assessment methods? (To categorize existing works and provide the taxonomy, a systematic instruction for finding the suitable method for potential applications in this domain).
— **RQ4:** What are the open challenges in the domain of reliability assessment methods for DNNs? (To specify the remaining areas for future research).

The motivation for this survey is the numerous recent papers published on the reliability of DNNs emphasizing the need for such a literature review. We have searched for the papers systematically through scientific search servers. The main databases and publishers we have used are: Google Scholar, IEEE Explore, ACM Digital Library, Science Direct, and Elsevier. The initial set of papers is provided by searching some keywords in the mentioned servers, including "reliability of DNNs", "hardware reliability of DNN accelerators", "resilient DNNs", "robust DNNs", "the vulnerability of DNNs", "soft errors in DNNs", "fault injection in DNNs" ("DNN" also replaced with "CNN").

Subsequently, based on the title and abstract of each paper, we select them. This selection is based on the criterion of whether the paper may be concerned with the reliability of DNNs or not. In addition, the references and citations of the papers have been checked for the chosen papers to find more related papers. In this process, we selected 242 papers based on their titles and abstracts.

In the next step, we study the introduction, conclusion, and methodology sections of each paper to decide whether we include the paper in the review or not. The inclusion criteria of the papers are:

— The paper is published by one of the scientific publishers and has passed through a peer-review process,
— The focus of the work is DNN, neither generic reliability assessment methods using DNNs as one of the examples nor employing DNNs for assessing the reliability of a platform.
— The work includes a reliability assessment method for DNNs,
— The method of reliability assessment is clear and well-defined,
— Terms including reliability, robustness, resilience, or vulnerability **must** refer clearly to reliability issues, as defined in Section 2.3.

Papers that have included similar keywords but have not matched the above conditions are excluded. As a result, we have included 139 papers published from 2017 to the end of 2022 in this literature review to build up the taxonomy of the literature review and methods' categorization.

In the following, we have designed a **Data Extraction Form (DEF)** based on the RQs. In this form, we have taken note of reviewing the papers to find some specific data such as:

— General method of reliability modeling (FI, analytical, or hybrid),
— The platform where DNNs are implemented,
— The fault model and fault locations in case of FI,
— Details of reliability assessment method,
— Reliability evaluation metrics.

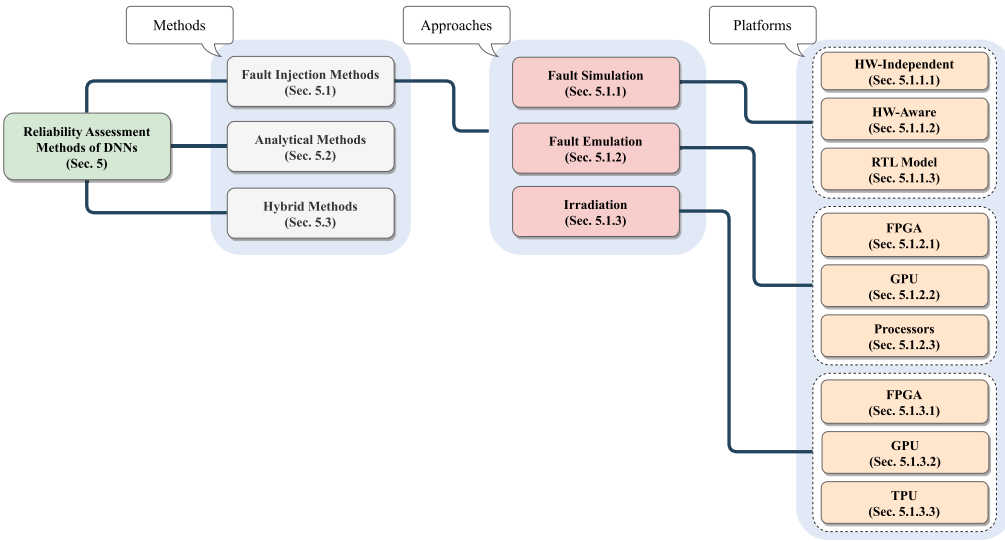


Fig. 7. Top-level overview of the reliability assessment methods in this work.

In the final step, after reviewing all the selected papers and filling in the DEF, we synthesized and analyzed the obtained data from the papers. Thereafter, we have provided the categorization taxonomy of the reliability assessment methods for DNNs, have characterized them in this article, and analyzed them to find the open challenges.

4 STUDY OVERVIEW

This section presents an overview of the study and the analyzed statistics of the included works in different categories. As mentioned, we have included 139 papers from 2017 to 2022 for categorizing the reliability assessment methods for DNNs.

4.1 Taxonomy

Figure 7 represents the top-level categorization overview of the study to address RQ2 and RQ3. Reliability assessment of DNNs is categorized into three main methods: Fault Injection, Analytical, and Hybrid.

4.1.1 ***Fault Injection (FI)*** *Methods.* The works based on this method evaluate the reliability of DNNs by fault injection campaign. There exist several taxonomies for the fault injection approaches in the hardware reliability domain [12, 54, 55, 57, 58]. Therefore, we adapt them for categorizing the related works on DNNs into three approaches addressed in Figure 7 and Table 1. FI methods are categorized into three approaches of fault injection as follows:

— **Fault Simulation:** DNNs are implemented either in software by high-level programming languages or **Hardware Description Languages (HDL)** and faults are injected into the model of the DNN. In the former case, some works consider a DHA model in their software implementations while others do not. We divide works on this approach into hardware-independent, hardware-aware, and RTL model platforms. RTL models represent ASIC-based DHAs.
— **Emulation in Hardware:** Research works on this approach implement and run DNNs on a DHA (i.e., FPGA, GPU, or processor) and inject the faults into the components of the accelerator by a software function, FI framework, and so on.

Table 1. Fault Injection Categorization with the Corresponding References

FI Method	Fault Simulation			Fault Emulation			Irradiation		
DNN Platform	**HW-independent**	**HW-aware**	**RTL Model**	**FPGA**	**GPU**	**Processors**	**FPGA**	**GPU**	**TPU**
Fault Type	Transient [59–61] [62–67] [68–73] [74–78] [79–82]	Transient [9, 83] [84–87] [88–91]	Transient [36, 92–94]	Transient [8] [95–99] [100–102] [103–105] [106–108]	Transient [10] [44, 109–111] [112–115] [116? –119] [120–122]	Transient [36] [92, 123, 124] [125–127] [128–130]	Transient [95, 96, 103, 106] [108, 131, 132] [133–135]	Transient [10, 115, 117] [121, 122] [136, 137]	Transient [138, 139]
	Permanent [72] [140–143]	Permanent [144] [6, 145–147] [148–150]	Permanent [7, 58, 151] [152–154]	Permanent [105, 106] [155, 156]	Permanent [137, 157, 158] [159, 160]				
Fault Location	Weights [61, 62] [63–65, 67, 69] [68, 70, 71, 73, 74] [75–79] [80–82, 140] [141–143, 161?]	Weights [9, 85, 86] [88–90] [91, 146, 148]	PEs, MACs [7, 151, 152] [153, 154]	Configuration Bits [8, 95] [96–99] [100–102] [103, 104, 107] [108, 162, 163]	Registers, Instructions [10, 44, 109, 110] [111–114] [115, 116, 118] [119–122] [157–160]	Register File [36, 92, 123, 124] [125–127] [128–130]	Entire FPGA Package [95] [96, 103, 132] [108, 131, 133] [135]	Entire GPU [10, 115, 117] [121, 122] [136? , 137]	Entire Chip refs [138, 139]
	Activations [59] [60, 64, 66, 72] [76, 78]	Activations [6] [9, 83? , 84] [87–90] [91, 144, 145] [147–149]	Registers, Buffers, LUTs [36, 92–94]	On-Chip Memories [8] [98, 100, 101] [102, 105, 106] [155, 162, 163]	Weights, Activations [117, 137]	Instructions [130]	HyperRAM [106, 134]		
Evaluation	Accuracy Loss [59] [60–63, 68] [69, 71–74] [75–79] [80–82, 141, 143]	Accuracy Loss [6, 9, 84, 87, 88] [89–91, 144] [145–147] [148–150]	Accuracy Loss [7, 93, 151] [152–154]	Accuracy Loss [8, 97, 99? , 100] [101, 102, 105] [106, 107, 156] [108, 155, 162, 163]	Accuracy Loss [113, 120, 158] [159, 160]	Fault Classification [36, 92, 123, 124] [125–127] [128–130]	Fault Classification [103, 133, 135]	Fault Classification [10, 115, 121] [122, 137]	Fault Classification [138, 139]
	Fault Classification [64, 65, 67] [140, 142, 143]	Fault Classification [87]	Fault Classification [36, 58, 92] [93, 94]	Fault Classification [8, 97–99] [101–103] [104, 107, 163]	Fault Classification [10, 44, 109–111] [114–116, 118] [119, 121, 122, 137] [157–160]	Reliability Equations [36, 124, 126] [129, 130]	Reliability Equations [95, 96, 103] [106, 108] [131, 133, 134]	FIT Rate [10, 115] [121, 122]	FIT Rate [138, 139]
	SDC Rate [66, 70]	SDC Rate [83, 85]		Reliability Equations [95, 96, 103] [104, 105]	Vulnerability Factors [10, 44] [109, 110, 112, 113] [114, 116, 118] [119, 121, 122]	Vulnerability Factors [129, 130]			

— **Irradiation:** DNN is implemented on a DHA (i.e., FPGA, GPU, or TPU) placed under an irradiating facility to inject beams onto it.

Most of the works on DNNs' reliability assessment use FI methods. Therefore, we characterize three approaches of FI methods in Table 1. In each approach of FI methods, the works are distinguished based on DNN platforms. Furthermore, in each category, we elaborate on how the works determine the fault types and locations and evaluate the reliability by metrics. The details will be discussed in Section 5.1.

4.1.2 Analytical Methods. Works relying on an analytical method for estimating DNNs' reliability attempt to determine how parameters and neurons of a DNN affect the output based on the connections of neurons and layers. Therefore, they analyze the structure of DNNs and provide a model for the impact of faults on the outputs to find more critical and sensitive components in the DNN. Hence, they can evaluate the reliability of DNNs by means of vulnerability analysis derived by analyses and eliminate the complexity of simulating/emulating the faults in reliability assessment.

4.1.3 Hybrid Methods. Both fault injection and analytical methods are used in this category of works to take advantage of both. In this regard, analytical methods can provide some mathematical models in addition to a straightforward fault injection into the system for reliability evaluation, so metrics of reliability evaluation can be obtained with less complexity than extensive FI experiments and more realistic than analytical methods.

4.2 Research Trends

To address RQ1, we present the main statistics on the papers included in this study. Figure 8 shows the distribution of the 139 included papers published over the years 2017–2022. Regarding the chart of Figure 8, it can be seen that research on the topic of DNNs' reliability started in 2017 and in the following years it drew increasingly more attention and turned into an active topic of study.

Figure 9 illustrates the number of papers based on different reliability assessment methods among all identified works in this literature review. It can be observed that the majority of works use fault injection to assess the reliability of DNNs while only 10% of the works consider analytical (11 works) and hybrid analytical/FI (3 works) methods. In this regard, we present Figure 10 to illustrate the distribution of works using FI over different approaches and DNN platforms. It shows that most of the works belong to the hardware-independent platform of simulation in the software approach. Moreover, in the emulation in hardware approach, most of the works are done on the GPU platform. Hence, the figures present the trend of the research domain and the distribution of works over different methods and approaches, leading to areas where there is still room for future research.

5 CHARACTERIZATION

In this section, details of reliability assessment methods for DNNs are presented based on the categorizations in Figure 7 and Table 1. We start from FI methods, which include the majority of works. Then, analytical and hybrid methods will be discussed.

5.1 Fault Injection Methods

In FI methods of reliability assessment, once the DNN platform and fault model are determined, perturbation and system execution are performed, and the reliability is evaluated. Regarding the categorization in Table 1, the identified approaches of FI methods on DNN reliability assessment are presented in this subsection, separately. Since FI is the most frequently used method in the

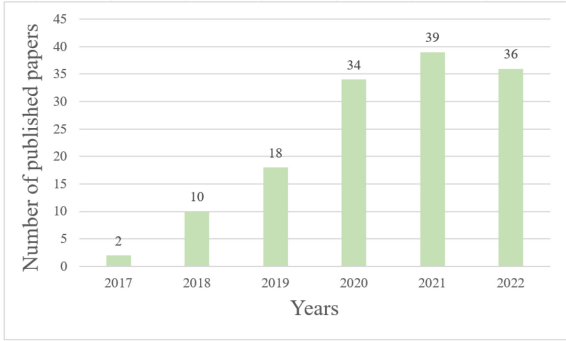


Fig. 8. Number of included papers over years.

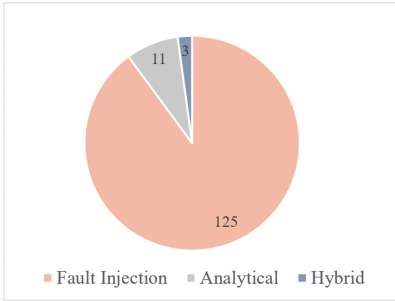


Fig. 9. Proportion of each method in the reliability assessment of DNNs among included works.

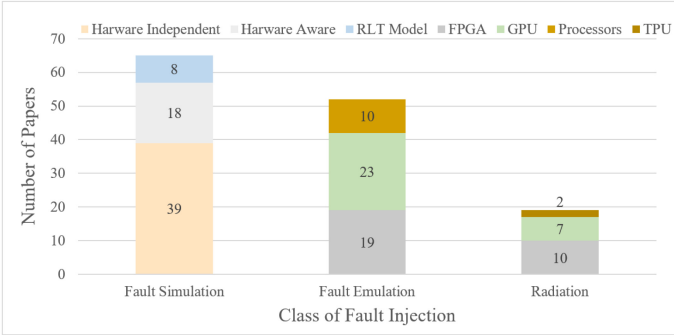


Fig. 10. Distribution of included papers over different FI approaches and platforms.

reliability assessment of DNNs, there are various presented evaluation metrics. To elaborate and distinguish different evaluation metrics, we have presented them for different approaches and platforms, separately.

5.1.1 Fault Simulation. In this subsection, the works assessing the reliability of DNNs by FI with a fault simulation approach are described. There are three platforms in this approach, i.e., hardware-independent, hardware-aware, and RTL models that are explained in the corresponding subsections.

*5.1.1.1 **Hardware-independent Platform**.* In this platform, DNNs are implemented in software DNN frameworks. Therefore, fault injection is performed on top of the frameworks, i.e., PyTorch (used in References [59, 68, 70–72, 76, 78]), Keras (used in References [61, 62, 80, 81]), TensorFlow (used in References [66, 79]), Caffe (used in Reference [77]), DarkNet (used in References [73, 140, 142]). Implementing the DNN in software provides a flexible environment for studying the effect of various fault models. As shown in the corresponding branch of Table 1, both transient and permanent faults are studied in this platform. However, most of the works studied transient faults (soft errors, SEU, MBU, etc.).

To model faults at the software level, the fault model is determined differently regarding the fault type and general aspect of DHAs. In this regard, modeling and injecting permanent faults are straightforward. They are active throughout the entire execution and set the value of a bit or variable (in weights or activations) to 0 or 1, as experimented in References [72, 140, 142]. To model transient faults, the following assumptions are considered for injecting faults into parameters, i.e.:

— DNN's parameters (e.g., weights) are stored in the memory of the accelerator. Hence, random transient faults are injected into random bits of weights as a bitflip in different executions, as experimented in References [61–65, 67–71, 73–82, 141, 143, 161].
— Faults in inputs/outputs of DNN's layers (i.e., activations) lead to the study of their impacts on both memory and logic. Activation memory faults are studied in References [72, 76], and faults in logic or datapath are investigated in References [59, 60, 64, 66, 78].

Therefore, to experiment the impact of faults on memory elements of DHAs at the software level, faults are injected into random weights and activations, and to model fault effects on logic, faults are injected into random activations. Most of the relevant works on Hardware-independent platform inject transient faults into the bits of randomly selected weights. Nearly all works in this class inject faults based on BER, determining the ratio of faulty bits throughout the values. In addition, to reach the 95% confidence level with 1% error margin, they repeat the tests several times with different random faults as in References [77, 80, 140, 142].

***Evaluation*:** For evaluating the reliability, different metrics are considered. References [59–63, 68, 69, 71–82, 141, 143] report accuracy loss under fault campaign experiments. They compare the accuracy of the faulty network with the accuracy of the fault-free network on the same test set. Some works classify the injected faults regarding the outputs of the faulty network compared with the golden model output. References [140, 142, 143] inject one permanent fault per experiment and classify them into three classes:

— **Masked:** No difference between the outputs of the faulty network and the golden model.
— **Observed-safe:** Different output of the faulty network with the golden model, while the confidence score of the top-ranked element is reduced by less than 5% with respect to the one of the golden one.
— **Observed-unsafe:** Different output of faulty network with the golden model, while the confidence score of the top-ranked element is reduced by more than 5% with respect to the one of the golden one.

Moreover, in References [65, 67] transient faults are injected into the encrypted weights of a network, and they are classified based on the effect of faults on execution of the program and results, as:

— **Silent or safe:** Similar to "masked," mentioned above in References [140, 142].
— **SDC:** Only affects the output results of the network.

— **Detected as a software exception:** Affects the execution of the program and stops it.
— **Detected by padding check action:** Corrupts the ciphertext.

Burel et al. [64] have adopted the fault classification scheme for semantic segmentation applications in which DNNs label each pixel of an input image according to a set of known classes. The corresponding classes are:

— **Masked:** Similar to "masked," mentioned above.
— **No Impact SDC:** No labels of pixels are modified.
— **Tolerable SDC:** Labels of less than 1% of pixels are modified, and no class is removed/added due to the fault.
— **Critical SDC:** Labels of more than 1% of pixels are modified or any class is removed/added due to the fault.

A specific way of fault evaluation based on fault classification is only considering the faults that affect the output as SDC, since they are critical. References [66, 70] evaluate the network based on the proportion of faults that affect the output classification results as SDC rate. Therefore, the reliability of a network can be evaluated by fault classification based on their effect on the outputs, whether by changing the output results, or by a threshold of accuracy loss, or system exceptions. This way of evaluation assists in understanding how faults would be propagated and affect the network.

***Software FI Tools*:** Some fault injectors are presented as tools that are able to support the reliability study of DNNs with different fault models in software frameworks of DNNs. PyTorchFI [164], TensorFI [165–167] and its extension TensorFI+ [168, 169], and Ares [170] inject faults into DNNs, which are implemented in PyTorch, Tensorflow, and Keras, respectively. All of these open-source frameworks can inject both permanent and transient faults into weights as well as activations with specified error rates, hence, the accuracy loss can be evaluated. TensorFI also benefits from providing the SDC rate. These frameworks are used in the reliability studies of DNNs, e.g., PyTorchFI in References [60, 70], TensorFI in Reference [66], and Ares in Reference [80].

Moreover, to enhance the efficiency of the aforementioned tools, additional fault injectors have been introduced. One such injector, known as BinFI [171], is an extension of TensorFI that aims to identify critical bits in DNNs. Another fault injector, namely, LLTFI [172], is proposed to inject transient faults into specific instructions of DNN models in either PyTorch or TensorFlow and has been found to be faster than TensorFI. Additionally, a checkpoint-based fault injector is proposed in Reference [173] that enables studying the impact of SDCs independently of the DNN implementation framework.

5.1.1.2 Hardware-aware Platform. This platform includes works that consider an abstract model of the accelerator in their implementation of DNNs in software. They implement the network in DNN software frameworks as well as high-level programming languages. Therefore, they take advantages of simulation in software fault injection while they also apply the reliability assessment to the abstract model of the accelerator.

References [83, 87] implement a DNN in Tiny-DNN and map it to the RTL implementation of the accelerator. They study the effect of transient faults in memory and datapath accurately. In these studies, FI is performed in software while all of its parameters are integrated with the corresponding hardware components. Authors in Reference [88] implement the DNN and the fault injector in software inspired by an FPGA-based DNN accelerator. Moreover, in References [9, 91], DNN and FI are implemented in Keras, and the architecture of a systolic array accelerator is considered for a fault-tolerant design. Similarly, authors in Reference [85] and [86] evaluate their proposed reliability improvement technique on memories in TensorFlow while injecting transient faults into

the weights. PyTorch is used in References [89, 90] to implement the DNN, and transient faults are injected into activations (datapath or MAC units) and weights (memory) regarding the systolic array accelerator model. Reference [84] also uses PyTorch and injects faults by a custom framework called TorchFI to inject faults into the outputs of CONV and FC layers of the network.

The effect of permanent faults at PEs' outputs is studied in References [6, 144] where the model of the accelerator is adopted from implementing the DNN in an N2D2 framework [174]. Furthermore, authors in References [145, 149] use PyTorch and study permanent faults in MAC units of an accelerator while training to improve the reliability at inference. Authors in Reference [148] have developed a Keras-based accelerator simulator to study the effect of permanent faults on the on-chip memory of accelerators by injecting permanent faults into fmaps and weights. Weight remapping strategy in memory to decrease the effect of permanent faults is evaluated in Reference [146] using Ares. SCALE-Sim [175], a systolic CNN accelerator simulator, is adopted in Reference [150] to study permanent faults in PEs and computing arrays in systolic array-based accelerators.

Similar to the Hardware-independent platform, faults are injected based on BER, or fault rate, and experiments are repeated to reach 95% confidence level and 1% error margin [9, 87, 91].

Evaluation: Nearly all works in this class evaluate the DNN by accuracy loss after fault injection [6, 9, 84, 86, 88–91, 144, 146–150]. References [83] and [85] evaluate the reliability by SDC rate as the proportion of faults that caused misclassification in comparison with the golden model. In addition, authors in Reference [87] differentiate SDCs of injected transient faults into defined classes and calculate FIT for the accelerator (*accel*) by its components (*comp*) with Equation (2) in which FIT_{raw} is provided by the manufacturer, $Size_{comp}$ is the total number of the component bits, and SDC_{comp} is obtained by FI.

$$FIT_{accel} = \sum_{comp} FIT_{raw} \times Size_{comp} \times SDC_{comp} \tag{2}$$

In addition, in this work, SDCs are classified by comparing the faulty and golden model outputs as:

- **SDC-1:** Fault caused a misclassification in the top-ranked output class.
- **SDC-5:** Fault caused the top-ranked element not to exist in the top-5 predicted output classes.
- **SDC-10%:** Fault caused a variation in the output confidence score of the top-ranked output class more than 10% compared to the golden model.
- **SDC-20%:** Fault caused a variation in the output confidence score of the top-ranked output class more than 20% compared to the golden model.

5.1.1.3 RTL Model Platform. Research works that leverage the RTL model of ASIC-based DHAs and simulate fault injections are described in the following. We identify three groups of FI experiments in this platform, divided based on the architecture of DHAs:

- 2D systolic array accelerators [7, 93, 151–154],
- RTL implementation of DNNs [94]
- **Multi-Processor System-on-Chips (MPSoCs)** for DNNs, [58].

In the first group, a configuration of TPU is utilized in References [7, 93, 153, 154], and a model of a 2D systolic array is implemented in References [151, 152]. Reference [7] also uses Eyeriss [176] architecture for the accelerator. In this group, FI is performed at RTL, and all works inject random permanent faults into PEs/MACs of the arrays, except Reference [93], which injects random transient faults into buffers, control and data registers.

The second group, which includes Reference [94], implements DNNs in RTL to enable a fault simulation study in approximated DNNs. In this work, SEU injected into Look-Up Tables are simulated and studied.

In the third group, which exploits MPSoCs, faults are emulated in the components of the target multicore processor. Authors in Reference [58] propose a three-level pipeline FI framework that simulates permanent faults in the hardware model of an MPSoC and evaluate the reliability at the software level. In their framework, the RTL model of the platform is provided as well as the fault injector unit at the lowest level. The software implementation of the DNN exists in the middle level of the framework that performs a pipelined inference and runs each layer of the network on a separate core. In the top-level of the framework, synchronization of layers and reliability evaluation is fulfilled.

Evaluation: Most works in this class evaluate the reliability by accuracy loss. Nonetheless, fault classification is performed in References [58, 93, 94]. Authors in Reference [58] adopted the classification of Reference [87], which was discussed in Hardware-aware platform (Section 5.1.1) previously. Furthermore, they added two more classes for the faults that cause Hang (the HDL simulation never finishes) and Crash (the HDL simulation immediately stops). Authors in Reference [94] classify the faults similar to the general fault classification scheme (Masked, SDC, crash) with different terminology.

In addition, Reference [93] classifies SDCs on how they impact classification outputs compared with the golden model:

— **Tolerable Misclassification:** The input is misclassified the same as the golden model with different output confidence scores,
— **No Impact Misclassification:** The input is misclassified in both golden and faulty models but into different classes,
— **Critical Misclassification:** The input is correctly classified in the golden model but misclassified in the faulty model,
— **Tolerable Correct Classification:** The input is correctly classified in both golden and faulty models with different output confidence scores,
— **Beneficial Correct Classification:** The input is misclassified in the golden model but correctly classified in the faulty model.

5.1.2 Fault Emulation. In this subsection, research works that assess the reliability of DNNs by emulating FI in hardware accelerators are explored. FPGA and GPU platforms are described, respectively.

5.1.2.1 FPGA Platform. DNNs are implemented fully or partially (e.g., one layer) on FPGAs to perform the inference phase as described in Section 2.2, and faults are being emulated on different locations of the accelerator. In most of the works on the FPGA platform, the fault injector unit is implemented in software that is run on a processor, and faults are injected into the FPGA running the DNN under analysis. This HW/SW co-design process benefits from the high-performance execution of DNNs and fast fault injection. It is worth mentioning that some works implement only a part of the DNN (e.g., one specific layer) on the FPGA [97, 98, 108].

In this group of works, Zynq-based architecture **System-on-Chips (SoCs)** [177], which take advantage of an ARM processor co-existing with the FPGA, are deployed. We categorize this group of studies into three classes:

— A host computer (e.g., a PC) initializes the faults [97–99, 107, 108],
— The on-board embedded processor initializes the faults [8, 95, 100–106, 162, 163],
— Fault injection module resides inside the hardware design implementation [96, 155, 156].

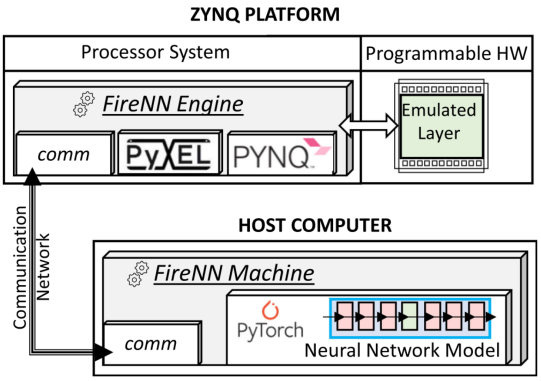


Fig. 11. An overview of the architecture of the FireNN platform [97, 98].

In the first class, faults are generated by a host computer of the accelerator design. Then, the faults, network parameters, and FPGA configuration bits will be sent to the board. The FPGA starts running, and the on-board processor collects the results. The on-board processor plays the role of a controller between FPGA and the host computer. In the end, the results would be passed back to the host computer for further processing and reliability evaluation. All works of this class emulate transient faults (SEU) in configuration bits of the FPGA and exploit the accuracy loss of the DNN for reliability evaluation. Nevertheless, authors in Reference [107] explore transient faults in **Flip Flops (FFs)** exhaustively beside random transient faults in configuration memory and classify them as tolerable, critical, and crashes.

FireNN is proposed in References [97, 98] as a platform for deploying DNNs on Zynq-based architecture SoCs along with a host computer in a way that DNN is run partially on the FPGA to perform a reliability evaluation. As shown in Figure 11, *FireNN machine* runs the neural network and communicates with the *FireNN engine* for reliability evaluation of the layer under analysis running on the FPGA. Faults are generated by the host computer and are injected to the FPGA through the engine. This platform injects SEUs in weights, layer inputs, and configuration bits.

In the second class, faults are generated and injected into the FPGA's configuration bits or on-chip memories by the embedded processor. The embedded processor or a host computer is responsible for the reliability evaluation. The proposed method in References [162, 163] provides an injection of permanent faults into the configuration bits of the FPGA as well as into the on-chip memory blocks through the interfaces between the embedded processor and FPGA on Zynq SoC. References [95, 103, 104] provide a similar design to inject transient faults into configuration bits of the FPGA. The effects of transient faults into both on-chip memories and configuration bits of an FPGA running pruned DNNs are studied in Reference [100]. Authors in Reference [95] provide random-accumulated FI and exhaustive FI approach on the configuration bits to emulate neutron and ionizing radiation. Moreover, permanent and transient faults in on-chip memory (HyperRAM) are studied in References [105, 106] with a software emulator and are validated by radiation results.

It is worth mentioning that injecting faults into the configuration memory is a repetitive process, where in each experiment of FI, the faulty configuration bits are loaded to the configuration memory. Then, the system is run and the results are collected. Thereafter, the next fault(s) are injected into the fault-free configuration bits loaded to the corresponding memory to analyze the newly injected fault(s).

A framework named Fiji-FIN is proposed in Reference [102], and the underlying method is also used in References [8, 101]. This framework is capable of injecting transient faults into both

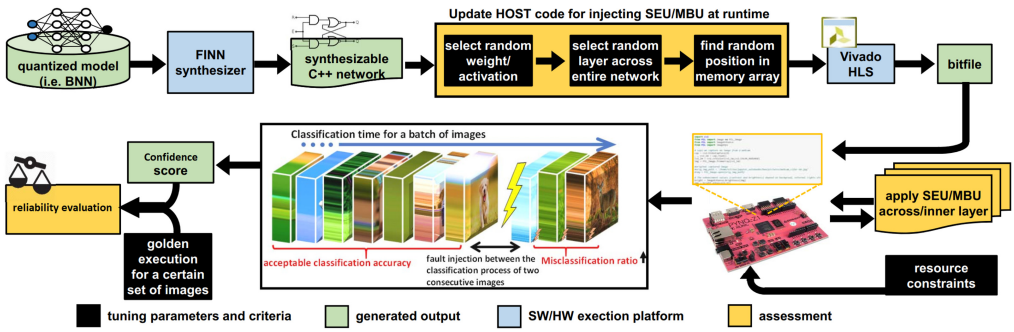


Fig. 12. Fiji-FIN framework for fault injection into FPGAs [102].

configuration bits of FPGA and on-chip memories. In this method, FINN framework [178] is used to develop and train the BNN, and the proposed framework manipulates the FINN's output to prepare it for the fault injection. The bit stream file of the FPGA is obtained by an HLS tool and imported to the FPGA. While the system is running, the faults are generated and injected by the embedded processor and the reliability is evaluated in comparison with the golden model. Figure 12 depicts in detail the steps of this FI framework.

In the third class, References [155] and [156] inject permanent faults, and the work in Reference [96] injects transient faults into the hardware implementation of the network. Authors in Reference [155] use the FINN framework to implement the QNN with 2-bit weights and activations, and a block has been added into the hardware design that is deployed for injecting stuck-at faults into the output of PEs. Reference [156] injects permanent faults into the registers of the RTL model of the network. Authors in Reference [96] explore the effect of transient faults to the configuration bits of FPGAs in which different accelerator architectures (Softcore FGPU and ZynqNet HLS) are implemented.

Evaluation: For evaluating the reliability of DNNs on the FPGA platform, accuracy loss is exploited in References [8, 100–102, 106, 108, 155, 156, 162, 163]. Moreover, fault classification is also performed in References [8, 97–99, 101, 103, 104, 163]. References [103, 104] classify SEUs in configuration bits of the FPGA as critical if a fault caused misclassification with respect to the golden model; otherwise, the fault is tolerable. In addition, Benign Errors are considered in Reference [104], which are the faults that caused true classification of the inputs that were misclassified in the golden model. Another fault classification is presented in References [97, 98] that does not only consider critical and tolerable faults but also categorizes the faults that prevent the accelerator from generating the classification output. In this regard, the effect of faults on the system performance degradation is the criterion for classifying faults in Reference [99].

Reliability is evaluated by different metrics considering accuracy loss regarding the application of the target networks in References [162, 163]. These works consider top-5 and top-1 accuracy loss for image and audio classification tasks, respectively. For object detection, **mean Average Precision (mAP)**, and for image generation, **Structural Similarity Index (SSIM)** is adopted. Regarding the adopted metrics for accuracy loss in each network, the faults are classified into three classes with different ranges of accuracy loss ($\leq$1%, 1%$\sim$5%, $\geq$5%) caused by FI. In addition, they categorize the faults that are caused by a system exception that may delay or terminate processes.

To characterize the status of DNN layers' vulnerability, authors in Reference [8] classify the parameters of layers (i.e., weights and activations) separately by performing FI. In this work, parameters of layers are labeled as Low-risk, Medium-risk, and High-risk if FI process into the target layers' parameters results in less than 1%, 1%$\sim$5%, and more than 5% accuracy loss, respectively.

The metric AVF (defined in Section 2.3) is adopted in References [103, 104] and expresses the probability of fault propagating to the output. These works obtain the AVF through the FI by dividing the number of faults propagated to the output by the total number of injected faults. Furthermore, authors in Reference [104] provide a formula to estimate the cross-section (defined in Section 2.3) of the configuration memory in Equation (3), where the obtained *AVF* by FI is multiplied by the number of bits utilized by the design times the cross-section of bits of the configuration memory. This calculation can lead to further reliability metrics that authors present in Reference [104].

$$\sigma = AVF \times (\#UtilizedBits) \times \left(\frac{\sigma_{static}}{\#MemBits}\right) \tag{3}$$

In this regard, Reference [105] estimates the SER of HyperRam saving the weights similar to Equation (3) based on the extracted information from radiation experiment reports. By providing the rate of faults likely to occur in the memory, they inject faults into the weights of CNN on an FPGA accelerator.

Moreover, Reference [95] expressed the reliability of the neural network with n layers ($L_1, L_2, \ldots, L_n$) that are implemented serially as different modules on the FPGA, as an exponential distribution in Equation (4).

$$R_{NN}(t) = e^{-(\lambda_{L_1}+\lambda_{L_2}+\cdots+\lambda_{L_n})t}, \tag{4}$$

where $\lambda = \frac{1}{MTTF}$ **(MTTF (Mean Time to Failure))**.

5.1.2.2 GPU Platform. In this subsection, we explore FI in DNNs in which faults are emulated and injected into the GPU. Nearly all works on this platform have studied the effect of transient faults on GPUs. Permanent faults are studied in References [137, 157–160, 179]. To perform FI on GPUs, researchers adopt an FI framework on GPUs; except in References [117, 137], which implemented their own FI process on CUDA and TensorRT [180], respectively. FI frameworks in GPUs including FlexGripPlus [181], NVBitFI [182], and CAROL-FI [183] are used in References [113–116, 120, 157], and [122], respectively. Nonetheless, an FI framework is proposed in Reference [179] adapting and customizing NVBitFI for studying permanent faults in GPUs and is leveraged in References [158–160]. Moreover, a cross-layer fault injector framework CLASSES is presented in Reference [184] to inject SEUs at the architecture level, enabling study of the corresponding fault effects in Reference [112]. In all works, the rate of injected faults and the number of experiments in the target locations varies and depends on the confidence level and error margin, as mentioned in References [10, 44, 109, 121, 122].

SASSIFI [185] is the most frequently used framework for FI into GPUs running DNNs, which is used in References [10, 44, 109–111, 118, 119, 121]. This framework is developed by NVIDIA to conduct fault injections and is a powerful framework with different fault models covering various locations of GPUs and provides extensive reliability evaluation metrics. The studies that use SASSIFI for fault injection investigate the effect of transient faults with SASSIFI's bit-flip model into the **ISA (Instruction Set Architecture)** visible states, including general-purpose registers, memory values' predicate registers, and condition registers in a single or multiple thread.

***Evaluation*:** Reliability evaluation of DNNs in GPUs is carried out more extensively than in other platforms. Nearly all works have classified injected faults [10, 44, 109–111, 114–116, 118, 119, 121, 122, 137, 157, 159, 160]. The general model for classifying faults in the mentioned works is as follows:

— **Masked:** Fault does not affect the output,
— **SDC:** Output confidence score differs from that of the golden model,
— **DUE:** The program hangs or the system reboots (also called *Crash* in References [10, 121]).

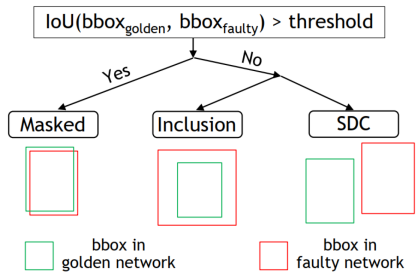


Fig. 13. Fault classification in the object detection task based on bounding boxes [137].

Furthermore, SDC is also categorized regarding the effect of faults on the accuracy of the DNN for the object recognition task in References [44, 109]. They define three categories of SDCs based on the effect of faults on the output confidence score and ranking of objects:

— **Non-critical:** Output confidence score changed, and no misclassification occurred and no objects ranking modified,
— **Light-critical:** Objects ranking modified, and no misclassification occurred,
— **Critical:** Impacted the output confidence score and caused misclassification.

However, the fault classification of SDCs proposed in Reference [122] is beyond the classic SDCs and is based on the impact of faults on the precision and recall for object detection tasks in a self-driving car, as follows:

— **Non-critical**: Precision maintains larger than 90% (a new object is detected that is not in the original classification) and recall remains 100% (all previous objects are detected).
— **Critical:** Precision is lower than 90% (many wrong objects detected) and recall is not 100% (real objects are not detected).

Furthermore, new classes of faults are presented in Reference [137], which considers the margins of the bounding box in the DNN for object detection. The authors compare the overlaps of the bounding box of the detected objects in each image for golden and faulty models and categorize the SDCs based on a threshold. Their fault classification method is depicted in Figure 13.

Vulnerability factors are also adopted to analyze the reliability of DNNs on GPU platform [10, 44, 109, 110, 114, 118, 119, 121, 122]. Vulnerability factors express the probability of propagating faults from a particular component to the output. Since faults may be injected into different locations, the vulnerability factor of the location (in different abstraction levels from architecture to program) can be measured. In this regard, **Kernel Vulnerability Factor (KVF)** [109, 118], **Layer Vulnerability Factor (LVF)** [109, 112, 118], **Instruction Vulnerability Factor (IVF)** [109, 110, 119], **Program Vulnerability Factor (PVF)** [10, 44, 109, 121], Operation Vulnerability Factor [116], and **Architecture Vulnerability Factor (AVF)** [10, 44, 109, 113, 114, 121, 122] have been presented. These metrics provide a thorough understanding of the vulnerability of each location either in DNN or in GPU.

5.1.2.3 Processors Platform. DNNs exploit processors mostly for IoT and edge applications. The research works in which faults are emulated on multi-core processors running DNNs are reviewed in this subsection. Soft errors in the register file of ARM processors running DNNs have been studied extensively in References [36, 92, 123–130]. The vulnerability of instructions is studied in Reference [130]. To emulate faults modeling soft errors in target processors, ARM-FI is developed and adopted in References [128–130] and SOFIA [92] is exploited in

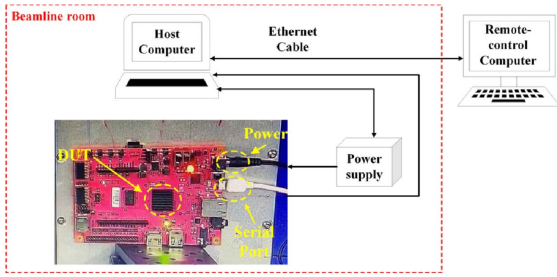


Fig. 14. Block diagram of the setup of beam experiment in Reference [108].

References [36, 92, 123–127] as fault injection frameworks. Each of the aforementioned fault injectors enables fault emulation in different components of processors.

Evaluation: All works in this class have evaluated the reliability by fault classification. The classification is performed similarly to the general scheme of classifying faults in the previous platforms (Masked, tolerable SDC, critical SDC, and DUE).

Furthermore, References [36, 92] classify the faults in an object detection task for autonomous vehicles as:

- **Incorrect probability:** All objects detected correctly with different output confidence scores,
- **Wrong detection:** Misclassification or missing an object,
- **No prediction:** No object detection.

Mean Work To Failure (MWTF) is also exploited as a reliability metric to show the amount of work a neural network can perform until meeting a failure, as:

$$MWTF = \frac{1}{executiontime \times AVF_{critical-faults}}, \tag{5}$$

where $AVF_{critical-faults}$ is the probability of an erroneous classification due to faults. MWTF is adopted as a relationship between performance and reliability in References [129, 130]. AVF is obtained as the reliability metric for the register file in References [124, 129, 130]. PVF is leveraged to express the vulnerability of operations and instructions in Reference [130].

5.1.3 Irradiation. The most realistic way of fault injection is to irradiate the devices under the beam of particles, e.g., neutron or ion. In this subsection, the research works that study the reliability of DNN accelerators, i.e., FPGA and GPU under radiation, are described.

5.1.3.1 FPGA Platform. Zynq SoCs have been examined under radiation tests to assess the reliability of DNNs in References [95, 96, 103, 106, 108, 133, 134]. FPGAs are irradiated with neutrons in References [95, 96, 103, 108, 131–133] and with protons in Reference [135]. References [132] and [135] have applied fault-aware training to DNNs and studied its impact under radiation. HyperRAM, which includes constant and dynamic variables (e.g., weights and biases) is bombarded with ionizing particles in References [106, 134]. The research works set up the configuration of the system before the experiment mostly based on HW/SW co-design and save the results for further analysis. Figure 14 shows an example of the setup of the FPGA irradiation.

Evaluation: Radiation experiments enable reliability evaluation by SER or FIT metrics [103, 106, 108, 134]. To formulate the SER, cross-section is defined as the proportion of observed faults (*errors*) over all particles collided to the surface (*Flux*), as expressed in Equation (6) [108]. Cross-section σ is expressed as a unit of cm^2 and is the probability that a particle may cause an observable

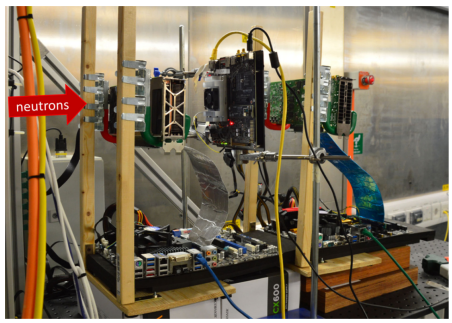


Fig. 15. Setup of neutron irradiation to GPU [10, 121, 136].

error [103]. The cross-section is exclusively adopted in References [131, 132].

$$\sigma = errors/Flux \tag{6}$$

The cross-section can lead to SER or FIT calculation by getting multiplied by the particle flux that the device will experience in the environment (ϕ). SER represents the number of failures of the device in 10^9 hours as shown in Equation (7).

$$SER = \sigma \times \phi \tag{7}$$

Most research works that study irradiation on FPGAs evaluate the reliability of devices under test by the above metrics. In addition, some works classify the faults radiated into FPGA by observing the outputs [103, 133, 135]. Here, both works provide fault classification based on output confidence scores of the neural network. Reference [103] sets up an HW/SW co-design implementation on a target board and identifies the faults causing no misclassification (tolerable) and misclassification (critical). Thereafter, the FIT of different classes of faults is obtained. References [133, 135] also present the cross-sections of the device for different classes of faults (including tolerable errors, critical errors, and crashes). Moreover, the reliability is estimated by the aforementioned metrics in Reference [95] as expressed in Equation (4).

5.1.3.2 GPU Platform. Reliability of DNNs on GPUs is assessed under neutron beam radiation in References [10, 115, 117, 121, 122, 136, 137]. All GPUs under test are manufactured by NVIDIA and have different architectures. They also provide tests by enabling and disabling ECC configurations, and different data representations. Each work has specified flux of neutrons and radiation time, e.g., Reference [137] tests the GPU equivalent to 2,000 years of exposure to terrestrial neutron, or Reference [10] reports data that cover more than 110,000 years of GPU operation. Figure 15 illustrates the radiation test setup in References [10, 121, 136].

Evaluation: Research works of this group present reliability evaluation of DNNs on GPUs by FIT as well as fault classification similar to the works on FPGAs radiation. Authors in References [10, 121] identify faults that caused SDC and Crash and report their FIT, separately. References [115] and [122] report FIT of faults caused SDC and DUE separately in different data representations of the DNN, and in Reference [137] irradiated faults are classified based on Figure 13. SDC rate is also the adopted evaluation metric in Reference [117].

5.1.3.3 TPU Platform. The reliability of Google's **Tensor Processing Unit (TPU)** is studied under neutron beam radiation in References [139] and [138]. These works experimented Coral TPU chip, a low-power accelerator for DNNs, with several neural networks for image classification and object detection tasks.

Evaluation: The research works performing radiation experiments on Coral TPU have evaluated the reliability by FIT and cross-section as well as by fault classification. In this regard, SDC and DUE fault effects are reported based on FIT and cross-section.

5.2 Analytical Methods

Analytical methods in reliability assessment model the reliability mathematically and do not inject faults into the platform to be simulated to evaluate the reliability. These methods rely on the function and algorithm of DNNs, and if needed, also consider the structure of the accelerator. Nevertheless, they carry out fault injection to assess the efficacy of the methods. For the sake of generalization, all works in this group analyze the relations of neurons and layers to find their effect and contribution to the output. In this regard, they estimate the vulnerability of neurons and analyze how a faulty neuron may impact the output to find critical neurons. Therefore, they link the reliability of the network with the vulnerability of its neurons and provide an analytical model of calculating the reliability for DNNs.

We have identified four approaches in analytical methods:

— **Layerwise Relevance Propagation (LRP)**-based analysis [186–190],
— Gradient-based analysis [191–194],
— Estimation-based analysis [192, 193, 195],
— ML-based analysis [196].

In the first approach, DNNs are analyzed based on an algorithm called Layerwise Relevance Propagation (LRP) that leads to obtaining critical scores for neurons/fmaps. The second approach is based on the gradients of weights/fmaps with respect to the output leading to their sensitivity. Research works in the third approach estimate the vulnerability of DNNs by finding correlations between some information from DNNs and the vulnerability of layers/fmaps. In the last approach, ML-based techniques are adopted in the context of fault analysis in DNNs.

In the LRP-based analysis, a hypothesis is raised in Reference [189] proposing that the higher the contribution of neurons to the DNN's output, the more impact they have on the classification accuracy. Accuracy loss is one of the most important metrics in the reliability evaluation. Therefore, the more impact a neuron has on the accuracy, the more vulnerable it is, which means it has more influence on the reliability of the network, consequently. Hence, the authors adopted the LRP algorithm to obtain the value of the contribution of each neuron to the output. LRP indicates the proportion of each connected neuron in constructing the value of the target neuron and calculates this ratio for all neurons from the last layers to the first. LRP specifies $R_{i,j}(y_0, t)$ for each neuron j in layer i, which is its output contribution score between 0 and 1 with the input y_0 and output class t. Then, the average score of each neuron over the entire training set of M inputs is obtained representing the resilience of the corresponding neuron as Equation (8).

$$r_{i,j} = \frac{M}{\sum_{m=0}^{M-1} R_{i,j}(y_{0,m}, t_m)} \tag{8}$$

Thereafter, the sorted list of neurons regarding their $r_{i,j}$ represents the most to least vulnerable neurons that can lead to protecting the most vulnerable neurons to improve reliability. Furthermore, by this analytical method, another reliability improvement method is presented in Reference [190] based on balancing the resilience distribution inside the DNN. Similarly, Reference [186] proposes an approach to extract the saliency or importance of each neuron and proposes a mapping scheme for neurons on PEs of a systolic array to minimize the score of corrupted weights.

Authors in Reference [187] extend the LRP algorithm based on different output classes of input images and provide the list of neurons' resilience scores (score maps) for individual classes

separately, as well as the score map of the whole network regardless of the output classes. Then, all sorted score maps are combined in descending order to set the maximum score for each corresponding neuron. Subsequently, a scheduling algorithm is applied to map neurons to PEs of an MPSoC based on the score maps.

In gradient-based analysis, three papers are identified. Explainable AI, which explains how the network computes the output by the input, is exploited in Reference [194] to obtain the sensitivity of layers and the importance of weights. This work defines the sensitivity of layers in compliance to the difference of the two highest output confidence scores of the last layer. Therefore, they obtain the average sensitivity of all layers and relate it to the importance of weights. They provide the most important weights and their critical bits consequently to be protected.

The sensitivity of filters and weights are analyzed in Reference [191], which refers to the amount of accuracy drop with bit-flip occurrence in weights. In the proposed method in this paper, the gradient of weights with respect to the output is calculated over a dataset considering a cost function. Also, the expectation for the probability of weights to be faulty is obtained as a noise measurement (ε_w). The sensitivity of a weight w is measured as Equation (9).

$$Sensitivity_w = gradient_w \times \varepsilon_w \tag{9}$$

Sensitivity analysis in this work leads to allocation of robust hardware to the more sensitive weights.

References [192, 193] have presented three gradient-based approaches for vulnerability estimation of fmaps in a DNN. *Gradient* approach considers the absolute values of fmaps' gradients with respect to the cross-entropy loss at the output in a backpropagation as the vulnerability of fmaps. *Gain* approach measures the noise gain by obtaining the expectation for a set of corrupted neurons affecting the DNN's accuracy based on the derivatives of outputs with respect to the neurons over a set of data and the variance of noise source. *Modified Gain* is also proposed based on the *Gain* approach to violate the independence between neurons and noise. The three mentioned approaches evaluate the vulnerability of fmaps in a DNN.

Authors in References [192, 193] also presented three estimation-based approaches for the vulnerability of fmaps. They estimate the relative fmaps' vulnerability by calculating the *max neuron value*, *fmap range*, and *average L2* over the input samples. They have provided approximate yet scalable and fast approaches to estimate the vulnerability of fmaps.

Reference [195] presents an equation to estimate the misclassification rate of CNNs in case of soft error occurrence in a specific layer. The authors consider any operation resulting in a non-zero value as a critical computation, since soft errors may corrupt their results. The estimation is based on the proportion of critical operations (*Crit_OPs*) in the target layer i and subsequent layers relative to all operations in those layers, to model the misclassification rate (*SERN*) in a CNN with n layers. Equation (10) provides a representation of this estimation.

$$SERN = \frac{Crit\_OPs_i + \sum_{i+1}^{n} OPs}{\sum_{i}^{n} OPs} \tag{10}$$

An ML-based approach for analytical reliability analysis is presented in Reference [196] where **Open-Set Recognition (OSR)** methods are explored to analyze the criticality of faults in DNNs' parameters. The concept of OSR is to identify whether the output classification corresponds to the trained classes of the DNN. This concept is adapted to analyze the output logits (output of *softmax* in the last layer) of DNNs to identify the critical fault in the parameters. Four different OSR-based methods have been leveraged for this task and their efficacies are reported. In each method, a threshold for the output logits is obtained for identifying critical fault occurrence.

All the works in this group evaluate their analytical methods on the reliability by FI. The FI methods that are used in these works are similar to the FI methods presented and characterized in Section 5.1. It is shown that analytical methods can evaluate/estimate the vulnerability/sensitivity of different components of DNNs, including neurons, fmaps, and weights. Analytical methods are more lightweight than FI by far and are accelerator-agnostic. However, their analysis results can be utilized for designing robust DNN accelerators. Among the existing approaches, estimation-based analyses are faster than others while less accurate when the results are compared with FI experiments. LRP-based and gradient analyses provide more accurate results close to FI experiments, yet they are faster and incur less complexity.

5.3 Hybrid Methods

In hybrid methods, both FI and analytical methods are carried out to assess the reliability of DNNs. To that end, Reference [197] proposes a reliability assessment framework called Fidelity based on a hybrid method. This framework studies the transient faults in both data and control path of accelerators. Fidelity contains fault injection in the software framework TensorFlow to obtain the probability of masking faults in the DNN. In addition, the framework is capable of analyzing the architectural model of the accelerator and mapping **Flip Flops (FFs)** of datapath and control logic to the parameters of a high-level implementation of the DNN. By the fault injection and elaborate analysis, it models the probability of activeness/inactiveness of FFs during the execution time as well as the probability of masking faults. Subsequently, the framework provides the *FIT rate* of the accelerator. Furthermore, the framework is validated by analyzing the NVDLA [198], i.e., an open-source NVIDIA's DNN accelerator. To further improve this method, a software model for NVDLA is proposed in Reference [199] to enable reliability study of accelerators at the software level and provide a more accurate, more hardware-aware, and faster method to obtain *FIT rate* of the accelerator.

Zhang et al. [200] propose a hybrid of ML-based analysis and FI to estimate the vulnerability of all parameters in DNNs by a low number of fault injections. The proposed method involves selecting a set of random parameters of the DNN and evaluating their vulnerabilities by injecting bitflip faults and measuring the accuracy loss. Thereafter, some features for the selected parameters (absolute value, gradient, calculation times, and layer location) are extracted. A random forest as a machine learning approach is trained and tested using the features and vulnerability of the corresponding parameters so when it reaches a high accuracy, it can be used for vulnerability estimation of the entire set of parameters.

6 DISCUSSION

In this section, we will first discuss the reliability assessment methods for DNNs based on the works reviewed and presented in Section 5. Then, we will summarize the current status in the three main categories of reliability assessment: FI, analytical, and hybrid methods, respectively, and address their pros and cons in the research domain of this literature review. Thereafter, we will present a qualitative comparison of different reliability assessment methods for DNNs. Last, we will list the open challenges as well as major potential research directions for the future.

Table 2 lists the pros and cons of all the methods categorized in this work and described in Section 5.

Of the reviewed papers, FI, as a conventional method for reliability assessment, is frequently used for evaluating the DNNs' reliability. FI provides realistic results about how faults impact the system's execution. FI methods can be conducted for modeling various faults that can be injected at the different locations in the platform for reliability evaluation. Moreover, they are applicable to any platform at any system abstraction level and provide various reliability evaluations based

Table 2. Pros and Cons of Reliability Assessment Methods for DNNs

Method	Pros	Cons
Fault Simulation	– Low design time and fast execution in high-level software implementations – Adoptable for various DNNs, DHA models, and fault models – Enabling reliability study of variations of DNNs under approximation, quantization, encryption, etc. – The availability of open-source frameworks for high-level software simulation – No need for special facilities and capable of being run on regular PCs – Enabling a fast evaluation of reliability enhancement methods at high-level software implementations – Providing various reliability evaluation metrics	– High time complexity to achieve a sufficient confidence level – Not realistic model of fault effects in high-level software implementations – Inaccurate results at high-level software implementations – Time-consuming design and development for HDL implementations
Fault Emulation	– Providing realistic reliability analysis of DHA – Enabling experiments for real conditions of DHA operation – Providing full access to possible locations of the DHA for FI – Enabling realistic studying of faults in datapath – Providing fault-tolerant designs and evaluating them directly – Providing several evaluation metrics and fault classifications	– Time-consuming design and development – Need for the physical DHA – Different platforms need their own specific design and development to perform FI – Need for platform-specific frameworks for FI
Irradiation	– Performing realistic experiments as real physical faults are injected into the chip – Suitable for developing fault models – Enabling the study for validating simulation and emulation approaches – Providing the real behavior of the DHA when faced with a physical effect	– Need for specific facilities for performing radiation – Low control over accuracy of fault injection in terms of number and locations of occurred faults – Lack of the visibility of fault propagation
Analytical	– Implementable at software-level – Scalable and less complex than FI – Leading to fault-tolerant hardware designs – Providing information for algorithm-level resiliency for DNNs – DHA-agnostic	– Not providing quantitative evaluation metrics – Not considering DHA models – Inaccurate in estimating the vulnerabilities of DNN components (neurons, fmaps, etc.)
Hybrid	– Combining fast FI with an analytical approach – Capability of reliability study for DHAs – Possibility of evaluation by either vulnerability estimation or quantitative metrics	– Need for detailed information of the DHA (depending on the method) – Accuracy of the results could be low (depending on the method)

on metrics and fault classifications. Therefore, many research works choose FI as their primary method of DNNs' reliability assessment. Nevertheless, FI methods are accompanied by a prohibitively high complexity due to the need to consider several cases for fault occurrence and to iteratively repeat the executions.

Analytical methods have been proposed as a way to cope with the high complexity of FI methods. These methods study the function of DNNs and assess the model's reliability using mathematical equations, leading to less complex approaches. Since analytical methods are developed mathematically, they have the potential to be generalized and adapted to various DNNs. Notably, analytical methods have the potential to be exploited in the reliability assessment of the training phase.

Table 3. Qualitative Analysis Comparing Different Reliability Assessment Methods for DNNs

	Fault injection	Analytical	Hybrid
Time Complexity	High	Low to Moderate	Moderate
HDA-aware	Yes	No	Yes
Leading to fault-tolerant design	Yes	Yes	Yes
Fault models variety	All fault models	Few fault models	Few fault models
Implementation system level	Software and hardware	Software	Software
Evaluation accuracy	Moderate to high	Low to moderate	Moderate
Development time	Low to Moderate	Moderate	High
Evaluation metrics	Accuracy loss Fault classification Vulnerability factors SDC rate Reliability equations	Criticality scores Sensitivity Vulnerability estimation	FIT Rate Vulnerability estimation

However, current analytical methods do not consider the accelerator models, and there is a gap in the use of reliability evaluation metrics. While this survey identifies a relatively small number of works relying on analytical methods for DNNs' reliability assessment, the future of research in this area should pay greater attention to the potential of analytical methods.

Finally, hybrid methods combine the strength of both FI and analytical methods. By applying analysis of the network or the accelerator in addition to conducting fault injection, hybrid methods are capable of obtaining a comprehensive and realistic evaluation of reliability. Although a limited number of research works are identified in this category in the present survey, yet there is a huge space to explore for proposing new hybrid methods in the future. Table 3 presents a qualitative comparison between the categorized methods of reliability assessment for DNNs regarding the papers included in this survey.

The analysis of statistics presented in Figure 9 highlights that the majority of the identified research works employ FI to assess the DNNs' reliability. This can be attributed to the fact that, while DNNs are an emerging topic in computer science, the problem of reliability has been a classic issue for a long time. In addition, the investigation of reliability over DNNs has started gaining traction since 2017, as indicated in Figure 8. As a result, it is not surprising that the early research in this area has primarily focused on conventional methods such as FI. This could be the main reason for the significant imbalance in the number of published papers across different method categories. However, in the future, the emergence of analytical and hybrid methods is expected to bridge this gap and increase their application in the field of DNN reliability assessment.

To address open challenges in reliability assessment methods for DNNs, this survey has identified the following main observations:

— Although some research works, such as Reference [201], have studied the impact of faulty data during training, no work on the reliability assessment of the training phase has been identified that considers faulty parameters or computational units. This issue should be studied in future research;
— Nearly all included works focus on CNNs, with image classification and object detection tasks excluding other types of DNNs, such as RNNs and LSTMs as well as different applications that should also be evaluated in terms of reliability;
— The survey has identified no software FI framework in hardware-aware platforms. Hence, DNN accelerator simulators could be exploited or developed for reliability assessment of DNNs in this platform;
— Fault emulation on FPGAs can take advantage of HLS designs. Therefore, a general FI framework for these platforms could be presented using HLS to minimize design time;

— Based on this survey, very few works study the reliability of the control part of DHAs, especially in FPGAs and ASICs. The control part may play a significant role in the reliability of DNN accelerators, and this should be explored in future studies;
— There is a limited number of analytical methods for DNNs reliability assessment in this survey, all of which rely on finding critical neurons for fault-tolerant designs. Also, only one work tries to predict the accuracy loss caused by soft errors, and ML-based approaches are proposed in one work. Nevertheless, none of them can estimate the reliability of DNNs on their own or evaluate the reliability using specific metrics. ML-based algorithms can significantly assist in efficient reliability assessment, and therefore, there is a huge potential for developing new analytical methods of reliability assessment for DNNs;
— Analytical methods could be generalized for other DNNs and applications rather than considering only CNNs and image processing;
— Hybrid methods appear to be powerful and capable of being exploited for developing reliability assessment frameworks. They can be one of the major methods for reliability assessment of DNNs in future works;
— Several FI research works carry out accuracy loss and fault classification as an evaluation of reliability. Also, some works considered FIT. However, there is still an urgent need to present DNN-specific metrics for reliability evaluation.

As an outcome of this survey, in addition to the listed open challenges, the major possible research directions for future studies in this domain are addressed below:

— Although analytical and hybrid methods have potential in the literature, they are not evolved to the extent that their effectiveness can be fully realized. Existing methods have shown that analytical and hybrid methods are capable of assessing the DNNs' reliability as realistically as FI and lead to effective fault-tolerant designs. Moreover, ML-based approaches in conjunction with analytical and hybrid methods are emerging. Therefore, researchers can be directed to develop novel analytical and hybrid methods, especially those that adopt ML-based algorithms, for reliability assessment of DNNs that are faster, less complex, more scalable, and more specific to DNNs than the conventional FI approaches.
— Bringing reliability as a classical issue into an emerging topic such as DNNs requires new tools to respond to the requirements of the new domain. Therefore, the new research not only needs to adopt commonly used metrics in the reliability domain, but also requires the introduction and proposal of novel DNNs-specific reliability evaluation metrics.
— There are several IoT and edge applications for DNNs emerging day by day, and reliability is not only a concern for safety-critical applications. New research can focus on the unstudied applications of DNNs while taking reliability into consideration.

7 CONCLUSION

DNNs are being utilized in an increasingly diverse range of applications in our daily lives. Consequently, their deployment in safety-critical applications has emerged to be expanding day by day. However, threats to reliability are one of the major issues that they experience in the real world. To address this, several studies have been published in recent years to assess the reliability of DNNs, with or without the use of accelerators, resulting in the development of various assessment methods. In this work, we conduct a systematic literature review to present a categorization of the reliability assessment methods for DNNs.

Out of the 139 papers related to the subject of the review, three major approaches to reliability assessment of DNNs were identified, i.e., Fault Injection, Analytical, and Hybrid methods. Since the majority of works assess the reliability using conventional fault injection methods, the related

works relying on FI methods are characterized based on different approaches and platforms. In addition, we have addressed the advantages and disadvantages of the different methods and highlighted the open challenges that may become the focus of future studies in this domain. Based on the analysis of this survey, future research could focus on developing lightweight, DNN-specific analytical and hybrid methods for assessing reliability, as well as providing new quantitative evaluation metrics that take into account emerging applications for DNNs.

REFERENCES

[1] Alberto Bosio, Ian O'Connor, Marcello Traiola, Jorge Echavarria, Jürgen Teich, Muhammad Abdullah Hanif, Muhammad Shafique, Said Hamdioui, Bastien Deveautour, Patrick Girard, Arnaud Virazel, and Koen Bertels. 2021. Emerging computing devices: Challenges and opportunities for test and reliability. In *IEEE European Test Symposium (ETS'21)*. IEEE, 1–10.

[2] Håkan Forsberg, Joakim Lindén, Johan Hjorth, Torbjörn Månefjord, and Masoud Daneshtalab. 2020. Challenges in using neural networks in safety-critical applications. In *AIAA/IEEE 39th Digital Avionics Systems Conference (DASC'20)*. IEEE, 1–7.

[3] Alessandra Nardi and Antonino Armato. 2017. Functional safety methodologies for automotive applications. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD'17)*. IEEE, 970–975.

[4] Younis Ibrahim, Haibin Wang, Junyang Liu, Jinghe Wei, Li Chen, Paolo Rech, Khalid Adam, and Gang Guo. 2020. Soft errors in DNN accelerators: A comprehensive review. *Microelectron. Reliab.* 115 (2020), 113969.

[5] Muhammad Shafique, Mahum Naseer, Theocharis Theocharides, Christos Kyrkou, Onur Mutlu, Lois Orosa, and Jungwook Choi. 2020. Robust machine learning systems: Challenges, current trends, perspectives, and the road ahead. *IEEE Des. Test* 37, 2 (2020), 30–57.

[6] Stéphane Burel, Adrian Evans, and Lorena Anghel. 2021. MOZART: Masking outputs with zeros for architectural robustness and testing of DNN accelerators. In *IEEE 27th International Symposium on On-Line Testing and Robust System Design (IOLTS'21)*. IEEE, 1–6.

[7] Krishna Teja Chitty-Venkata and Arun K. Somani. 2020. Model compression on faulty array-based neural network accelerator. In *IEEE 25th Pacific Rim International Symposium on Dependable Computing (PRDC'20)*. IEEE, 90–99.

[8] Navid Khoshavi, Arman Roohi, Connor Broyles, Saman Sargolzaei, Yu Bi, and David Z. Pan. 2020. SHIELDeNN: Online accelerated framework for fault-tolerant deep neural network architectures. In *57th ACM/IEEE Design Automation Conference (DAC'20)*. IEEE, 1–6.

[9] Elbruz Ozen and Alex Orailoglu. 2020. Low-cost error detection in deep neural network accelerators with linear algorithmic checksums. *J. Electron. Test.* 36, 6 (2020), 703–718.

[10] Fernando Fernandes dos Santos, Pedro Foletto Pimenta, Caio Lunardi, Lucas Draghetti, Luigi Carro, David Kaeli, and Paolo Rech. 2018. Analyzing and increasing the reliability of convolutional neural networks on GPUs. *IEEE Trans. Reliab.* 68, 2 (2018), 663–677.

[11] Sparsh Mittal. 2020. A survey on modeling and improving reliability of DNN algorithms and accelerators. *J. Syst. Archit.* 104 (2020), 101689.

[12] Annachiara Ruospo, Ernesto Sanchez, Lucas Matana Luza, Luigi Dilillo, Marcello Traiola, and Alberto Bosio. 2023. A survey on deep learning resilience assessment methodologies. *Computer* 56, 2 (2023), 57–66.

[13] Fei Su, Chunsheng Liu, and Haralampos-G. Stratigopoulos. 2023. Testability and dependability of AI hardware: Survey, trends, challenges, and perspectives. *IEEE Des. Test* 40, 2 (2023).

[14] Cesar Torres-Huitzil and Bernard Girau. 2017. Fault and error tolerance in neural networks: A review. *IEEE Access* 5 (2017), 17322–17341.

[15] Antonio Cicchetti, Federico Ciccozzi, and Alfonso Pierantonio. 2019. Multi-view approaches for software and system modelling: A systematic literature review. *Softw. Syst. Model.* 18, 6 (2019), 3207–3233.

[16] Mathieu Lavallée, Pierre-N. Robillard, and Reza Mirsalari. 2013. Performing systematic literature reviews with novices: An iterative approach. *IEEE Trans. Educ.* 57, 3 (2013), 175–181.

[17] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S. Emer. 2017. Efficient processing of deep neural networks: A tutorial and survey. *Proc. IEEE* 105, 12 (2017), 2295–2329.

[18] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.

[19] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. ImageNet classification with deep convolutional neural networks. *Adv. Neural Inf. Process. Syst.* 25 (2012), 1097–1105.

[20] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going deeper with convolutions. In *IEEE Conference on Computer Vision and Pattern Recognition*. 1–9.

[21] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).
[22] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*. 770–778.
[23] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. 2016. You only look once: Unified, real-time object detection. In *IEEE Conference on Computer Vision and Pattern Recognition*. 779–788.
[24] C. J. B. Yann, Y. LeCun, and C. Cortes. The MNIST DATABASE of Handwritten Digits. Retrieved from: http://yann.lecun.com/exdb/mnist/
[25] A. Krizhevsky, v. Nair, and G. Hinton. 2009. The CIFAR-10 Dataset. Retrieved from: https://www.cs.toronto.edu/~kriz/cifar.html
[26] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. ImageNet: A large-scale hierarchical image database. In *IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 248–255.
[27] Moritz Menze and Andreas Geiger. 2015. Object scene flow for autonomous vehicles. In *IEEE Conference on Computer Vision and Pattern Recognition*. 3061–3070.
[28] Mark Everingham, Luc Van Gool, Christopher K. I. Williams, John Winn, and Andrew Zisserman. 2010. The Pascal visual object classes (VOC) challenge. *Int. J. Comput. Vis.* 88, 2 (2010), 303–338.
[29] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2017. Quantized neural networks: Training neural networks with low precision weights and activations. *J. Mach. Learn. Res.* 18, 1 (2017), 6869–6898.
[30] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2016. Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or −1. *arXiv preprint arXiv:1602.02830* (2016).
[31] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. 265–283.
[32] Keras: The Python deep learning API. 2015. Retrieved from: https://keras.io/
[33] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An imperative style, high-performance deep learning library. *Adv. Neural Inf. Process. Syst.* 32 (2019), 8026–8037.
[34] Joseph Redmon. Darknet: Open Source Neural Networks in C. Retrieved from: http://pjreddie.com/darknet/
[35] Tiny-CNN Framework. 2012. Retrieved from: https://github.com/tiny-dnn/tiny-dnn
[36] Geancarlo Abich, Jonas Gava, Ricardo Reis, and Luciano Ost. 2020. Soft error reliability assessment of neural networks on resource-constrained IoT devices. In *27th IEEE International Conference on Electronics, Circuits and Systems (ICECS'20)*. IEEE, 1–4.
[37] Manar Abu Talib, Sohaib Majzoub, Qassim Nasir, and Dina Jamal. 2021. A systematic literature review on hardware implementation of artificial intelligence algorithms. *J. Supercomput.* 77 (2021), 1897–1938.
[38] Kaiyuan Guo, Shulin Zeng, Jincheng Yu, Yu Wang, and Huazhong Yang. 2019. A survey of FPGA-based neural network inference accelerators. *ACM Trans. Reconfig. Technol. Syst.* 12, 1 (2019), 1–26.
[39] Neng Hou, Xiaohu Yan, and Fazhi He. 2019. A survey on partitioning models, solution algorithms and algorithm parallelization for hardware/software co-design. *Des. Automat. Embed. Syst.* 23, 1 (2019), 57–77.
[40] Meriam Dhouibi, Ahmed Karim Ben Salem, Afef Saidi, and Slim Ben Saoud. 2021. Accelerating deep neural networks implementation: A survey. *IET Comput. Digit. Techniq.* 15, 2 (2021), 79–96.
[41] Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre-luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, C. Richard Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Daniel Killebrew, Andy Koch, Naveen Kumar, Steve Lacy, James Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire Mahony, Kieran Miller, Rahul Nagarajan, Ravi Narayanaswami, Ray Ni, Kathy Nix, Thomas Norrie, Mark Omernick, Narayana Penukonda, Andy Phelps, Jonathan Ross, Matt Ross, Amir Salek, Emad Samadiani, Chris Severn, Gregory Sizikov, Matthew Snelham, Jed Souter, Dan Steinberg, Andy Swing, Mercedes Tan, Gregory Thorson, Bo Tian, Horia Toma, Erick Tuttle, Vijay Vasudevan, Richard Walter, Walter Wang, Eric Wilcox, and Doe Hyun Yoon. 2017. In-datacenter performance analysis of a Tensor Processing Unit. In *44th Annual International Symposium on Computer Architecture*. 1–12.
[42] Diksha Moolchandani, Anshul Kumar, and Smruti R. Sarangi. 2021. Accelerating CNN inference on ASICs: A survey. *J. Syst. Archit.* 113 (2021), 101887.

[43] Jon Perez-Cerrolaza, Jaume Abella, Leonidas Kosmidis, Alejandro J. Calderon, Francisco Cazorla, and Jose Luis Flores. 2022. GPU devices for safety-critical systems: A survey. *Comput. Surv.* 55, 7 (2022), 1–37.
[44] Younis Ibrahim, Haibin Wang, Man Bai, Zhi Liu, Jianan Wang, Zhiming Yang, and Zhengming Chen. 2020. Soft error resilience of deep residual networks for object recognition. *IEEE Access* 8 (2020), 19490–19503.
[45] Liangzhen Lai, Naveen Suda, and Vikas Chandra. 2018. CMSIS-NN: Efficient neural network kernels for arm cortex-M CPUs. *arXiv preprint arXiv:1801.06601* (2018).
[46] Mohammad Saeid Mahdavinejad, Mohammadreza Rezvan, Mohammadamin Barekatain, Peyman Adibi, Payam Barnaghi, and Amit P. Sheth. 2018. Machine learning for Internet of Things data analysis: A survey. *Digit. Commun. Netw.* 4, 3 (2018), 161–175.
[47] Ramon Sanchez-Iborra and Antonio F. Skarmeta. 2020. TinyML-enabled frugal smart objects: Challenges and opportunities. *IEEE Circ. Syst. Mag.* 20, 3 (2020), 4–18.
[48] Robert C. Baumann. 2005. Radiation-induced soft errors in advanced semiconductor technologies. *IEEE Trans. Device Mater. Reliab.* 5, 3 (2005), 305–316.
[49] G. C. K. Y. Chen, K. Y. Chuah, M. F. Li, Daniel S. H. Chan, C. H. Ang, J. Z. Zheng, Y. Jin, and D. L. Kwong. 2003. Dynamic NBTI of PMOS transistors and its impact on device lifetime. In *41st IEEE International Reliability Physics Symposium*. IEEE, 196–202.
[50] Shekhar Borkar. 2005. Designing reliable systems from unreliable components: The challenges of transistor variability and degradation. *IEEE Micro* 25, 6 (2005), 10–16.
[51] Israel Koren and C. Mani Krishna. 2007. Fault-tolerant Systems. (2007).
[52] Barry Johnson. 1984. Fault-tolerant microprocessor-based systems. *IEEE Micro* 4, 06 (1984), 6–21.
[53] Arijit Biswas, Paul Racunas, Razvan Cheveresan, Joel Emer, Shubhendu S. Mukherjee, and Ram Rangan. 2005. Computing architectural vulnerability factors for address-based structures. In *32nd International Symposium on Computer Architecture (ISCA'05)*. IEEE, 532–543.
[54] Mohammad Eslami, Behnam Ghavami, Mohsen Raji, and Ali Mahani. 2020. A survey on fault injection methods of digital integrated circuits. *Integration* 71 (2020), 154–163.
[55] Annachiara Ruospo, Lucas Matana Luza, Alberto Bosio, Marcello Traiola, Luigi Dilillo, and Ernesto Sanchez. 2021. Pros and cons of fault injection approaches for the reliability assessment of deep neural networks. In *IEEE 22nd Latin American Test Symposium (LATS'21)*. IEEE, 1–5.
[56] Régis Leveugle, A. Calvez, Paolo Maistri, and Pierre Vanhauwaert. 2009. Statistical fault injection: Quantified error and confidence. In *Design, Automation & Test in Europe Conference & Exhibition*. IEEE, 502–506.
[57] Alfredo Benso and Stefano DiCarlo. 2011. The art of fault injection. *J. Contr. Eng. Appl. Inform.* 13, 4 (2011), 9–18.
[58] Annachiara Ruospo, Angelo Balaara, Alberto Bosio, and Ernesto Sanchez. 2020. A pipelined multi-level fault injector for deep neural networks. In *IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT'20)*. IEEE, 1–6.
[59] Muhammad Salman Ali, Tauhid Bin Iqbal, Kang-Ho Lee, Abdul Muqeet, Seunghyun Lee, Lokwon Kim, and Sung-Ho Bae. 2020. ERDNN: Error-resilient deep neural networks with a new error correction layer and piece-wise rectified linear unit. *IEEE Access* 8 (2020), 158702–158711.
[60] Chandramouli Amarnath, Mohamed Mejri, Kwondo Ma, and Abhijit Chatterjee. 2022. Soft error resilient deep learning systems using neuron gradient statistics. In *IEEE 28th International Symposium on On-Line Testing and Robust System Design (IOLTS'22)*. IEEE, 1–7.
[61] Austin P. Arechiga and Alan J. Michaels. 2018. The effect of weight errors on neural networks. In *IEEE 8th Annual Computing and Communication Workshop and Conference (CCWC'18)*. IEEE, 190–196.
[62] Austin P. Arechiga and Alan J. Michaels. 2018. The robustness of modern deep learning architectures against single event upset errors. In *IEEE High Performance Extreme Computing Conference (HPEC'18)*. IEEE, 1–6.
[63] Stéphane Burel, Adrian Evans, and Lorena Anghel. 2021. Zero-overhead protection for CNN weights. In *IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT'21)*. IEEE, 1–6.
[64] Stéphane BurelT, Adrian EvansT, and Lorena Anghel. 2022. Improving DNN fault tolerance in semantic segmentation applications. In *IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT'22)*. IEEE, 1–6.
[65] Riccardo Cantoro, Nikolaos I. Deligiannis, Matteo Sonza Reorda, Marcello Traiola, and Emanuele Valea. 2020. Evaluating data encryption effects on the resilience of an artificial neural network. In *IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT'20)*. IEEE, 1–4.
[66] Zitao Chen, Guanpeng Li, and Karthik Pattabiraman. 2021. A low-cost fault corrector for deep neural networks through range restriction. In *51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'21)*. IEEE, 1–13.
[67] Nikolaos Ioannis Deligiannis, Riccardo Cantoro, Matteo Sonza Reorda, Marcello Traiola, and Emanuele Valea. 2021. Towards the integration of reliability and security mechanisms to enhance the fault resilience of neural networks. *IEEE Access* 9 (2021), 155998–156012.

[68] Zhen Gao, Xiaohui Wei, Han Zhang, Wenshuo Li, Guangjun Ge, Yu Wang, and Pedro Reviriego. 2020. Reliability evaluation of pruned neural networks against errors on parameters. In *IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT'20)*. IEEE, 1–6.

[69] Behnam Ghavami, Mani Sadati, Zhenman Fang, and Lesley Shannon. 2022. FitAct: Error resilient deep neural networks via fine-grained post-trainable activation functions. In *Design, Automation & Test in Europe Conference & Exhibition (DATE'22)*. IEEE, 1239–1244.

[70] Brunno F. Goldstein, Sudarshan Srinivasan, Dipankar Das, Kunal Banerjee, Leandro Santiago, Victor C. Ferreira, Alexandre S. Nery, Sandip Kundu, and Felipe M. G. França. 2020. Reliability evaluation of compressed deep learning models. In *IEEE 11th Latin American Symposium on Circuits & Systems (LASCAS'20)*. IEEE, 1–5.

[71] Hui Guan, Lin Ning, Zhen Lin, Xipeng Shen, Huiyang Zhou, and Seung-Hwan Lim. 2019. In-place zero-space memory protection for CNN. In *33rd International Conference on Neural Information Processing Systems*. 5734–5743.

[72] Le-Ha Hoang, Muhammad Abdullah Hanif, and Muhammad Shafique. 2020. FT-ClipAct: Resilience analysis of deep neural networks and improving their fault tolerance using clipped activation. In *Design, Automation & Test in Europe Conference & Exhibition (DATE'20)*. IEEE, 1241–1246.

[73] Myeungjae Jang and Jeongkyu Hong. 2021. MATE: Memory-and retraining-free error correction for convolutional neural network weights. *J. Inf. Commun. Converg. Eng.* 19, 1 (2021), 22–28.

[74] Suyong Lee, Insu Choi, and Joon-Sung Yang. 2022. Bipolar vector classifier for fault-tolerant deep neural networks. In *59th ACM/IEEE Design Automation Conference*. 673–678.

[75] Elaheh Malekzadeh, Nezam Rohbani, Zhonghai Lu, and Masoumeh Ebrahimi. 2021. The impact of faults on DNNs: A case study. In *IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT'21)*. IEEE, 1–6.

[76] Mohamed A. Neggaz, Ihsen Alouani, Pablo R. Lorenzo, and Smail Niar. 2018. A reliability study on CNNs for critical embedded systems. In *IEEE 36th International Conference on Computer Design (ICCD'18)*. IEEE, 476–479.

[77] Mohamed A. Neggaz, Ihsen Alouani, Smail Niar, and Fadi Kurdahi. 2019. Are CNNs reliable enough for critical applications? An exploratory study. *IEEE Des. Test* 37, 2 (2019), 76–83.

[78] Elbruz Ozen and Alex Orailoglu. 2021. SNR: Squeezing numerical range defuses bit error vulnerability surface in deep neural networks. *ACM Trans. Embed. Comput. Syst.* 20, 5s (2021), 1–25.

[79] Jonathan Ponader, Kyle Thomas, Sandip Kundu, and Yan Solihin. 2021. MILR: Mathematically induced layer recovery for plaintext space error correction of CNNs. In *51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'21)*. IEEE, 75–87.

[80] Majid Sabbagh, Cheng Gongye, Yunsi Fei, and Yanzhi Wang. 2019. Evaluating fault resiliency of compressed deep neural networks. In *IEEE International Conference on Embedded Software and Systems (ICESS'19)*. IEEE, 1–7.

[81] Rizwan Tariq Syed, Markus Ulbricht, Krzysztof Piotrowski, and Milos Krstic. 2021. Fault resilience analysis of quantized deep neural networks. In *IEEE 32nd International Conference on Microelectronics (MIEL'21)*. IEEE, 275–279.

[82] Jinyu Zhan, Ruoxu Sun, Wei Jiang, Yucheng Jiang, Xunzhao Yin, and Cheng Zhuo. 2021. Improving fault tolerance for reliable DNN using boundary-aware activation. *IEEE Trans. Comput.-Aid. Des. Integ. Circ. Syst.* 41, 10 (2021), 3414–3425.

[83] Arash Azizimazreah, Yongbin Gu, Xiang Gu, and Lizhong Chen. 2018. Tolerating soft errors in deep learning accelerators with reliable on-chip memory designs. In *IEEE International Conference on Networking, Architecture and Storage (NAS'18)*. IEEE, 1–10.

[84] Brunno F. Goldstein, Victor C. Ferreira, Sudarshan Srinivasan, Dipankar Das, Alexandre S. Nery, Sandip Kundu, and Felipe M. G. França. 2021. A lightweight error-resiliency mechanism for deep neural networks. In *22nd International Symposium on Quality Electronic Design (ISQED'21)*. IEEE, 311–316.

[85] Masoomeh Jasemi, Shaahin Hessabi, and Nader Bagherzadeh. 2020. Enhancing reliability of emerging memory technology for machine learning accelerators. *IEEE Trans. Emerg. Topics Comput.* 9, 4 (2020), 2234–2240.

[86] Jae-San Kim and Joon-Sung Yang. 2019. DRIS-3: Deep neural network reliability improvement scheme in 3D die-stacked memory based on fault analysis. In *56th ACM/IEEE Design Automation Conference (DAC'19)*. IEEE, 1–6.

[87] Guanpeng Li, Siva Kumar Sastry Hari, Michael Sullivan, Timothy Tsai, Karthik Pattabiraman, Joel Emer, and Stephen W. Keckler. 2017. Understanding error propagation in deep learning neural network (DNN) accelerators and applications. In *International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–12.

[88] Wenshuo Li, Guangjun Ge, Kaiyuan Guo, Xiaoming Chen, Qi Wei, Zhen Gao, Yu Wang, and Huazhong Yang. 2020. Soft Error Mitigation for Deep Convolution Neural Network on FPGA Accelerators. In *2nd IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS'20)*. IEEE, 1–5.

[89] Elbruz Ozen and Alex Orailoglu. 2020. Boosting bit-error resilience of DNN accelerators through median feature selection. *IEEE Trans. Comput.-Aid. Des. Integ. Circ. Syst.* 39, 11 (2020), 3250–3262.

[90] Elbruz Ozen and Alex Orailoglu. 2020. Just say zero: Containing critical bit-error propagation in deep neural networks with anomalous feature suppression. In *IEEE/ACM International Conference on Computer Aided Design (ICCAD'20)*. IEEE, 1–9.

[91] Elbruz Ozen and Alex Orailoglu. 2019. Sanity-check: Boosting the reliability of safety-critical deep neural network applications. In *IEEE 28th Asian Test Symposium (ATS'19)*. IEEE, 7–75.

[92] Vitor Bandeira, Felipe Rosa, Ricardo Reis, and Luciano Ost. 2019. Non-intrusive fault injection techniques for efficient soft error vulnerability analysis. In *IFIP/IEEE 27th International Conference on Very Large Scale Integration (VLSI-SoC'19)*. IEEE, 123–128.

[93] Panayiotis Corneliou, Panagiota Nikolaou, Maria K. Michael, and Theocharis Theocharides. 2021. Fine-grained vulnerability analysis of resource constrained neural inference accelerators. In *IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT'21)*. IEEE, 1–6.

[94] Anahita Hosseinkhani and Behnam Ghavami. 2021. Improving soft error reliability of FPGA-based deep neural networks with reduced approximate TMR. In *11th International Conference on Computer Engineering and Knowledge (ICCKE'21)*. IEEE, 459–464.

[95] Fabio Benevenuti, Fabiano Libano, Vincent Pouget, Fernanda Lima Kastensmidt, and Paolo Rech. 2018. Comparative analysis of inference errors in a neural network implemented in SRAM-based FPGA induced by neutron irradiation and fault injection methods. In *31st Symposium on Integrated Circuits and Systems Design (SBCCI'18)*. IEEE, 1–6.

[96] Fabio Benevenuti, Márcio Gonçalves, Evaldo Carlos Fonseca Pereira Junior, Rafael Galhardo Vaz, Odair Lelis Gonçalez, José Rodrigo Azambuja, and Fernanda Lima Kastensmidt. 2021. Neutron-induced faults on CNN for aerial image classification on SRAM-based FPGA using softcore GPU and HLS. In *21th European Conference on Radiation and Its Effects on Components and Systems (RADECS'21)*. IEEE, 1–4.

[97] Corrado De Sio, Sarah Azimi, and Luca Sterpone. 2020. An emulation platform for evaluating the reliability of deep neural networks. In *IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT'20)*. IEEE, 1–4.

[98] Corrado De Sio, Sarah Azimi, and Luca Sterpone. 2022. FireNN: Neural networks reliability evaluation on hybrid platforms. *IEEE Trans. Emerg. Topics Comput.* 10, 2 (2022), 549–563.

[99] Boyang Du, Sarah Azimi, Corrado De Sio, Ludovica Bozzoli, and Luca Sterpone. 2019. On the reliability of convolutional neural network implementation on SRAM-based FPGA. In *IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT'19)*. IEEE, 1–6.

[100] Zhen Gao, Yi Yao, Xiaohui Wei, Tong Yan, Shulin Zeng, Guangjun Ge, Yu Wang, Anees Ullah, and Pedro Reviriego. 2022. Reliability evaluation of FPGA based pruned neural networks. *Microelectron. Reliab.* 130 (2022), 114498.

[101] Navid Khoshavi, Connor Broyles, and Yu Bi. 2020. Compression or corruption? A study on the effects of transient faults on BNN inference accelerators. In *21st International Symposium on Quality Electronic Design (ISQED'20)*. IEEE, 99–104.

[102] Navid Khoshavi, Connor Broyles, Yu Bi, and Arman Roohi. 2020. Fiji-FIN: A fault injection framework on quantized neural network inference accelerator. In *19th IEEE International Conference on Machine Learning and Applications (ICMLA'20)*. IEEE, 1139–1144.

[103] Fabiano Libano, Brittany Wilson, J. Anderson, Michael J. Wirthlin, Carlo Cazzaniga, Christopher Frost, and Paolo Rech. 2018. Selective hardening for neural networks in FPGAs. *IEEE Trans. Nuclear Sci.* 66, 1 (2018), 216–222.

[104] Fabiano Libano, Brittany Wilson, Michael Wirthlin, Paolo Rech, and John Brunhaver. 2020. Understanding the impact of quantization, accuracy, and radiation on the reliability of convolutional neural networks on FPGAs. *IEEE Trans. Nuclear Sci.* 67, 7 (2020), 1478–1484.

[105] Lucas Matana Luza, Annachiara Ruospo, Alberto Bosio, Ernesto Sanchez, and Luigi Dilillo. 2021. A model-based framework to assess the reliability of safety-critical applications. In *24th International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS'21)*. IEEE, 41–44.

[106] Lucas Matanaluza, Annachiara Ruospo, Daniel Soderstrom, Carlo Cazzaniga, Maria Kastriotou, Ernesto Sanchez, Alberto Bosio, and Luigi Dilillo. 2021. Emulating the effects of radiation-induced soft-errors for the reliability assessment of neural networks. *IEEE Trans. Emerg. Topics Comput.* 10, 4 (2021).

[107] Ioanna Souvatzoglou, Athanasios Papadimitriou, Aitzan Sari, Vasileios Vlagkoulis, and Mihalis Psarakis. 2021. Analyzing the single event upset vulnerability of binarized neural networks on SRAM FPGAs. In *IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT'21)*. IEEE, 1–6.

[108] H.-B. Wang, Y.-S. Wang, J.-H. Xiao, S.-L. Wang, and T.-J. Liang. 2021. Impact of single-event upsets on convolutional neural networks in Xilinx Zynq FPGAs. *IEEE Trans. Nuclear Sci.* 68, 4 (2021), 394–401.

[109] Khalid Adam, Izzeldin Ibrahim Mohamed, and Younis Ibrahim. 2021. A selective mitigation technique of soft errors for DNN models used in healthcare applications: DenseNet201 case study. *IEEE Access* 9 (2021), 65803–65823.

[110] Khalid Adam, Izzeldin I. Mohd, and Younis Ibrahim. 2021. Analyzing the instructions vulnerability of dense convolutional network on GPUS. *Int. J. Electric. Comput. Eng.* 11, 5 (2021), 4481–4488.

[111] Khalid Adam, Izzeldin I. Mohd, and Younis M. Younis. 2021. The impact of the soft errors in convolutional neural network on GPUs: Alexnet as case study. *Procedia Comput. Sci.* 182 (2021), 89–94.
[112] Cristiana Bolchini, Luca Cassano, Antonio Miele, and Alessandro Nazzari. 2022. Selective hardening of CNNs based on layer vulnerability estimation. In *IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT'22)*. IEEE, 1–6.
[113] Niccolò Cavagnero, Fernando Dos Santos, Marco Ciccone, Giuseppe Averta, Tatiana Tommasi, and Paolo Rech. 2022. Transient-fault-aware design and training to enhance DNNs reliability with zero-overhead. In *IEEE 28th International Symposium on On-Line Testing and Robust System Design (IOLTS'22)*. IEEE, 1–7.
[114] Josie E. Rodriguez Condia, Fernando Fernandes dos Santos, Matteo Sonza Reorda, and Paolo Rech. 2021. Combining architectural simulation and software fault injection for a fast and accurate CNNs reliability evaluation on GPUs. In *IEEE 39th VLSI Test Symposium (VTS'21)*. IEEE, 1–7.
[115] Fernando Fernandes Dos Santos, Angeliki Kritikakou, Josie E. Rodriguez Condia, Juan-David Guerrero-Balaguera, Matteo Sonza Reorda, Olivier Sentieys, and Paolo Rech. 2022. Characterizing a neutron-induced fault model for deep neural networks. *IEEE Trans. Nuclear Sci.* 70, 4 (2022).
[116] Tyler Garrett and Alan D. George. 2021. Improving dependability of onboard deep learning with resilient TensorFlow. In *IEEE Space Computing Conference (SCC'21)*. IEEE, 134–142.
[117] Siva Kumar Sastry Hari, Michael Sullivan, Timothy Tsai, and Stephen W. Keckler. 2021. Making convolutions resilient via algorithm-based error detection techniques. *IEEE Trans. Depend. Sec. Comput.* 19, 4 (2021).
[118] Younis Ibrahim, Haibin Wang, and Khalid Adam. 2020. Analyzing the reliability of convolutional neural networks on GPUs: GoogLeNet as a case study. In *International Conference on Computing and Information Technology (ICCIT'20)*. IEEE, 1–6.
[119] Younis Ibrahin, Junyang Liu, Xuanxuan Yang, Hongwei Sha, and Haibin Wang. 2020. Analyzing the impact of soft errors in deep neural networks on GPUs from instruction level. *WSEAS Trans. Syst. Contr.* 15 (2020), 699–708.
[120] Rubens Luiz Rech and Paolo Rech. 2020. Impact of layers selective approximation on CNNs reliability and performance. In *IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT'20)*. IEEE, 1–4.
[121] Fernando Fernandes dos Santos, Lucas Draghetti, Lucas Weigel, Luigi Carro, Philippe Navaux, and Paolo Rech. 2017. Evaluation and mitigation of soft-errors in neural network-based object detection in three GPU architectures. In *47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W'17)*. IEEE, 169–176.
[122] Fernando Fernandes dos Santos, Philippe Navaux, Luigi Carro, and Paolo Rech. 2019. Impact of reduced precision in the reliability of deep neural networks for object detection. In *IEEE European Test Symposium (ETS'19)*. IEEE, 1–6.
[123] Geancarlo Abich, Ricardo Reis, and Luciano Ost. 2021. The impact of precision bitwidth on the soft error reliability of the MobileNet network. In *IEEE 12th Latin America Symposium on Circuits and System (LASCAS'21)*. IEEE, 1–4.
[124] Geancarlo Abich, Jonas Gava, Rafael Garibotti, Ricardo Reis, and Luciano Ost. 2021. Applying lightweight soft error mitigation techniques to embedded mixed precision deep neural networks. *IEEE Trans. Circ. Syst. I: Reg. Pap.* 68, 11 (2021), 4772–4782.
[125] Geancarlo Abich, Rafael Garibotti, Jonas Gava, Ricardo Reis, and Luciano Ost. 2022. Impact of thread parallelism on the soft error reliability of convolution neural networks. In *IEEE 13th Latin America Symposium on Circuits and System (LASCAS'22)*. IEEE, 1–4.
[126] Geancarlo Abich, Rafael Garibotti, Ricardo Reis, and Luciano Ost. 2022. The impact of soft errors in memory units of edge devices executing convolutional neural networks. *IEEE Trans. Circ. Syst. II: Express Briefs* 69, 3 (2022), 679–683.
[127] Jonas Gava, Guilherme Dorneles, Ricardo Reis, Rafael Garibotti, and Luciano Ost. 2022. Soft error assessment of CNN inference models running on a RISC-V processor. In *29th IEEE International Conference on Electronics, Circuits and Systems (ICECS'22)*. IEEE, 1–4.
[128] Zhi Liu, Zhen Deng, and Xinni Yang. 2022. Using checksum to improve the reliability of embedded convolutional neural networks. *Microelectron. Reliab.* 136 (2022), 114666.
[129] Zhi Liu and Xinni Yang. 2022. An efficient structure to improve the reliability of deep neural networks on ARMs. *Microelectron. Reliab.* 136 (2022), 114729.
[130] Zhi Liu, Yuhong Liu, Zhengming Chen, Gang Guo, and Haibin Wang. 2021. Analyzing and increasing soft error resilience of Deep Neural Networks on ARM processors. *Microelectron. Reliab.* 124 (2021), 114331.
[131] Dimitris Agiakatsikas, Nikos Foutris, Aitzan Sari, Vasileios Vlagkoulis, Ioanna Souvatzoglou, Mihalis Psarakis, Mikel Luján, Maria Kastriotou, and Carlo Cazzaniga. 2021. Evaluation of the Xilinx deep learning processing unit under neutron irradiation. In *21st European Conference on Radiation and Its Effects on Components and Systems (RADECS'21)*. IEEE, 1–4.
[132] Giulio Gambardella, Nicholas J. Fraser, Ussama Zahid, Gianluca Furano, and Michaela Blott. 2022. Accelerated radiation test on quantized neural networks trained with fault aware training. In *IEEE Aerospace Conference (AERO'22)*. IEEE, 1–7.

[133] F. Libano, P. Rech, B. Neuman, J. Leavitt, M. Wirthlin, and J. Brunhaver. 2021. How reduced data precision and degree of parallelism impact the reliability of convolutional neural networks on FPGAs. *IEEE Trans. Nuclear Sci.* 68, 5 (2021), 865–872.

[134] Lucas Matana Luza, Daniel Söderström, Georgios Tsiligiannis, Helmut Puchner, Carlo Cazzaniga, Ernesto Sanchez, Alberto Bosio, and Luigi Dilillo. 2020. Investigating the impact of radiation-induced soft errors on the reliability of approximate computing systems. In *IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT'20)*. IEEE, 1–6.

[135] Pierre Maillard, Yanran P. Chen, Jason Vidmar, Nicholas Fraser, Giulio Gambardella, Minal Sawant, and Martin L. Voogel. 2022. Radiation tolerant deep learning processor unit (DPU) based platform using Xilinx 20nm Kintex UltraScale™ FPGA. *IEEE Trans. Nuclear Sci.* 70, 4 (2022).

[136] Pedro Martins Basso, Fernando Fernandes dos Santos, and Paolo Rech. 2020. Impact of tensor cores and mixed precision on the reliability of matrix multiplication in GPUs. *IEEE Trans. Nuclear Sci.* 67, 7 (2020), 1560–1565.

[137] Atieh Lotfi, Saurabh Hukerikar, Keshav Balasubramanian, Paul Racunas, Nirmal Saxena, Richard Bramley, and Yanxiang Huang. 2019. Resiliency of automotive object detection networks on GPU architectures. In *IEEE International Test Conference (ITC'19)*. IEEE, 1–9.

[138] Rubens Luiz Rech Junior, Sujit Malde, Carlo Cazzaniga, Maria Kastriotou, Manon Letiche, Christopher Frost, and Paolo Rech. 2022. High energy and thermal neutron sensitivity of Google Tensor Processing Units. *IEEE Trans. Nuclear Sci.* 69, 3 (2022), 567–575.

[139] Rubens Luiz Rech and Paolo Rech. 2022. Reliability of Google's tTensor Processing Units for embedded applications. In *Design, Automation & Test in Europe Conference & Exhibition (DATE'22)*. IEEE, 376–381.

[140] Alberto Bosio, Paolo Bernardi, Annachiara Ruospo, and Ernesto Sanchez. 2019. A reliability analysis of a deep neural network. In *IEEE Latin American Test Symposium (LATS'19)*. IEEE, 1–6.

[141] Seo-Seok Lee and Joon-Sung Yang. 2022. Value-aware parity insertion ECC for fault-tolerant deep neural network. In *Design, Automation & Test in Europe Conference & Exhibition (DATE'22)*. IEEE, 724–729.

[142] Annachiara Ruospo, Alberto Bosio, Alessandro Ianne, and Ernesto Sanchez. 2020. Evaluating convolutional neural networks reliability depending on their data representation. In *23rd Euromicro Conference on Digital System Design (DSD'20)*. IEEE, 672–679.

[143] Annachiara Ruospo, Ernesto Sanchez, Marcello Traiola, Ian O'connor, and Alberto Bosio. 2021. Investigating data representation for efficient and reliable convolutional neural networks. *Microprocess. Microsyst.* 86 (2021), 104318.

[144] Stephane Burel, Adrian Evans, and Lorena Anghel. 2022. Mozart+: Masking outputs with zeros for improved architectural robustness and testing of DNN accelerators. *IEEE Trans. Device Mater. Reliab.* 22, 2 (2022), 120–128.

[145] Le-Ha Hoang, Muhammad Abdullah Hanif, and Muhammad Shafique. 2021. TRe-Map: Towards reducing the overheads of fault-aware retraining of deep neural networks by merging fault maps. In *24th Euromicro Conference on Digital System Design (DSD'21)*. IEEE, 434–441.

[146] Thai-Hoang Nguyen, Muhammad Imran, Jaehyuk Choi, and Joon-Sung Yang. 2021. Low-cost and effective fault-tolerance enhancement techniques for emerging memories-based deep neural networks. In *58th ACM/IEEE Design Automation Conference (DAC'21)*. IEEE, 1075–1080.

[147] Ayesha Siddique, Kanad Basu, and Khaza Anuarul Hoque. 2021. Exploring fault-energy trade-offs in approximate DNN hardware accelerators. In *22nd International Symposium on Quality Electronic Design (ISQED'21)*. IEEE, 343–348.

[148] Yung-Yu Tsai and Jin-Fu Li. 2021. Evaluating the impact of fault-tolerance capability of deep neural networks caused by faults. In *IEEE 34th International System-on-Chip Conference (SOCC'21)*. IEEE, 272–277.

[149] Ussama Zahid, Giulio Gambardella, Nicholas J. Fraser, Michaela Blott, and Kees Vissers. 2020. FAT: Training neural networks for reliable inference under hardware faults. In *IEEE International Test Conference (ITC'20)*. IEEE, 1–10.

[150] Yingnan Zhao, Ke Wang, and Ahmed Louri. 2022. FSA: An efficient fault-tolerant systolic array-based DNN accelerator architecture. In *IEEE 40th International Conference on Computer Design (ICCD'22)*. IEEE, 545–552.

[151] Cheng Liu, Cheng Chu, Dawen Xu, Ying Wang, Qianlong Wang, Huawei Li, Xiaowei Li, and Kwang-Ting Cheng. 2021. HyCA: A hybrid computing architecture for fault-tolerant deep learning. *IEEE Trans. Comput.-aid. Des. Integ. Circ. Syst.* 41, 10 (2021), 3400–3413.

[152] Dawen Xu, Cheng Chu, Qianlong Wang, Cheng Liu, Ying Wang, Lei Zhang, Huaguo Liang, and Kwang-Ting Cheng. 2020. A hybrid computing architecture for fault-tolerant deep learning accelerators. In *IEEE 38th International Conference on Computer Design (ICCD'20)*. IEEE, 478–485.

[153] Jeff Jun Zhang, Kanad Basu, and Siddharth Garg. 2019. Fault-tolerant systolic array based accelerators for deep neural network execution. *IEEE Des. Test* 36, 5 (2019), 44–53.

[154] Jeff Jun Zhang, Tianyu Gu, Kanad Basu, and Siddharth Garg. 2018. Analyzing and mitigating the impact of permanent faults on a systolic array based neural network accelerator. In *IEEE 36th VLSI Test Symposium (VTS'18)*. IEEE, 1–6.

[155] Giulio Gambardella, Johannes Kappauf, Michaela Blott, Christoph Doehring, Martin Kumm, Peter Zipf, and Kees Vissers. 2019. Efficient error-tolerant quantized neural network accelerators. In *IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT'19)*. IEEE, 1–6.

[156] Behzad Salami, Osman S. Unsal, and Adrian Cristal Kestelman. 2018. On the resilience of RTL NN accelerators: Fault characterization and mitigation. In *30th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD'18)*. IEEE, 322–329.

[157] Josie E. Rodriguez Condia, Juan-David Guerrero-Balaguera, Fernando F. Dos Santos, Matteo Sonza Reorda, and Paolo Rech. 2022. A multi-level approach to evaluate the impact of GPU permanent faults on CNN's reliability. In *IEEE International Test Conference (ITC'22)*. IEEE, 278–287.

[158] Juan-David Guerrero-Balaguera, Josie E. Rodriguez Condia, and Matteo Sonza Reorda. 2022. Neural network's reliability to permanent faults: Analyzing the impact of performance optimizations in GPUs. In *29th IEEE International Conference on Electronics, Circuits and Systems (ICECS'22)*. IEEE, 1–4.

[159] Juan-David Guerrero-Balaguera, Robert Limas Sierra, and Matteo Sonza Reorda. 2022. Effective fault simulation of GPU's permanent faults for reliability estimation of CNNs. In *IEEE 28th International Symposium on On-Line Testing and Robust System Design (IOLTS'22)*. IEEE, 1–6.

[160] Juan-David Guerrero-Balaguera, Luigi Galasso, Robert Limas Sierra, Ernesto Sanchez, and Matteo Sonza Reorda. 2022. Evaluating the impact of permanent faults in a GPU running a deep neural network. In *IEEE International Test Conference in Asia (ITC-Asia'22)*. IEEE, 96–101.

[161] Zheyu Yan, Yiyu Shi, Wang Liao, Masanori Hashimoto, Xichuan Zhou, and Cheng Zhuo. 2020. When single event upset meets deep neural networks: Observations, explorations, and remedies. In *25th Asia and South Pacific Design Automation Conference (ASP-DAC'20)*. IEEE, 163–168.

[162] Dawen Xu, Ziyang Zhu, Cheng Liu, Ying Wang, Huawei Li, Lei Zhang, and Kwang-Ting Cheng. 2020. Persistent fault analysis of neural networks on FPGA-based acceleration system. In *IEEE 31st International Conference on Application-specific Systems, Architectures and Processors (ASAP'20)*. IEEE, 85–92.

[163] Dawen Xu, Ziyang Zhu, Cheng Liu, Ying Wang, Shuang Zhao, Lei Zhang, Huaguo Liang, Huawei Li, and Kwang-Ting Cheng. 2021. Reliability evaluation and analysis of FPGA-based neural network acceleration system. *IEEE Trans. Very Large Scale Integ. Syst.* 29, 3 (2021), 472–484.

[164] Abdulrahman Mahmoud, Neeraj Aggarwal, Alex Nobbe, Jose Rodrigo Sanchez Vicarte, Sarita V. Adve, Christopher W. Fletcher, Iuri Frosio, and Siva Kumar Sastry Hari. 2020. PyTorchFi: A runtime perturbation tool for DNNs. In *50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W'20)*. IEEE, 25–31.

[165] Zitao Chen, Niranjhana Narayanan, Bo Fang, Guanpeng Li, Karthik Pattabiraman, and Nathan DeBardeleben. 2020. TensorFI: A flexible fault injection framework for TensorFlow applications. In *IEEE 31st International Symposium on Software Reliability Engineering (ISSRE'20)*. IEEE, 426–435.

[166] Guanpeng Li, Karthik Pattabiraman, and Nathan DeBardeleben. 2018. TensorFI: A configurable fault injector for TensorFlow applications. In *IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW'18)*. IEEE, 313–320.

[167] Niranjhana Narayanan, Zitao Chen, Bo Fang, Guanpeng Li, Karthik Pattabiraman, and Nathan Debardeleben. 2022. Fault injection for TensorFlow applications. *IEEE Trans. Depend. Sec. Comput.* 20, 4 (2022).

[168] Sabuj Laskar, Md Hasanur Rahman, and Guanpeng Li. 2022. TensorFI+: A scalable fault injection framework for modern deep learning neural networks. In *IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW'22)*. IEEE, 246–251.

[169] Sabuj Laskar, Md Hasanur Rahman, Bohan Zhang, and Guanpeng Li. 2022. Characterizing deep learning neural network failures between algorithmic inaccuracy and transient hardware faults. In *IEEE 27th Pacific Rim International Symposium on Dependable Computing (PRDC'22)*. IEEE, 54–67.

[170] Brandon Reagen, Udit Gupta, Lillian Pentecost, Paul Whatmough, Sae Kyu Lee, Niamh Mulholland, David Brooks, and Gu-Yeon Wei. 2018. Ares: A framework for quantifying the resilience of deep neural networks. In *55th ACM/ESDA/IEEE Design Automation Conference (DAC'18)*. IEEE, 1–6.

[171] Zitao Chen, Guanpeng Li, Karthik Pattabiraman, and Nathan DeBardeleben. 2019. BinFI: An efficient fault injector for safety-critical machine learning systems. In *International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–23.

[172] Udit Kumar Agarwal, Abraham Chan, and Karthik Pattabiraman. 2022. LLTFI: Framework agnostic fault injection for machine learning applications (tools and artifact track). In *IEEE 33rd International Symposium on Software Reliability Engineering (ISSRE'22)*. IEEE, 286–296.

[173] Elvis Rojas, Diego Pérez, Jon C. Calhoun, Leonardo Bautista Gomez, Terry Jones, and Esteban Meneses. 2021. Understanding soft error sensitivity of deep learning models and frameworks through checkpoint alteration. In *IEEE International Conference on Cluster Computing (CLUSTER'21)*. IEEE, 492–503.

[174] N2D2 CAD framework for DNNs. 2016. Retrieved from: https://github.com/cea-list/N2D2

[175] Ananda Samajdar, Yuhao Zhu, Paul Whatmough, Matthew Mattina, and Tushar Krishna. 2018. SCALE-Sim: Systolic CNN accelerator simulator. *arXiv preprint arXiv:1811.02883* (2018).

[176] Yu-Hsin Chen, Joel Emer, and Vivienne Sze. 2016. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. *ACM SIGARCH Comput. Archit. News* 44, 3 (2016), 367–379.

[177] XILINX. 2021. SoCs with Hardware and Software Programmability. Retrieved from: https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html

[178] Yaman Umuroglu, Nicholas J. Fraser, Giulio Gambardella, Michaela Blott, Philip Leong, Magnus Jahre, and Kees Vissers. 2017. FINN: A framework for fast, scalable binarized neural network inference. In *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. 65–74.

[179] Juan-David Guerrero-Balaguera, Luigi Galasso, Robert Limas Sierra, and Matteo Sonza Reorda. 2022. Reliability assessment of neural networks in GPUs: A framework for permanent faults injections. In *IEEE 31st International Symposium on Industrial Electronics (ISIE'22)*. IEEE, 959–962.

[180] NVIDIA Corporation. 2021. NVIDIA TensorRT. Retrieved from: https://developer.nvidia.com/tensorrt

[181] Josie E. Rodriguez Condia, Boyang Du, Matteo Sonza Reorda, and Luca Sterpone. 2020. FlexGripPlus: An improved GPGPU model to support reliability analysis. *Microelectron. Reliab.* 109 (2020), 113660.

[182] Timothy Tsai, Siva Kumar Sastry Hari, Michael Sullivan, Oreste Villa, and Stephen W. Keckler. 2021. NVBitFI: Dynamic fault injection for GPUs. In *51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'21)*. IEEE, 284–291.

[183] Daniel Oliveira, Laércio Pilla, Nathan DeBardeleben, Sean Blanchard, Heather Quinn, Israel Koren, Philippe Navaux, and Paolo Rech. 2017. Experimental and analytical study of Xeon Phi reliability. In *International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–12.

[184] Cristiana Bolchini, Luca Cassano, Antonio Miele, and Alessandro Toschi. 2022. Fast and accurate error simulation for CNNs against soft errors. *IEEE Trans. Comput.* 72, 4 (2022).

[185] Siva Kumar Sastry Hari, Timothy Tsai, Mark Stephenson, Stephen W. Keckler, and Joel Emer. 2017. SASSIFI: An architecture-level fault injection tool for GPU application resilience evaluation. In *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS'17)*. IEEE, 249–258.

[186] Muhammad Abdullah Hanif and Muhammad Shafique. 2020. SalvageDNN: Salvaging deep neural network accelerators with permanent faults through saliency-driven fault-aware mapping. *Philosoph. Trans. Roy. Societ. A* 378, 2164 (2020), 20190164.

[187] Annachiara Ruospo and Ernesto Sanchez. 2021. On the reliability assessment of artificial neural networks running on AI-oriented MPSoCs. *Appl. Sci.* 11, 14 (2021), 6455.

[188] Annachiara Ruospo, Gabriele Gavarini, Ilaria Bragaglia, Marcello Traiola, Alberto Bosio, and Ernesto Sanchez. 2022. Selective hardening of critical neurons in deep neural networks. In *25th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS'22)*. IEEE, 136–141.

[189] Christoph Schorn, Andre Guntoro, and Gerd Ascheid. 2018. Accurate neuron resilience prediction for a flexible reliability management in neural network accelerators. In *Design, Automation & Test in Europe Conference & Exhibition (DATE'18)*. IEEE, 979–984.

[190] Christoph Schorn, Andre Guntoro, and Gerd Ascheid. 2019. An efficient bit-flip resilience optimization method for deep neural networks. In *Design, Automation & Test in Europe Conference & Exhibition (DATE'19)*. IEEE, 1507–1512.

[191] Wonseok Choi, Dongyeob Shin, Jongsun Park, and Swaroop Ghosh. 2019. Sensitivity based error resilient techniques for energy efficient deep neural network accelerators. In *56th Annual Design Automation Conference*. 1–6.

[192] Abdulrahman Mahmoud, Siva Kumar Sastry Hari, Christopher W. Fletcher, Sarita V. Adve, Charbel Sakr, Naresh Shanbhag, Pavlo Molchanov, Michael B. Sullivan, Timothy Tsai, and Stephen W. Keckler. 2020. HarDNN: Feature map vulnerability evaluation in CNNs. *arXiv preprint arXiv:2002.09786* (2020).

[193] Abdulrahman Mahmoud, Siva Kumar Sastry Hari, Christopher W. Fletcher, Sarita V. Adve, Charbel Sakr, Naresh R. Shanbhag, Pavlo Molchanov, Michael B. Sullivan, Timothy Tsai, and Stephen W. Keckler. 2021. Optimizing selective protection for CNN resilience. In *International Symposium on Software Reliability Engineering (ISSRE'21)*. 127–138.

[194] Muhammad Sabih, Frank Hannig, and Jürgen Teich. 2021. Fault-tolerant low-precision DNNs using explainable AI. In *51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W'21)*. IEEE, 166–174.

[195] Liqi Ping, Jingweijia Tan, and Kaige Yan. 2020. SERN: Modeling and analyzing the soft error reliability of convolutional neural networks. In *Great Lakes Symposium on VLSI*. 445–450.

[196] G. Gavarini, D. Stucchi, A. Ruospo, G. Boracchi, and E. Sanchez. 2022. Open-set recognition: An inexpensive strategy to increase DNN reliability. In *IEEE 28th International Symposium on On-Line Testing and Robust System Design (IOLTS'22)*. IEEE, 1–7.

[197] Yi He, Prasanna Balaprakash, and Yanjing Li. 2020. Fidelity: Efficient resilience analysis framework for deep learning accelerators. In *53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'20)*. IEEE, 270–281.

[198] NVIDIA Corporation. 2021. NVDLA Open Source Project. Retrieved from: http://nvdla.org
[199] Alessandro Veronesi, Francesco Dall'Occo, Davide Bertozzi, Michele Favalli, and Milos Krstic. 2022. Exploring software models for the resilience analysis of deep learning accelerators: The NVDLA case study. In *25th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS'22)*. IEEE, 142–147.
[200] Yangchao Zhang, Hiroaki Itsuji, Takumi Uezono, Tadanobu Toba, and Masanori Hashimoto. 2022. Estimating vulnerability of all model parameters in DNN with a small number of fault injections. In *Design, Automation & Test in Europe Conference & Exhibition (DATE'22)*. IEEE, 60–63.
[201] Abraham Chan, Arpan Gujarati, Karthik Pattabiraman, and Sathish Gopalakrishnan. 2022. The fault in our data stars: studying mitigation techniques against faulty training data in machine learning applications. In *52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'22)*. IEEE, 163–171.

Received 8 May 2023; accepted 15 December 2023

Appendix 10

X

M. H. Ahmadilivani, M. Taheri, J. Raik, M. Daneshtalab, and M. Jenihhin, "Enhancing Fault Resilience of QNNs by Selective Neuron Splitting," in 2023 IEEE 5th International Conference on Artificial Intelligence Circuits and Systems (AICAS), pp. 1–5, 2023.

Enhancing Fault Resilience of QNNs by Selective Neuron Splitting

Mohammad Hasan Ahmadilivani[1], Mahdi Taheri[1], Jaan Raik[1], Masoud Daneshtalab[1,2], and Maksim Jenihhin [1]

[1]Tallinn University of Technology, Tallinn, Estonia
[2]Mälardalen University, Västerås, Sweden
[1]{mohammad.ahmadilivani, mahdi.taheri, jaan.raik, maksim.jenihhin}@taltech.ee
[2]masoud.daneshtalab@mdu.se

2023 IEEE 5th International Conference on Artificial Intelligence Circuits and Systems (AICAS) | 979-8-3503-3267-4/23/$31.00 ©2023 IEEE | DOI: 10.1109/AICAS57966.2023.10168633

***Abstract*—The superior performance of Deep Neural Networks (DNNs) has led to their application in various aspects of human life. Safety-critical applications are no exception and impose rigorous reliability requirements on DNNs. Quantized Neural Networks (QNNs) have emerged to tackle the complexity of DNN accelerators, however, they are more prone to reliability issues.**

In this paper, a recent analytical resilience assessment method is adapted for QNNs to identify critical neurons based on a Neuron Vulnerability Factor (NVF). Thereafter, a novel method for splitting the critical neurons is proposed that enables the design of a Lightweight Correction Unit (LCU) in the accelerator without redesigning its computational part.

The method is validated by experiments on different QNNs and datasets. The results demonstrate that the proposed method for correcting the faults has a twice smaller overhead than a selective Triple Modular Redundancy (TMR) while achieving a similar level of fault resiliency.

I. Introduction

Artificial Intelligence (AI) has shifted the paradigm of computer science in the latest decade with Deep Neural Networks (DNNs), one of AI's illustrious instruments, demonstrating remarkable precision levels [1]. This has led to their adoption in several safety-critical applications like autonomous driving [2]. As DNN accelerators become more prevalent in safety-critical applications, hardware reliability of digital circuits has become increasingly more noticeable. The reliability of DNNs is determined by the ability of their accelerators to function correctly [3] in the presence of environment-related faults (soft errors, electromagnetic effects, temperature variations) or faults in the underlying hardware (manufacturing defects, process variations, aging effects) [4].

Various emerging techniques are explored to improve the computational efficiency of DNNs' complex architectures, such as reducing the bit precision of parameters, which has led to the emergence of Quantized Neural Networks (QNNs). However, the effectiveness of such techniques raises concerns about the reliability of QNNs, particularly in safety-critical applications. Soft errors, a type of fault caused by charged particles colliding with transistors, can cause a logic value to flip, dramatically influencing the functionality of QNNs [5], [6].

Throughout the literature, protecting DNNs against soft errors is primarily achieved through architecture-level methods such as hardened PEs or Triple Modular Redundancy (TMR) [7]. However, to alleviate overheads, there is a need, first, to identify the critical neurons within a neural network before applying the mentioned mitigation techniques to harden them against the faults.

Reliability assessment serves as the initial step towards exploiting an effective protection mechanism. Fault Injection (FI) is a conventional method for reliability assessment that is vastly adopted for DNNs. However, identifying the critical points in a QNN requires an exhaustive FI that is prohibitively complex due to their large number of parameters. To address this issue, analytical resilience assessment approaches are proposed to evaluate the reliability of DNNs by analyzing them at the algorithm level [8].

In previous works, the criticality of neurons has been identified based on their contribution scores to outputs [9]–[12]. Hence, there is no clear resilience evaluation metric for selecting the critical neurons in the literature, and recent works extract the criticality based on the ranked scores. To tackle the drawbacks of the state-of-the-art in DNNs' resilience analysis methods, a prior study has proposed a method called DeepVigor [13], which provides vulnerability factors for all bits, neurons, and layers of DNNs accurately. However, it does not consider QNNs. In this work, we adapt and optimize DeepVigor for identifying critical neurons in QNNs. The resilience analysis enables us to design a method for correcting soft errors in the datapath of DNN accelerators.

In this paper, we identify critical neurons in QNNs based on a Neuron Vulnerability Factor (NVF) obtained by fault propagation analysis through the QNNs. The NVF represents the probability of misclassification due to a fault in a neuron which determines the level of criticality for neurons. To the best of our knowledge, for the first time, a protection technique based on splitting neurons' operations is proposed that modifies the network in a way that a Lightweight Correction Unit (LCU) corrects the faults in critical neurons. The proposed method does not require redesigning the computational part of the accelerator. The accelerator executes the modified network, and only its controller needs to be aware of the critical neurons to be operated on the LCU. Our method imposes half the overhead of TMR since it corrects faults with only one additional neuron instead of two.

The contributions of this work are as follows:

- Developing an analytical fault resilience assessment method for QNNs to identify the most critical neurons based on the conducted Neuron Vulnerability Factor (NVF);
- Proposing a novel high-level modification method for QNNs to improve fault resiliency by splitting the operations of critical neurons, without requiring a redesign of the computational part of the accelerator;
- Designing an effective Lightweight Correction Unit (LCU) for selected critical neurons in accelerators, with low overhead (twice less than that of TMR) and high fault resiliency (similar to that of TMR).

The paper is organized as follows. The proposed method for enhancing fault resilience of QNNs is presented in Section II, experiments are performed and discussed in Section III, and the paper is concluded in Section IV.

The work is supported in part by the EU through European Social Fund in the frames of the "ICT programme" ("ITA-IoIT" topic), by the Estonian Research Council grant PUT PRG1467 "CRASHLESS", Estonian Centre for Research Excellence EXCITE and by Estonian-French PARROT project "EnTrustED".

II. Method For Resilience Enhancement of QNNs

A. Accelerator Model

Fig. 1 illustrates the accelerator model considered in this work which is inspired by [14]. It consists of a computational part (an array of Processing Elements (PEs), activation functions, pooling, and normalization), buffers for parameters (weight and bias), inputs, and outputs, and the controller. It is assumed that faults may happen in the computational part of the accelerator, thus, the *Outputs Buffer* may contain faulty values of output activations. The controller is responsible for feeding the inputs, transferring the outputs, and controlling the function of the accelerator.

To apply the resilience enhancement method to accelerator, a Lightweight Correction Unit (LCU) is added to the design in which the controller only needs to be aware of the critical neurons. Once the outputs of a layer are calculated, the controller transfers the critical neurons to LCU, replaces its corrected outputs back to the *Outputs Buffer*, and continues the operations of the accelerator. The design of the LCU is proposed in Subsection II-C.

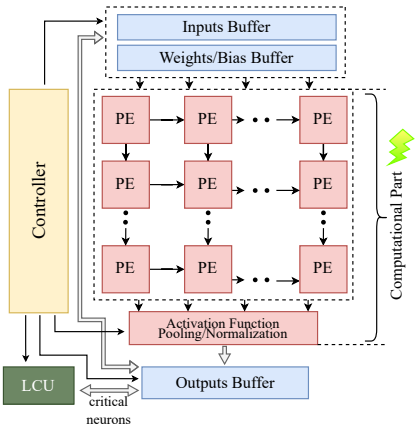


Fig. 1: An abstract view of the accelerator and where the faults may happen.

B. Identifying Critical Neurons by Resilience Analysis

Algorithm 1 presents the resilience analysis of QNNs to obtain Neuron Vulnerability Factors (NVF) for all neurons throughout the QNN in convolution and fully-connected layers. It is assumed that the neural network is quantized into an 8-bit signed integer data type, and the output activation of the neuron is analyzed. The algorithm, first, checks whether or not to analyze an input for the neuron (lines 3-5) by the gradients of a loss function ($\mathcal{L}$) that represents the impact of the neuron's erroneous output on the golden top class of the network.

Then, it finds minimum positive and maximum negative values for the neuron (δ), that cause a misclassification in the QNN from its golden output (lines 6, 7). Thereafter, it maps the obtained δ to a corresponding possible bitflip location in the data type (lines 8, 9) and counts it as a vulnerable location (lines 10, 11). In the end, regarding the counted of vulnerable times for each bit, it calculates the probability of misclassification of the network by each bitflip in the output of the neuron as the NVF over the whole inputs (line 15).

A key observation in the analysis is that the *0 to 1* bitflip is much more critical than *1 to 0* bitflip. Because the former enlarges the values in the activation and propagates to the output, while the latter is masked. This observation leads us to the protection mechanism proposed in the next Subsection. It is worth mentioning that the resilience analysis method is not limited to a single-bit flip fault model, and it implicitly considers multi-bit faults.

By obtaining the NVF of all neurons through the QNN, the critical neurons can be found based on the values for NVF. Different thresholds can be set to select the critical neurons and protect them, considering how many of them are affected by the protection techniques leading to execution overheads.

Algorithm 1 Resilience Analysis of QNNs

Input: Trained QNN with a set of neurons Q and N outputs, set of input images X;

Output: NVF of all neurons;

Assume: $\delta \in$ [-128,127]; $\mathcal{E}_{c_t}$ is the output score for the golden top class; C_g is golden classification; C_δ is classification result after injecting δ; *vul_map_arr_pos* and *vul_map_arr_neg* include counters for each bit corresponds to each vulnerability range for positive and negative numbers;

1: **for** $neuron \in$ Q **do**:
2: **for** $input \in X$ **do**:
3: $\mathcal{L} = sigmoid(\sum_{j=0}^{N}(\mathcal{E}_{c_t} - \mathcal{E}_{c_j}))$
4: $grad = \nabla\mathcal{L}/out_{neuron}$
5: **if** $grad$!= 0 **then**
6: $r_{upper} = min(\delta), \delta > 0, s.t.\ C_g \neq C_f$
7: $r_{lower} = max(\delta), \delta < 0, s.t.\ Cl_g \neq Cl_f$
8: bit_{upper} = int($\sqrt{r_{upper}}$) + 1;
9: bit_{lower} = int($\sqrt{|r_{lower}|}$);
10: $vul\_map\_arr\_pos[bit_{upper}]$++;
11: $vul\_map\_arr\_neg[bit_{lower}]$++;
12: **end if**;
13: **end for**;
14: $vul\_map\_arr = (vul\_map\_arr\_pos + vul\_map\_arr\_neg)\ /\ 2$
15: $NVF_{neuron} = \frac{\sum_{i=1}^{8}(\frac{1}{8}\times\sum_{j=1}^{i}(vul\_map\_arr[j]))}{size(X)}$
16: **end for**;

C. Resilience Enhancement by Splitting Critical Neurons and LCU

The proposed fault resilience enhancement targets the critical neurons identified based on a threshold on NVF. The idea is to split the selected neurons' operation into two neurons in the QNN at a high level and correct the critical outputs in the accelerator. Fig. 2 depicts how a critical neuron is split into two halves. As it is shown, the input parameters (weights and bias) of the neuron are halved, keeping the output parameters non-modified, and the new neurons are replaced with the critical neuron in the QNN. In this way, the neuron can be split into two neurons without changing the intermediate values of the further layers and the neural network's outputs. Noteworthy, the method is applied to all identified critical neurons in convolution and fully-connected layers.

Splitting the critical neurons provides an opportunity for fault correction using the split neurons without redesigning the

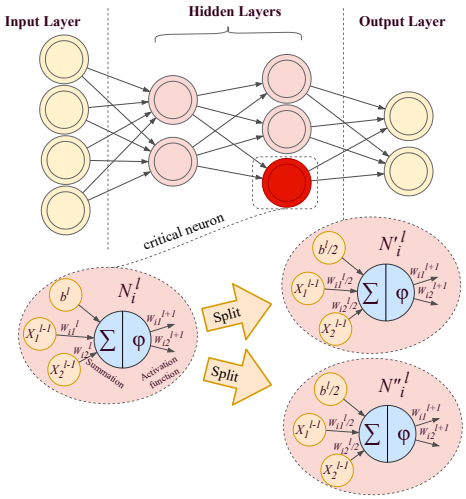


Fig. 2: Operation splitting for a neuron in a QNN involves halving the input parameters while keeping the output parameters non-modified. A critical neuron is replaced with its corresponding split neurons in the QNN.

computational part of the accelerator. The network is modified in a way that the selected critical neurons from the analysis are split. The modified network can then be mapped to the accelerator using the existing controller and mapping algorithm of the accelerator. However, the controller needs to be aware of the critical neurons so that it can transfer them to LCU to perform the correction and write them back to the *Output Buffers* (Fig. 1).

LCU is designed to leverage the neuron-splitting method for correction. The inputs of LCU are two split neurons representing one critical neuron, and the output is one corrected 8-bit data that will be written back to the corresponding neurons.

The data type (signed integer 8-bit) contains one sign bit and 7 bits for the integer. As the neuron's operation is split, the range of output values for each replaced neuron would be divided by 2. Therefore, the Most Significant Bit (MSB) in the integer part of the output should always be 0. Regarding the observation in the analysis about bitflips (Subsection II-B), any faulty bit can be set to zero to be less critical.

Therefore, to output the corrected value, LCU performs two operations: 1) a bit-wise AND over the two inputs, 2) resets the MSB of the integer part to 0. In this way, many single and also multiple faults that occur to the bits will be masked by these two operations. Since the correction operations are merely an *AND* and a *bit reset*, the correction unit is *lightweight*. The operation of the LCU correction is depicted in Fig. 3 performing on the faulty outputs of PEs running two splits of a critical neuron. The corrected output is written back to *Outputs Buffer* as the outputs of the corresponding PEs.

III. Experiments

A. Experimental Setup

The experimented QNNs in this work are fully quantized (all parameters and activation) to 8-bit signed integer using TFLite [15]. The experiments in this work have been performed on a 7-layer MLP and LeNet-5 trained on MNIST as well as an AlexNet

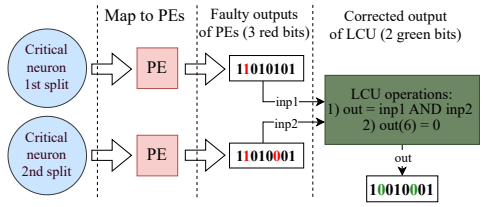


Fig. 3: An example of how LCU corrects faulty critical neurons.

trained on CIFAR-10. The baseline accuracy of each network on the test data is 70.1%, 89.1%, and 62.9%, respectively.

The resilience analysis and enhancement (Sections II-B and II-C) are implemented in PyTorch considering the accelerator model. The resilience analysis is conducted over the training set. The critical neurons regarding different thresholds for NVF are obtained to explore the number of neurons to be protected, which imposes an overhead as well.

To show the efficacy of the resilience enhancement method, a statistical FI is performed. In the FI process, one single bitflip in the output of a random neuron in the network is injected, and whole inference over the test set is performed, and the overall accuracy is obtained. To meet the 95% confidence level with a 1% error margin in the statistical FI based on [16], we repeated the FI process for each MLP-7, LeNet-5, and AlexNet for 6,750, 7,650, and 9,500 random faults, respectively.

As a baseline comparison of the proposed design for LCU, we also apply a TMR to the critical neurons for the detection and correction of faults. We adopt two metrics for comparing the results of methods and expressing the resiliency: 1) accuracy loss of QNNs over the fault injection, 2) the portion of critical faults in a fault injection campaign. Critical faults are the ones that misclassify the network from its golden classification.

B. Experimental Results

1) An Exploration on NVF of QNNs

As mentioned, NVF explores the probability of a faulty neuron's output that misclassifies the QNN from its golden output. Table I presents the number of critical neurons in different NVFs ranging from 0% (all neurons are critical) to 50% (no neuron is critical). According to the table, different thresholds of NVF count a different portion of neurons as critical among QNNs. However, it is observed that all neurons among QNNs have NVF of less than 50%. It is noteworthy that a higher threshold for NVF means a less number of critical neurons to be protected. This table represents the overhead of any protection mechanism over the critical neurons.

Table I: Exploration of number and portion of critical neurons over different thresholds for NVF.

QNN	MLP-7		LeNet-5		AlexNet	
NVF threshold	#neurons	portion	#neurons	portion	#neurons	portion
NVF >= 0%	2816	100%	4684	100%	103168	100%
NVF >= 5%	2513	89.24%	4380	93.5%	46322	44.9%
NVF >= 10%	1382	49.07%	1659	35.41%	15818	15.33%
NVF >= 15%	903	32.06%	222	4.74%	5171	5.01%
NVF >= **20%**	**503**	**17.86%**	**187**	**3.99%**	**622**	**0.6%**
NVF >= 25%	272	9.6%	70	1.49%	398	0.38%
NVF >= 30%	184	6.5%	3	0.06%	232	0.2%
NVF >= 35%	85	3.01%	0	0%	147	0.14%
NVF >= 40%	26	0.92%	0	0%	56	0.05%
NVF >= 45%	7	0.2%	0	0%	6	0.005%
NVF >= 50%	0	0%	0	0%	0	0%

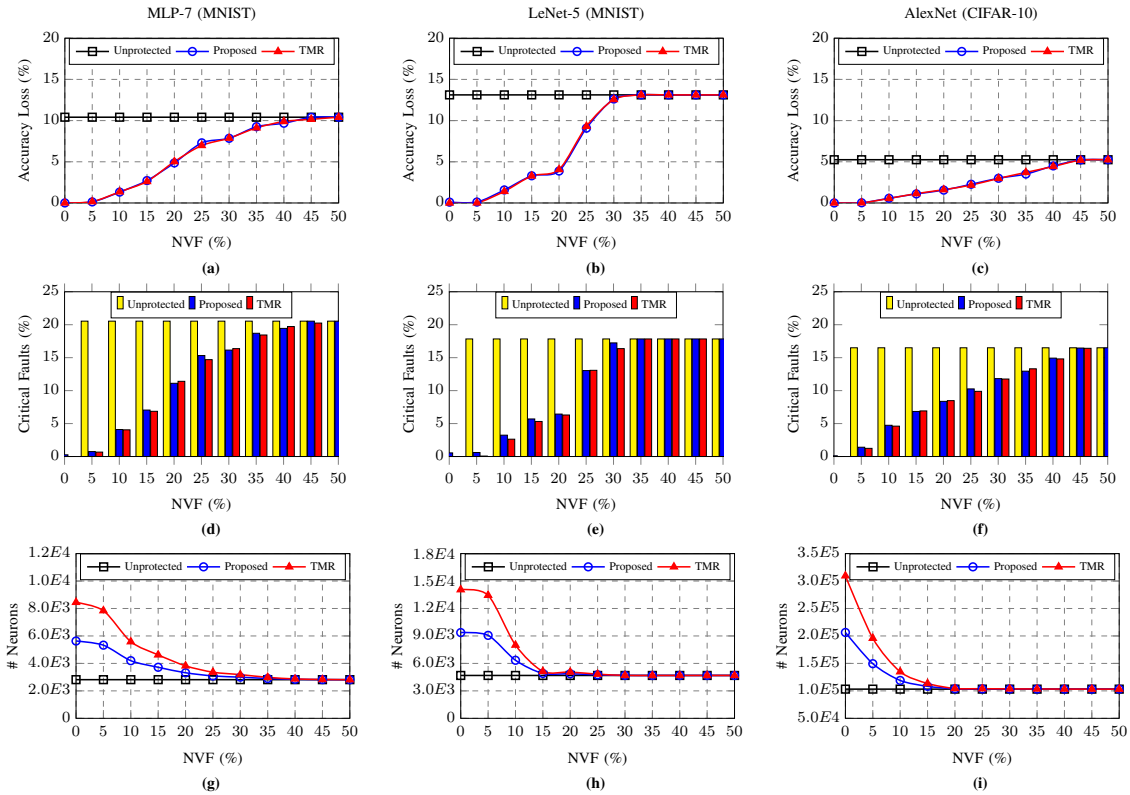


Fig. 4: QNNs comparison in terms of accuracy loss (a-c), critical faults (d-f), and network size (g-i) under different levels of protection: unprotected, proposed protection, and TMR, considering different thresholds for NVF from 0% to 50%.

2) Resilience Enhancement of QNNs

Fig. 4 illustrates the experimental results of accuracy loss (a-c) and critical faults (d-f) of the proposed resilience enhancement and TMR over different NVF thresholds for the QNNs. The results show how critical neurons are effectively selected and protected by the proposed method. As shown, all results of protecting QNNs by the proposed method are very close to those of selective TMR-based protection. Furthermore, Fig. 4-(g-i) shows that the QNNs' size (as measured by the number of neurons in each network) using the proposed protection is remarkably smaller than that of the TMR-based protected networks, resulting in half the overhead due to employing one additional neuron for correction instead of two.

Assuming a constraint on the accuracy loss to be less than 5% in Fig. 4, a common NVF for all three QNNs can be considered as 20% in which the accuracy loss is 4.86%, 3.88%, and 1.56% in the QNNs protected by the proposed method that is 2.14x, 3.38x, and 3.36x less than the unprotected QNNs, respectively. Regarding Table I, the resilience analysis suggests protecting 17.86% of neurons in MLP-7, 3.99% of neurons in LeNet-5, and 0.6% of neurons in AlexNet, respectively. The proposed protection mechanism results in 1.85x, 2.78x, and 1.97x fewer critical faults than unprotected QNNs in the MLP-7, LeNet-5, and AlexNet, respectively.

The proposed neuron splitting and correction method leverages only two neurons (one additional) for correcting faults, whereas TMR requires three neurons (two additional) to perform fault detection and correction. As a result, the overhead of the proposed method is significantly lower than that of TMR, while providing similar resilience. According to Table I, to protect QNNs with an NVF of 20% using TMR, quantized MLP-7, LeNet-5, and AlexNet require 3,822, 5,058, and 104,412 neurons, respectively, whereas the proposed method requires only 3,319, 4,871, and 103,790 neurons, respectively. Therefore, the proposed method reduces the overall size of QNNs by 15.15%, 3.84%, and 0.6% compared to TMR-based protection, which impacts the memory footprint and execution time of the accelerator accordingly.

IV. Conclusion

This paper proposes a QNN fault resilience enhancement method. It is achieved by a fault resilience analysis method for QNNs based on the computation of the vulnerability factor for all neurons of a QNN. A neuron splitting method is introduced to modify the network in a way that the critical neurons selected by the resilience analysis are split into two halves. This method enables us to design a Lightweight Correction Unit (LCU) within the accelerator without redesigning its computational parts. The results indicate that the proposed method significantly enhances the fault resiliency of QNNs, matching that of selective TMR methods, but with half the overhead. It means that the proposed method can improve fault resilience in QNNs, making them more reliable for safety-critical applications.

REFERENCES

[1] D. Silver *et al.*, "Mastering the game of go without human knowledge," *nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[2] S. Mozaffari *et al.*, "Deep learning-based vehicle behavior prediction for autonomous driving applications: A review," *IEEE T-ITS*, 2020.

[3] Y. Ibrahim *et al.*, "Soft errors in dnn accelerators: A comprehensive review," *Microelectronics Reliability*, vol. 115, p. 113969, 2020.

[4] M. Shafique *et al.*, "Robust machine learning systems: Challenges, current trends, perspectives, and the road ahead," *IEEE Design & Test*, vol. 37, no. 2, pp. 30–57, 2020.

[5] U. Zahid *et al.*, "Fat: Training neural networks for reliable inference under hardware faults," in *2020 IEEE International Test Conference (ITC)*. IEEE, 2020, pp. 1–10.

[6] N. Khoshavi *et al.*, "Fiji-fin: A fault injection framework on quantized neural network inference accelerator," in *2020 19th IEEE International Conference on Machine Learning and Applications (ICMLA)*. IEEE, 2020, pp. 1139–1144.

[7] S. Mittal, "A survey on modeling and improving reliability of dnn algorithms and accelerators," *Journal of Systems Architecture*, vol. 104, p. 101689, 2020.

[8] A. Mahmoud *et al.*, "Hardnn: Feature map vulnerability evaluation in cnns," *arXiv preprint arXiv:2002.09786*, 2020.

[9] C. Schorn and other, "Accurate neuron resilience prediction for a flexible reliability management in neural network accelerators," in *2018 DATE*. IEEE, 2018, pp. 979–984.

[10] C. Schorn *et al.*, "An efficient bit-flip resilience optimization method for deep neural networks," in *2019 DATE*. IEEE, 2019, pp. 1507–1512.

[11] A. Ruospo and E. Sanchez, "On the reliability assessment of artificial neural networks running on ai-oriented mpsocs," *Applied Sciences*, vol. 11, no. 14, p. 6455, 2021.

[12] M. Abdullah Hanif and M. Shafique, "Salvagednn: salvaging deep neural network accelerators with permanent faults through saliency-driven fault-aware mapping," *Philosophical Transactions of the Royal Society A*, vol. 378, no. 2164, p. 20190164, 2020.

[13] M. H. Ahmadilivani *et al.*, "Deepvigor: Vulnerability value ranges and factors for dnns reliability assessment," in *28th IEEE European Test Symposium*. In press, 2023.

[14] E. Ozen and A. Orailoglu, "Just say zero: Containing critical bit-error propagation in deep neural networks with anomalous feature suppression," in *39th ICCAD*, 2020, pp. 1–9.

[15] R. David *et al.*, "Tensorflow lite micro: Embedded machine learning for tinyml systems," *Machine Learning and Systems*, vol. 3, pp. 800–811, 2021.

[16] R. Leveugle *et al.*, "Statistical fault injection: Quantified error and confidence," in *2009 DATE*. IEEE, 2009, pp. 502–506.

Appendix 11

XI

M. H. Ahmadilivani, M. Barbareschi, S. Barone, A. Bosio, M. Daneshtalab, S. Della Torca, G. Gavarini, M. Jenihhin, J. Raik, A. Ruospo, et al., "Special Session: Approximation and Fault Resiliency of DNN Accelerators," in 2023 IEEE 41st VLSI Test Symposium (VTS), pp. 1–10, 2023.

Special Session: Approximation and Fault Resiliency of DNN Accelerators

Mohammad Hasan Ahmadilivani[1], Mario Barbareschi[2], Salvatore Barone[2], Alberto Bosio[3], Masoud Daneshtalab[4,1], Salvatore Della Torca[2], Gabriele Gavarini[5], Maksim Jenihhin[1], Jaan Raik[1], Annachiara Ruospo[5], Ernesto Sanchez[5], and Mahdi Taheri[1*]

[1]Tallinn University of Technology, Tallinn, Estonia
[2]University of Naples Federico II, Naples, Italy
[3]Ecole Centrale de Lyon, Lyon, France
[4]Mälardalen University, Västerås, Sweden
[5]Politecnico di Torino, Torino, Italy

2023 IEEE 41st VLSI Test Symposium (VTS) | 979-8-3503-4630-5/23/$31.00 ©2023 IEEE | DOI: 10.1109/VTS56346.2023.10140043

***Abstract*—Deep Learning, and in particular, Deep Neural Network (DNN) is nowadays widely used in many scenarios, including safety-critical applications such as autonomous driving. In this context, besides energy efficiency and performance, reliability plays a crucial role since a system failure can jeopardize human life. As with any other device, the reliability of hardware architectures running DNNs has to be evaluated, usually through costly fault injection campaigns. This paper explores approximation and fault resiliency of DNN accelerators. We propose to use approximate (AxC) arithmetic circuits to agilely emulate errors in hardware without performing fault injection on the DNN. To allow fast evaluation of AxC DNN, we developed an efficient GPU-based simulation framework. Further, we propose a fine-grain analysis of fault resiliency by examining fault propagation and masking in networks.**

***Index Terms*—deep neural networks, approximate computing, fault emulation, reliability, resiliency assessment**

I. Introduction

Deep Neural Networks (DNNs) have evolved to be increasingly applied to assist different aspects of human life, e.g., healthcare, transportation, security, IoT and edge applications [1]. In this context, energy efficiency and performance are the key constraints to be taken into account in designing DNN accelerators. Approximate Computing (AxC) is an emerging paradigm applied for improving their efficiency that produces acceptable results despite inaccuracies in the computations [2], [3].

Employing DNN accelerators in safety-critical applications has raised hardware reliability concerns. In compliance with ISO 26262 functional safety standard for road vehicles, the FIT (Failures In Time) rate of particular hardware components has to be 10 failures in 1 billion hours of operation at maximum to meet the target safety integrity level, which necessitates very circumspect design [4], [5]. The reliability of DNN accelerators is boosted by their ability to function correctly even in the presence of environment-related faults (soft errors, electromagnetic effects, temperature variations) or faults in the underlying hardware (manufacturing defects, process variations, nanoelectronics aging effects) [6]. DNNs are known to be resilient to faults due to their numerous interconnected layers and the ability to mask faults [7]. However, several studies in recent years have shown that the accuracy of DNNs may still drop significantly in the presence of faults [6], [8]–[11]. These observations demonstrate that the reliability of DNN accelerators must be considered alongside efficiency. Some research works studied the reliability of approximated DNNs to show the trade-off between reliability and efficiency [12], [13].

The key challenge for DNN efficiency and reliability is the exploration of the huge design space. As mentioned, employing AxC units in DNN accelerators is one of the eminent approaches to gaining efficiency. However, the design space for approximated DNNs is too large [14], and implementing different AxC units to find an optimum efficiency is impracticable for FPGA accelerators. Notably, Graphic Processing Units (GPUs) that are widely applied for accelerating the DNN training can be utilized to assist this process as well. To tackle the task of exploiting AxC in DNNs, we present a GPU-accelerated framework for DNN approximation exploration.

Addressing accelerators' reliability issues starts with architecture-level fault-resiliency evaluation. Fault Injection (FI) is a conventional method for this purpose that has been vastly applied for DNNs as well [15], [16]. The main approaches for FI experiments are fault simulation in software and fault emulation in hardware, both implying a huge fault space. Fast fault emulation in accelerators (especially in FPGAs, which are widely used for DNNs [17]) is still a challenge because of its iterative procedure, including numerous extra memory accesses as well as huge fault injection campaigns. To tackle this issue, we leverage AxC units in DNNs as a non-conventional use of both FI and AxC, to emulate errors in the accelerator hardware. In this method, AxC units and their variants are a substitution for FI targeting the fault resilience analysis of DNN architectures.

Moreover, reducing fault space can also be done at the software level. We have carried out an empirical study on the inherent resilience to faults and errors of DNNs, with the aim of investigating how they can mask a large portion of faults. In line with this, we propose the adoption of three different metrics to compute in advance (right after the injection of the

*The authors are sorted in alphabetic order.

fault) the effect the fault will have on the output vector score. In this way, it might be possible to both reduce the fault space and lower the FI time.

The paper is organized as follows: Section II introduces the GPU-accelerated framework for DNNs approximation exploration, Section III presents a method for harnessing approximation for agile analysis of fault resiliency in DNN accelerators, Section IV provides a fine-grain DNNs fault resiliency study by examining fault propagation and masking in networks, and Section V concludes the paper.

II. GPU Accelerated Framework for CNN Approximation

A. Motivations and Related Works

As stated in the introduction, the Approximate Computing paradigm is widely used to improve the energy efficiency of hardware accelerators for DNNs. In particular, one promising solution is to use approximate arithmetic circuits [18]–[20]. However, quantifying the error introduced by these circuits requires expensive hardware prototyping, and, as a result, a software emulator of the DNN accelerator is often executed on a CPU or General Purpose - Graphic Processing Unit (GP-GPU) instead. Nevertheless, this emulation is typically much slower than a software DNN implementation running on a CPU or GP-GPU that uses the standard floating-point arithmetic instructions and common DNN libraries because CPUs and GP-GPUs lack hardware support for approximate arithmetic operations; therefore, the latter operations must be emulated, that is costly.

To address this issue, we propose Inspect-NN (I-NN), that provides efficient emulation for approximate circuits to be deployed in DNNs accelerator: approximate circuits are implemented as look-up tables and accessed through the memory mechanism of CUDA-capable GP-GPUs, reducing the inference time of the emulated DNN accelerator by approximately 200 times compared to an optimized CPU version on complex DNNs.

In the following, we present the I-NN framework in Section II-B, while Section II-C discusses case studies concerning the use of the mentioned framework to assess the accuracy loss due to approximate multipliers in Artificial Neural Networks (ANNs).

B. Proposed method

The main purpose of the I-NN framework is to investigate the impact of erroneous components on Artificial Intelligence (AI) applications. In particular, it allows investigating how the accuracy of DNNs-based applications is affected by imprecise components, i.e., those that do not meet their nominal behavioral specifications either because of faults, or because they have been specifically designed to differ in a controlled way from that behavior, while pursuing performance advantages. Examples are arithmetic components designed while exploiting the Approximate Computing (AxC) design paradigm [21]. The behavior of imprecise components are modeled at the behavioral level by exploiting lookup tables, in which input operands select the corresponding output of the component. I-NN exploits parallelism allowed by GP-GPUs: the inference phase is split in blocks, each assigned to a thread block on the GP-GPU and executed independently and parallelly from the others. I-NN does the latter computation through a kernel, i.e., a CUDA function called by the CPU and executed on the GP-GPU: operations within each layer are parallelized so that each thread block execute a part of the overall operation; then, if needed, the output is normalized to be represented using *n* bits, with *n* being configurable. Data exchange between the CPU and the GP-GPU are minimized: data is copied from the GP-GPU memory to the CPU ones when strictly required; hence, if two consecutive layers are working on the GP-GPU, the first one feeds the GP-GPU memory address of the computed data to the next layer, rather than coping them back and forth from/to the CPU.

TABLE I: Error characterization and hardware requirements for approximate circuits taken from the EvoApproxLib-Lite library, as reported in [22]

Circuit name	MAE (%)	AWCE (%)	MRE (%)	Power (nW)	MAE (μm^2)
mul8s_1KV6	0.00	0.00	0.00	0.425	729.8
mul8s_1KV8	0.0018	0.0076	0.28	0.422	711.0
mul8s_1KV9	0.0064	0.026	0.90	0.410	685.2
mul8s_1KVA	0.019	0.075	2.53	0.391	641.1
mul8s_1KVM	0.049	0.20	2.40	0.369	652.8
mul8s_1KVP	0.051	0.21	2.73	0.363	635.0
mul8s_1KVQ	0.056	0.25	3.64	0.351	599.8
mul8s_1KX5	0.15	0.69	8.93	0.289	543.0
mul8s_1KXF	0.34	1.37	15.72	0.237	482.4
mul8s_1L2J	0.081	0.39	4.41	0.301	558.9
mul8s_1L2L	0.23	1.16	12.26	0.200	411.6
mul8s_1L2N	0.52	2.66	27.44	0.126	284.9
mul8s_1L12	3.08	12.30	135.77	0.052	172.2

C. Experimental Results

Case studies discussed in this Section concern the evaluation of the accuracy loss due to the use of multipliers taken from the EvoApproxLib-Lite [22] library of approximate circuits while targeting several pre-trained DNNs. In particular, through I-NN (i) we import the DNN to be analyzed directly from the most common machine learning frameworks, such as TensorFlow, TensorFlow LITE, and (ii) we define which specific approximate components have to be used, and (iii) we specify whether the analysis has to be performed at either coarse or fine grain. In coarse grain analysis, a single approximate component is deployed in the whole network. Conversely, in fine-grain analysis, each layer of the target DNN can use a different imprecise component.

We deploy multipliers from [22] – whose error characterization and hardware overhead are reported in Table I, for the reader convenience – to LeNet5 Convolutional Neural Network (CNN) [23], to MinNet, and to ResNet-8 [24], that, although trained using floating-point arithmetic, are all quantized to use 8-bit integer. The first CNN, i.e., LeNet5, has been trained to classify images from the Modified National Institute of Standards and Technology (MNIST) benchmark [25], on which it exhibits 99.07% accuracy. The MinNet CNN is a custom-made CNN inspired by the LeNet5 architecture: as for the latter, it consists of two Convolutional Layers (CLs), a Fully-Connected Layers (FCLs) and one Pooling Layers (PLs) between each CL, and it consists of approximately 160 thousand parameters. Despite its small size w.r.t. state-of-the-art networks, it exhibits 80.07% accuracy on the CIFAR-10 dataset [26]. Last,

TABLE II: Accuracy loss and computational time for approximate circuits taken from the EvoApproxLib-Lite library [22].

Circuit Name	LeNet5			MinNet		ResNet8	
	Acc. Loss (%)	GPU Time	CPU Time	Acc. Loss (%)	GPU Time	Acc. Loss (%)	GPU Time
mul8s_1KV6	0	13.23s	≈10h	0	13.0s	0	31.07s
mul8s_1KV8	0.07	13.19s	≈10h	-0.3	13.6s	-0.19	31.1s
mul8s_1KV9	0.15	13.27s	≈10h	0.3	13.6s	-0.42	31.3s
mul8s_1KVA	0.51	13.22s	≈10h	2.5	13.5s	-0.08	31.3s
mul8s_1KVM	0.16	13.23s	≈10h	-0.4	13.5s	0.12	31.5s
mul8s_1KVP	0.27	13.17s	≈10h	-0.8	13.7s	-0.18	31.4s
mul8s_1KVQ	0.61	13.18s	≈10h	0.5	13.5s	0.09	31.4s
mul8s_1KX5	1.77	13.18s	≈10h	5.5	13.5s	5.48	31.3s
mul8s_1KXF	1.57	13.18s	≈10h	-1.2	13.6s	8.45	31.2s
mul8s_1L2J	0.79	13.2s	≈10h	46.6	13.6s	74.61	31.5s
mul8s_1L2L	3.81	13.14s	≈10h	61.5	14.2s	73.73	31.8s
mul8s_1L2N	15.92	13.11s	≈10h	65.9	14.0 s	74.52	32.06s
mul8s_1L12	75.66	13.15s	≈10h	66.4	14.4s	74.49	33.6s

the ResNet-8 CNN, instead, has been trained while targeting images taken from the CIFAR-10 dataset [26], which consists of 60 thousand RGB images, each belonging to one among ten classes. The network, that consists of more than 300 thousand learned parameters, and it exhibits 84.31% accuracy on the mentioned dataset. During the inference phase, these three architectures require performing 400 thousand, 4 million and 40 million multiplications each, respectively; hence, they represent a good test case for the evaluation of execution time.

To estimate the error introduced by the approximation, we execute the approximate CNN to obtain its classification accuracy on the whole test data set, reporting the accuracy-loss and computational time required for the inference phase in Table II. The latter table also reports the error and hardware parameters for each of the considered approximate multipliers. We performed the inference phase on an NVIDIA RTX A5000 GP-GPU, that is built on the NVIDIA Ampere architecture and combines 256 Tensor Cores and 8192 CUDA cores with 24 GB of graphics memory. Furthermore, for comparison purpose, the computational time of the inference phase while resorting to a CPU-only implementation is reported in Table II. In this case, we leverage two 3.20 GHz Intel Xeon Silver 4210 CPUs, providing 20 cores / 40 threads computing power. We reported CPU time only for the LeNet5 case. For the MinNet and ResNet8, the CPU execution time was higher than 10 hours and we were not able to complete the experiments.

As it is easy to foresee, the speed-up provided by the GP-GPU is crucial: we can state that by exploiting the GP-GPU through our look-up table implementation of approximate multiplier allows for tremendous performance improvements, even though we compared the execution time. Furthermore, it can be noticed that the execution time increases as the number of multiplications performed during the inference phase increases, and it is independent of the particular approximate multiplier being deployed, as it can be observed in Table II.

III. Harnessing Approximation for Fault Injection in DNN Accelerators

A. Motivations and Related Works

A major consequence of single or multiple accumulated soft-error-caused bitflips affecting the weights of a given layer is their propagation as errors at the layer outputs (also known as layer Output Feature Map) and further throughout the subsequent layers, leading to incorrect DNN predictions. *Fault resilience* is the ability to tolerate the impact of faults on the output accuracy, and, in practice, it is one of the contributors to the final DNN accelerators' reliability. A relevant mitigation strategy at the architecture level can be a hardening of the DNN, e.g., by layer redesign or selective hardening of neurons, such as hardened Processing Elements (PEs) or Triple Modular Redundancy (TMR) variants [8]. These imply the assessment of layers' fault resiliency or identification of critical neurons in a neural network that are the most vulnerable to faults [27]–[29]. Fig. 1 presents a taxonomy for DNN reliability assessment methods. Along with analytical and hybrid methods [29], Fault Injection (FI) is a commonly used method for evaluating the fault resilience of DNNs [11], [30], [31]. The industry often employs fault injection by emulation in hardware, particularly in FPGAs, as it allows for evaluating real-scale DNN accelerator designs in significantly shorter run times than software-based simulations [9].

Fiji-FIN [32] is a representative framework implemented on the embedded Processing System for evaluating the resiliency of DNNs by emulating FI on FPGA. It measures accuracy degradation as a metric to study the impact of soft errors on network parameters. Designing fault injection campaigns for such frameworks requires significant effort, as each injection halts inference execution to manipulate DNN parameters. This interrupts classification time for a batch of inputs.

The state-of-the-art approaches for FI by emulation in FPGA using the embedded Processing System often require iterative procedures for each injected fault. In particular, such an iterative approach breaks the pipeline execution of the accelerator, requires a complex FI controller, and needs an extra FI control interconnection to handle the injection [32]–[34]. These procedures also involve multiple additional memory accesses, resulting in time-consuming processes and complex implementation.

Unlike the works mentioned above, our proposed method can be classified as fault injection by emulation in Programmable Logic. It leverages the functional approximation as a substitute for the errors generated by FI to improve processing and design time as well as the control complexity in the DNN fault resiliency analysis process. This approach allows the inference pipeline to be executed on a batch of inputs without interruption. This agile method enables a fast and efficient exploration of different options for network architecture, training, dataset selection, and more, to study the fault resilience of DNNs. Specifically, the introduced errors mimic single or multiple accumulated faults in weights. The method allows for efficient analysis of how subsequent layers in the network tolerate errors in the Output Feature Map of an assumed compromised layer are affected by faults in the weights of a compromised layer.

To the best of our knowledge, this is the first time that AxC units are utilized to enhance the efficiency and reduce the complexity of resilience analysis for DNNs.

B. Proposed method

AxC is commonly used to approximate hardware components to improve compute efficiency while maintaining functional accuracy. However, in practice, the errors induced by approximation can be used to mimic the errors caused by faults in logic circuits. These errors affect the outputs of the corresponding

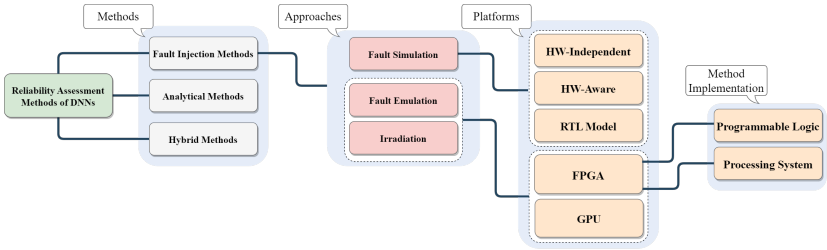


Fig. 1: A taxonomy of DNN reliability assessment methods

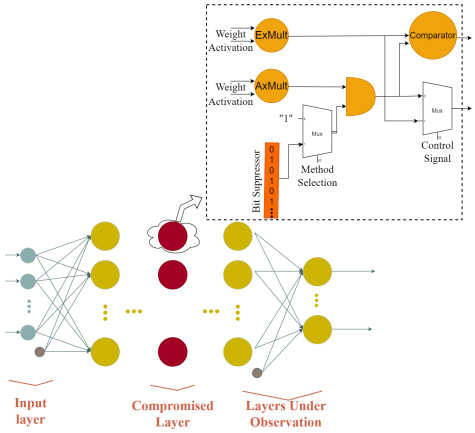


Fig. 2: Proposed method evaluation

units and propagate to subsequent layers, impacting their activations (Fig. 2). The proposed approach for evaluating DNN's fault resiliency using approximate computing (AxC) units is presented in Fig. 2. To implement our proposed method, an AxMult, or an AxMult + a bit suppression unit (AxMult+) is implemented along with the exact implementation of the multipliers (ExMult) in the network, depending on whether the network is being run in functional or fault resilience assessment mode. The golden inference for the validation dataset is run only once, and the layer outputs are stored and compared with a Comparator unit. The Bit Suppressor unit is meant to increase the probability of more significant bits of the neuron being impacted by faults. The less significant bits of the layer Output Feature Map are already affected by the AxMult with proper randomness depending on the data distribution in the network and layers.

The overall flow of the proposed method is illustrated in Fig. 3. In Step 1, the user initializes the method by selecting the compromised layer in the DNN structure, the validation dataset (i.e., DNN inputs), and the application-specific target fault rate assumed for the analysis. In Step 2, suitable AxC units are selected for Approximate Processing Elements (AxPEs), such as the AxC multipliers from a relevant library, e.g., the EvoApproxLib [35], or their variants with bit suppression. In Step 3, the selected AxMults started executing the compromised layer by enabling corresponding AxPEs along with the Exact Processing Elements (ExPE) in the DNN architecture. The DNN inference is run while keeping the network pipeline intact, and the resulting DNN output accuracy drop is recorded as the primary metric for analyzing DNN fault resilience. A more significant drop in accuracy with induced errors implies a less fault-resilient DNN implementation. At the same time, the outputs of the AxMults are compared with the ExMults outputs to calculate the actual error at each neuron. The rest of the inference is executed by ExMults for both erroneous and exact outputs, and the comparison is performed for all the subsequent neurons of the network.

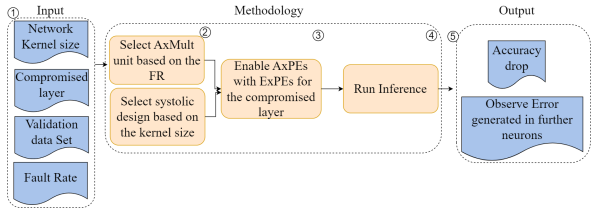


Fig. 3: Methodology flow

The characteristics of the approximation-induced errors can be evaluated using different metrics such as normalized error, number of flipped bits, and impact on the neural network classification accuracy drop. In this study, we rely on a simple set of metrics that includes:

- Normalized error: the average error on the output of each layer is calculated by subtracting the neurons' outputs of that layer from the golden output and dividing all the error values by the maximum value.
- Network accuracy: calculated by executing the network under different circumstances (faulty, AxMult, AxMult + bit suppressor and bit suppressor) over the test set.
- Bitflips in subsequent layers: calculated by comparing all bits in the next layers' outputs with the golden model and counting the bits that do not match as flipped bits.

1) Accelerator Model: Fig. 4 illustrates the accelerator model to perform resilience analysis on FPGA. It consists of two different systolic architecture designs based on the network under test. The $N \times N$ systolic architecture is used based on the convolution layers' kernel size to perform the most optimum dot matrix. At the same time, all designs have ExPE and AxPE to perform the resilience analysis and benefits of a dual register to store the results of both approximate systolic and exact systolic for further comparisons. An Error Detector (ED) module is also provided to compute the error generated at each neuron's output compared to the exact output and can be used for the neuron's vulnerability evaluation.

This implementation provides us following features:

(a) Understanding the vulnerability of neurons by computing the error generated through the hardware and further layers by comparing the exact and approximate systolic design outputs;
(b) Increasing the controllability for enabling errors in each layer individually and keeping the other layers correct;
(c) Eliminating the need for designing and deploying an extra complex controller for the fault injection procedure. A simple approximate unit enabling circuitry is employed instead;
(d) The inference pipeline process executes a batch of inputs with no need to break this process;
(e) The resilience assessment process is performed without an extra interconnect for weight sampling;
(f) The proposed approach is not iterative for each potential fault location (unlike the traditional fault injection). Thus, the analysis complexity is vastly reduced.

Note that the features (c)-(f) are specific for FI emulation in Programmable Logic and generally not available in Processing Logic based methods such as Fiji-FIN.

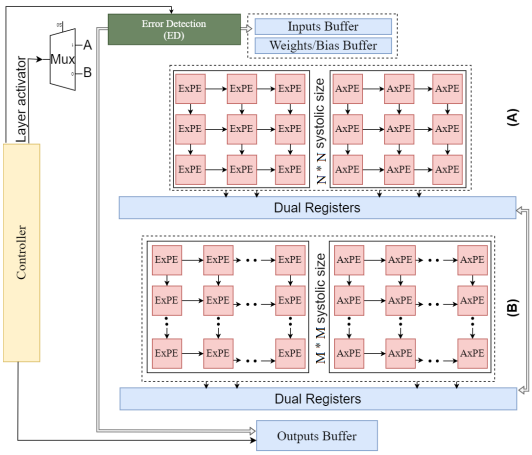


Fig. 4: Proposed systolic architecture for our Resiliency assessment DNN accelerator framework

C. Experimental Results

1) Evaluation methodology: To assess the feasibility of the proposed method, we implemented the same flow as shown in Fig. 2 with fault injection (FI). Using Table I, we narrowed down the list of candidate approximate multipliers from the EvoApproxLib library [35] based on several relevant metrics, with a primary focus on two established features, namely, the Variance of Error Distance (Var-ED) and Root Mean Square (RMS-ED) presented in [36]. These metrics are crucial in determining the approximation-induced errors that affect the performance of an AxC unit in DNNs. We selected mul8s_1L2N for the experiment based on these metrics and results achieved from the high-level experiments on the network through the proposed GPU accelerated framework for CNN approximation in Section II.

For the reference part, we repeated the fault resiliency evaluation on the original network, which was instrumented with a state-of-the-art FI method [32]. In this study, we considered the injection of multiple bitflips at a random location in all OFM' bits of the compromised layer for every input in the DNN validation test set. In this case, we assumed that 10% of the weights' bits were faulty.

To achieve a high FI confidence level using the statistical fault injection approach [37], we repeated the experiment for each fault model with 1000 random faults per image. The average accuracy of all repetitions was then reported.

We evaluated the impact of AxMult, AxMult + Bit Suppression (AXMult+), Bit Suppression alone, and fault injection, along with normalized error and the number of flipped bits, on the DNN accuracy. The results show a drop in DNN accuracy due to these factors. We compared the normalized error and the number of flipped bits for each scenario.

2) Experimental Setup: To evaluate the feasibility of the proposed method, a case-study Convolutional Neural Network (CNN) with two convolutional layers, two max-pooling, and one Fully-Connected (FC) layer was implemented and trained. The simulations were performed on an Intel® Core™ i7-6800K CPU @ 3.40GHz × 12, and the proposed method was implemented with Python 3. The hardware synthesis and implementation results are produced by the Xilinx Vivado HLS tool on a Xilinx Versal VCK190 FPGA (xcvc1902-vsva2197-2MP-e-S) at 166 MHz operational frequency.

The CNN under study is trained on a dataset of 2000 images of animals (cats and dogs) and humans for binary classification. The accuracy of the network over the test set (including 450 images of animals and humans) is 93.34%. Bit truncation quantization is applied in network parameters during training, and data precision is reduced to 8-bit.

3) Evaluation Results: We analyzed the similarity of the fault resiliency analysis results obtained by fault emulation and our proposed method using the metrics identified in Section III-B.

Fig. 6 shows the distribution of *normalized error* in the output of the second convolutional layer (Conv2) in the presence of 10% random faults in the first convolution layer (grey), errors induced by AxMult (blue), and errors induced by AxMult + bit suppressor (orange) enabled in the first convolution layer, respectively. Fig. 5 reports the impact of applying FI and our proposed method on the same convolutional layer and its effect on the second pooling layer of the network. These results demonstrate the similarity in error propagation trends between the proposed and reference methods.

In practice, by analyzing these charts, users can set a criticality threshold on the output error of the neurons based on their application and determine the number and indices of neurons to be used for any protection techniques. Generally, if we set the threshold at some error value, all methods suggest some neuron indices for mitigation techniques. As it can be concluded, both AxMult + bit suppression and FI show very similar behaviors. However, relying solely on the AxMult or bit suppression techniques is quite inaccurate for high fault ratios like this case study here.

For example, by setting the error threshold to 0.7, FI will recommend the user to protect 50 out of 1024 neurons of the

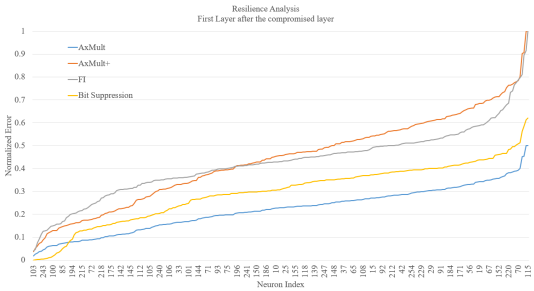


Fig. 5: Normalized output error of Pool2: Applying AxMult, AxMult+ , Bit Suppression and FI on the Conv1

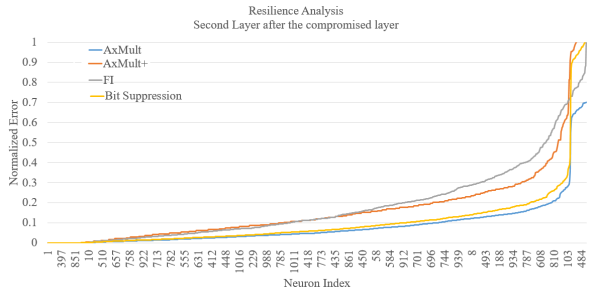


Fig. 6: Normalized output error of Conv2: Applying AxMult, AxMult+ , Bit Suppression and FI on the Conv1

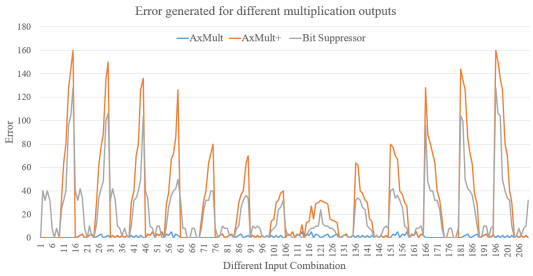


Fig. 7: Multiplication output error generated by AxMult, AxMult+ and Bit Suppression

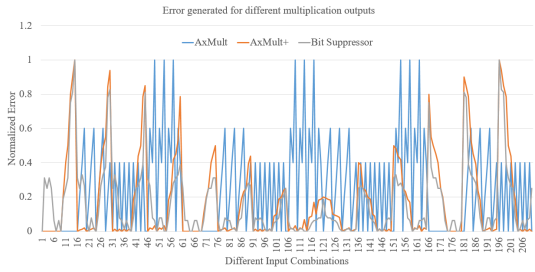


Fig. 8: Normalized Multiplication output error generated by AxMult, AxMult+, and Bit Suppression

Conv2 network's second CONV's neurons, while AxMult + bit suppression will recommend 53 out of 1024 neurons, including all the critical neurons recognized by FI. Fig. 7 and Fig. 8 show the error distribution of the three different methods, i.e., AxMult, AxMult + bit suppressor, and bit suppressor on the output of a multiplication operation with all the combinations of two 8-bit inputs. From Fig. 7, it is evident that the error values generated by AxMult + bit suppressor can almost cover a vast range of different values, and Fig. 8 shows that the error is evenly distributed on all different input combinations.

Table. III is reporting the number of bitflips and accuracy drop in subsequent layers caused by the compromised first convolution layer. These results also demonstrate the strong similarity of the trends in error propagation by the AxMult and its variants with the reference method. In case of accuracy drop, AxMult + bit suppression shows a strong correlation with the FI method and surpasses the other two methods.

Table IV reports details of the hardware accelerator implementation. Based on the results, the proposed implementation can be executed on the FPGA at 166 MHz clock frequency, and only by using ∼16% of the available LUTs on the board all three mentioned systolic-array size architectures can be implemented to improve the efficiency of the accelerator. The timing comparison of the proposed method and the state-of-the-art fault injection method are presented in Table. V. As it can be concluded, by keeping an acceptable accuracy of FI in identifying the critical neurons, we get thousands of times speed-up in the resilience assessment of the DNNs. (Specifically, it is 5417 times in this example). At the same time, the proposed method does not need extra interconnects to manage the assessment process, and the original controller of the accelerator can take care of the fault resiliency assessment process.

TABLE III: Bitflips and Accuracy drop induced by our proposed method vs. the reference fault injection method by fault rate 10% in OFM of the first convolution layer

Measured Layer	Bitflips in subsequent layers			
	FI (reference) [%]	AxMult [%]	AxMult+ [%]	Bit suppressor [%]
Conv1	10	10.30	10	10.20
Pool1	9.07	9.20	9.06	9.15
Conv2	16.76	16.80	16.77	16.83
Pool2	16.51	16.66	16.53	16.62
Accuracy drop [%]				
	16.73	9.33	18.33	24.73

TABLE IV: Hardware implementation of the proposed hardware accelerator

	Resource Utilization (%)				
Conv2D systolic size	LUT	FF	BRAM	Data Path Delay	CLK Frequency
3*3	0.03	0.00	0.83	Logic: ∼20% Route: ∼80%	166 MHz
5*5	0.09	0.00	0.83		
32*32	15.30	0.91	0.85		

TABLE V: Timing overheads of the proposed method vs. the reference fault injection method (Conv1 layer)

Network	Analysis Control Circuitry	Interconnects	DNN execution time in FPGA
Base CNN	N/A	Data Exchange Interconnect	∼120ms
Fault Resilience Assessment			
CNN instrumented with FI	Complex FI Controller	(Data Exchange + FI) Interconnect	∼650,000ms
CNN instrumented with AxMult+	Accelerator Controller	Data Exchange Interconnect	∼120ms

IV. Fault resiliency in DNNs

A. Motivations and Related Works

In the last few years, researchers have investigated the theory behind brain-inspired computational models to build artificial structures capable of addressing highly complex computational problems. Today, DNNs are considered attractive solutions in several fields due to their outstanding computational capabilities as well as their human-level performance. The human brain is a complex and fascinating system able to bear synapses or neuron faults and still keep working properly, thanks to its plastic ability to remodel, repair, and reorganize its neural functions. Similarly, artificial neural networks possess in their structure a certain degree of redundancy that leads to intrinsic robustness and resilience against the occurrence of faults. This is caused by two aspects: the first is related to their distributed and parallel structure; the second to the redundancy resulting from the over-provisioning [38]. Indeed, neural networks are furnished with a quantity of artificial neurons higher than the minimal number required to perform a computation. It means that they can bear a bounded number of errors thanks to the excessive neuron budget: once this number is exceeded, the precision degrades gracefully as the number of errors increases [39].

This structural feature allows them to have an attractive property known as *masking ability*, which corresponds to the ability of DNNs to stop the propagation of some faults by masking their effects. As an example, it has been shown that the presence in DNNs of the Rectified Linear Unit (ReLU) activation function halves the percentage of critical faults by stopping the propagation of faults on negative weights [40]. Understanding how faults propagate through the neural network is very important, as it may influence: the reliability assessment procedure; efficient fault detection and mitigation strategies.

The analysis of fault propagation in DNNs has been conducted in the literature by different perspectives. A preliminary theory-driven analysis is proposed in [41], where the authors explore inherent characteristics of fault propagations in DNNs from the theoretical aspect. They propose a formula to compute the perturbation caused by the *i*-th bit flip on a weight represented in a 32-bit floating-point format. The authors in [42] characterize the propagation of soft errors from the hardware to the application software of DNN systems. Based on this, they devise cost-effective solutions to mitigate Silent Data Corruption (SDC) in software and hardware. Further studies on faults propagations in DNNs are described in [43] and [44].

Nevertheless, it is important to underline that the major effort in the above-mentioned research works consists in understanding how critical faults (i.e., those that lead to application failures) propagate through the hardware-software system.

The intent of this section is twofold. On the one hand, it aims to show how a critical fault spreads through a network. On the other hand, this section tackles the problem from a different angle, showing how a *masked fault* is propagated within the system, analysing the role DNNs have in this process. The investigation of this latter category of faults is important for the following reasons:

- In a fault injection process, the identification of sets of faults that are masked may reduce the fault space and, as a consequence, lower the costs of the reliability assessment;
- In the design of DNN models, the knowledge of architectural elements that favour the masking ability of DNNs can lead to the design of more robust models.

This section presents an analysis on masked faults with the goal of identifying at what point in the computation their propagation is stopped and if it is possible to know in advance their effects on the output of the DNN.

B. Proposed Method

CNNs are a subset of DNNs composed of a set of convolutional layers. The output of each layer is a multidimensional tensor, often referred to as the *Output Feature Map* (OFM). In the field of Image Classification, the output of the network is represented by a vector called *logit*. A fault affecting a CNN can be classified as:

- **Critical**, if it causes a change in the network prediction;
- **Non-Critical**, if it impacts the logit without changing the prediction;
- **Masked**, if it does not modify the logit.

When a fault affects the parameters of a layer (i.e., weights), it may change its OFM, as well as the one of all the following layers. If the fault is masked, the difference between the *golden Output Feature Map* (gOFM) and the *faulty Output Feature Map* (fOFM) of the impacted layer should be small or zero. Contrarily, it is logical to assume that a critical fault also produces a fOFM that is radically different from the gOFM.

As a consequence of these two observations, it is possible to predict the impact of a fault without needing to carry out a complete inference. In fact, this section aims at showing that:

1) Masked faults, once triggered, rarely propagate for more than one layer. Thus, the only different OFM is the one of the layer directly affected by the fault;
2) Critical faults, can be immediately identified by performing some early measures, using some metrics that can be computed by comparing the fOFM and the gOFM of the affected layer.

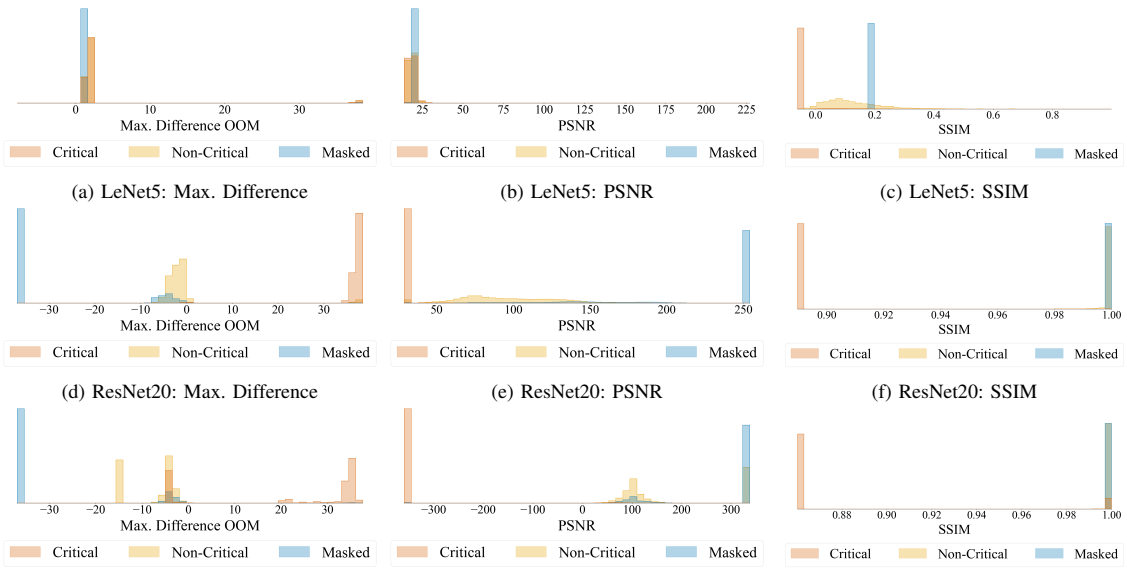
The OFM of a layer l can be interpreted as a collection of n filtered images, where n is the number of filters applied in layer l. Furthermore, the fOFM resulting from a fault in the network parameters can be interpreted as the gOFM plus a Gaussian noise. Therefore, it is possible to apply well-known objective image quality metrics, such as the Peak signal-to-noise Ratio (PSNR) and the Structural Similarity Index Metric (SSIM) [45].

This section proposes to use three different metrics to predict the criticality of a fault, starting from the OFM of the affected layer.

1) Max Difference: This first metric computes the maximum distance between the gOFM and the fOFM. This metric is presented as a baseline since, to the best of the authors' knowledge, there are no metrics that correlate the criticality of a fault with the changes in the OFM.

2) PSNR: This metric is directly proportional to the ratio between the peak signal (i.e., the maximum element of the gOFM) and the power of the corrupting noise, represented by the mean square error between the gOFM and the fOFM. The value can be computed as follows:

$$PSNR = 10 \cdot \log_{10} \frac{\max(gOFM)^2}{MSE(gOFM, fOFM)} \quad (1)$$



(a) LeNet5: Max. Difference (b) LeNet5: PSNR (c) LeNet5: SSIM

(d) ResNet20: Max. Difference (e) ResNet20: PSNR (f) ResNet20: SSIM

(g) DenseNet121: Max. Difference (h) DenseNet121: PSNR (i) DenseNet121: SSIM

Fig. 9: Metric probability distributions for the CNN under exam, observed in the layer where the fault is injected. Figures (a)-(c) refer to LeNet-5, Figures (d)-(f) refer to ResNet20 and Figures (g)-(i) refer to DenseNet121.

Where $\max(gOFM)$ is the maximum value of the gOFM and MSE is the Mean Square Error between the gOFM and the fOFM.

3) SSIM: this metric improves the PSRN, by including the concept of *structural information*, represented by the relationship of a neuron with its neighbours. The formula is composed by the product of three terms, the *luminance*, the *contrast* and the *structural* term. In the context of the study of the OFM, the simplified formula can be expressed as:

$$SSIM = \frac{(2\mu_f\mu_g + C_1)(2\sigma_{fg} + C_2)}{(\mu_f^2 + \mu_g^2 + C_1)(\sigma_f^2 + \sigma_g^2 + C_2)} \quad (2)$$

Where μ_g, μ_f are the mean of the gOFM and of the fOFM, σ_g, σ_f their standard deviation, σ_{fg} their cross-covariance. C_1 and C_2 are two regularization parameters.

C. Experimental Results

This section analyses three different CNNs used for Image Classification to study how a fault can propagate. The networks under analysis are: LeNet-5 with the MNIST dataset, ResNet20 with CIFAR-10 and Densenet-121 with ImageNet. For each network, we performed a statistical FI as described in [46]. The tool used to carry out the FI campaign is SCI-FI [47], that allows to speed up the FI process using the Fault Dropping and the Delayed Start techniques. The faults injected are single bit-flips in the network parameters, represented as 32-bit floating points. Further details on the networks under exam and on the FI campaigns are reported in Table VI.

Firstly, to demonstrate that Masked faults only modify the OFM of the layer affected by the fault, we report the percentage of Masked faults that affect more than one layer. In particular, for LeNet5, all the Masked faults do not modify any OFM besides the one of the impacted layer. For ResNet20, 87.99% of Masked faults show no effect in the OFM of the layer immediately after the impacted one, while for DenseNet this number rises to 99.17%.

TABLE VI: The networks under analysis

Network	Dataset	Dataset Size	Acc. [%]	Weights	Injected Faults
LeNet5	MNIST	10,000	98.85	61,706	2,212
ResNet20	CIFAR-10	10,000	91.72	269,722	15,675
DenseNet121	ImageNet	50,000	74.43	7,978,856	16,685

To show that Critical faults have a strong impact early on, we compute the metrics introduced in Section IV-B on the OFM of the layer affected by the fault. Figure 9 reports the metrics distributions for the Max Difference, the PSNR and the SSIM. Each image shows, for each network, the distribution of a metric computed for all the FI campaigns. In particular, the distribution is further subdivided according to the impact of the fault affecting the network when they were measured. This means that the distribution labelled 'Critical' reports only the value measured when a Critical fault is affecting the network. For a metric, the more separable the three distributions are, the better the metric is at predicting the effect of a fault.

In particular, we can observe a stark contrast between the metrics computed for LeNet5 and the other networks. This can be imputed to the lack of batch-normalization layers, that normalize the value of the weights (and of the OFM) between $[-1, 1]$. Consequently, even a bit-flip in the mantissa bits of the weight can have a large impact. Nonetheless, SSIM performs sufficiently well, as it correctly separates Critical and Non-Critical faults.

For the other two CNNs, we can notice that both the Max. Difference and the PSNR separate Masked faults from Critical

and Non-Critical faults. However, for ResNet20, SSIM outperforms the other metrics, as it completely splits ups Critical and Non-Critical faults while providing a good degree of separation between Masked and Non-Critical faults. Contrarily, for DenseNet-121, SSIM does not completely separate Masked from Critical. For this latter network, the best solution is offered by the PSNR.

Therefore, we observe how different metrics can correctly predict Masked and Critical faults without the need for a complete inference, by simply analysing the fOFM of the layer affected by the fault.

As a final note, we want to highlight that the cost of the computation of the metric is quite small, requiring only a portion of the time required for the computation of a whole layer. On average, the per-layer overhead added by the computation of one of the metrics is 76.51% for LeNet5 74.28% for ResNet20 and 73.54% for DenseNet121.

V. Conclusions

This paper explored approximation and fault resiliency of DNN accelerators. To allow fast evaluation of AxC DNN, an efficient GPU-based simulation framework was developed. The paper proposed a method for employing approximate (AxC) arithmetic circuits to agilely emulate errors in hardware without performing fault injection on the DNN. Finally, it presented a fine-grain analysis of fault resiliency by examining fault propagation and masking in networks.

Acknowledgments

This work was supported in part by the European Union through European Social Fund in the frames of the "Information and Communication Technologies (ICT) programme" ("ITA-IoIT" topic), by the Estonian Research Council grant PUT PRG1467 "CRASHLES" and by Estonian-French PARROT project "EnTrustED".

References

[1] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.

[2] G. Armeniakos, G. Zervakis, D. Soudris, and J. Henkel, "Hardware approximate techniques for deep neural network accelerators: A survey," *ACM Computing Surveys*, vol. 55, no. 4, pp. 1–36, 2022.

[3] A. Bosio, D. Ménard, and O. Sentieys, Eds., *Approximate Computing Techniques*. Springer International Publishing, 2022. [Online]. Available: https://doi.org/10.1007/978-3-030-94705-7

[4] A. Nardi and A. Armato, "Functional safety methodologies for automotive applications," in *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2017, pp. 970–975.

[5] M. Jenihhin, M. S. Reorda, A. Balakrishnan, and D. Alexandrescu, "Challenges of reliability assessment and enhancement in autonomous systems," in *2019 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, 2019, pp. 1–6.

[6] M. Shafique, M. Naseer, T. Theocharides, C. Kyrkou, O. Mutlu, L. Orosa, and J. Choi, "Robust machine learning systems: Challenges, current trends, perspectives, and the road ahead," *IEEE Design & Test*, vol. 37, no. 2, pp. 30–57, 2020.

[7] A. Bosio, P. Bernardi, A. Ruospo, and E. Sanchez, "A reliability analysis of a deep neural network," in *2019 IEEE Latin American Test Symposium (LATS)*. IEEE, 2019, pp. 1–6.

[8] S. Mittal, "A survey on modeling and improving reliability of dnn algorithms and accelerators," *Journal of Systems Architecture*, vol. 104, p. 101689, 2020.

[9] Y. Ibrahim, H. Wang, J. Liu, J. Wei, L. Chen, P. Rech, K. Adam, and G. Guo, "Soft errors in dnn accelerators: A comprehensive review," *Microelectronics Reliability*, vol. 115, p. 113969, 2020.

[10] C. Torres-Huitzil and B. Girau, "Fault and error tolerance in neural networks: A review," *IEEE Access*, vol. 5, pp. 17 322–17 341, 2017.

[11] F. Su, C. Liu, and H.-G. Stratigopoulos, "Testability and dependability of ai hardware: Survey, trends, challenges, and perspectives," *IEEE Design & Test*, 2023.

[12] L. M. Luza, D. Söderström, G. Tsiligiannis, H. Puchner, C. Cazzaniga, E. Sanchez, A. Bosio, and L. Dilillo, "Investigating the impact of radiation-induced soft errors on the reliability of approximate computing systems," in *2020 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*. IEEE, 2020, pp. 1–6.

[13] M. Taheri, M. Riazati, M. H. Ahmadilivani, M. Jenihhin, M. Daneshtalab, J. Raik, M. Sjõdin, and B. Lisper, "Deepaxe: A framework for exploration of approximation and reliability trade-offs in dnn accelerators," in *24th International Symposium on Quality Electronic Design*. https://doi.org/10.48550/arXiv.2303.08226, 2023.

[14] M. Pinos, V. Mrazek, F. Vaverka, Z. Vasicek, and L. Sekanina, "Acceleration techniques for automated design of approximate convolutional neural networks," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, pp. 1–1, 2023.

[15] A. Ruospo, L. M. Luza, A. Bosio, M. Traiola, L. Dilillo, and E. Sanchez, "Pros and cons of fault injection approaches for the reliability assessment of deep neural networks," in *2021 IEEE 22nd Latin American Test Symposium (LATS)*. IEEE, 2021, pp. 1–5.

[16] A. Bosio, I. O'Connor, M. Traiola, J. Echavarria, J. Teich, M. A. Hanif, M. Shafique, S. Hamdioui, B. Deveautour, P. Girard *et al.*, "Emerging computing devices: Challenges and opportunities for test and reliability," in *2021 IEEE European Test Symposium (ETS)*. IEEE, 2021, pp. 1–10.

[17] M. A. Talib, S. Majzoub, Q. Nasir, and D. Jamal, "A systematic literature review on hardware implementation of artificial intelligence algorithms," *The Journal of Supercomputing*, vol. 77, pp. 1897–1938, 2021.

[18] M. Barbareschi, S. Barone, and N. Mazzocca, "Advancing synthesis of decision tree-based multiple classifier systems: an approximate computing case study," *Knowledge and Information Systems*, pp. 1–20, Apr. 2021, company: Springer Distributor: Springer Institution: Springer Label: Springer Publisher: Springer London. [Online]. Available: https://link.springer.com/article/10.1007/s10115-021-01565-5

[19] M. Barbareschi, S. Barone, A. Bosio, J. Han, and M. Traiola, "A Genetic-algorithm-based Approach to the Design of DCT Hardware Accelerators," *ACM Journal on Emerging Technologies in Computing Systems*, vol. 18, no. 3, pp. 1–25, Jul. 2022. [Online]. Available: https://dl.acm.org/doi/10.1145/3501772

[20] M. Barbareschi, S. Barone, N. Mazzocca, and A. Moriconi, "A Catalog-based AIG-Rewriting Approach to the Design of Approximate Components," *IEEE Transactions on Emerging Topics in Computing*, 2022.

[21] A. Bosio, D. Ménard, and O. Sentieys, Eds., *Approximate Computing Techniques: From Component- to Application-Level*. Cham: Springer International Publishing, 2022. [Online]. Available: https://link.springer.com/10.1007/978-3-030-94705-7

[22] V. Mrazek, Z. Vasicek, L. Sekanina, H. Jiang, and J. Han, "Scalable Construction of Approximate Multipliers With Formally Guaranteed Worst Case Error," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 11, pp. 2572–2576, Nov. 2018, conference Name: IEEE Transactions on Very Large Scale Integration (VLSI) Systems.

[23] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998, conference Name: Proceedings of the IEEE.

[24] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Las Vegas, NV, USA: IEEE, Jun. 2016, pp. 770–778. [Online]. Available: http://ieeexplore.ieee.org/document/7780459/

[25] Y. LeCun, C. Cortes, and C. Burges, "MNIST Handwritten digit database," 1998. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[26] A. Krizhevsky, V. Nair, and G. Hinton, "CIFAR-10 (canadian institute for advanced research)," 2010. [Online]. Available: https://www.cs.toronto.edu/ kriz/cifar.html

[27] C. Schorn *et al.*, "Accurate neuron resilience prediction for a flexible reliability management in neural network accelerators," in *2018 DATE*. IEEE, 2018, pp. 979–984.

[28] A. Ruospo and E. Sanchez, "On the reliability assessment of artificial neural networks running on ai-oriented mpsocs," *Applied Sciences*, vol. 11, no. 14, p. 6455, 2021.

[29] M. H. Ahmadilivani, M. Taheri, J. Raik, M. Daneshtalab, and M. Jenihhin, "Deepvigor: Vulnerability value ranges and factors for dnns' reliability assessment," *arXiv preprint arXiv:2303.06931*, 2023.

[30] A. Ruospo, E. Sanchez, L. M. Luza, L. Dilillo, M. Traiola, and A. Bosio, "A survey on deep learning resilience assessment methodologies," *Computer*, vol. 56, no. 2, pp. 57–66, 2023.

[31] M. Taheri, M. H. Ahmadilivani, M. Jenihhin, M. Daneshtalab, and J. Raik, "Appraiser: Dnn fault resilience analysis employing approximation errors," in *26th International Symposium on Design and Diagnostics of Electronic Circuits and Systems*. In press, 2023.

[32] N. Khoshavi, C. Broyles, Y. Bi, and A. Roohi, "Fiji-fin: A fault injection framework on quantized neural network inference accelerator," in *2020 19th IEEE International Conference on Machine Learning and Applications (ICMLA)*. IEEE, 2020, pp. 1139–1144.

[33] M.-C. Hsueh, T. K. Tsai, and R. K. Iyer, "Fault injection techniques and tools," *Computer*, vol. 30, no. 4, pp. 75–82, 1997.

[34] N. Khoshavi, A. Roohi, C. Broyles, S. Sargolzaei, Y. Bi, and D. Z. Pan, "Shieldenn: Online accelerated framework for fault-tolerant deep neural network architectures," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2020, pp. 1–6.

[35] V. Mrazek, R. Hrbacek, Z. Vasicek, and L. Sekanina, "Evoapprox8b: Library of approximate adders and multipliers for circuit design and benchmarking of approximation methods," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, March 2017, pp. 258–261.

[36] M. S. Ansari, V. Mrazek, B. F. Cockburn, L. Sekanina, Z. Vasicek, and J. Han, "Improving the accuracy and hardware efficiency of neural networks using approximate multipliers," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 2, pp. 317–328, 2019.

[37] R. Leveugle, A. Calvez, P. Maistri, and P. Vanhauwaert, "Statistical fault injection: Quantified error and confidence," in *2009 Design, Automation & Test in Europe Conference & Exhibition*. IEEE, 2009, pp. 502–506.

[38] V. Piuri, "Analysis of fault tolerance in artificial neural networks," *Journal of Parallel and Distributed Computing*, vol. 61, no. 1, pp. 18 – 48, 2001.

[39] E. M. El Mhamdi and R. Guerraoui, "When neurons fail," in *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2017, pp. 1028–1037.

[40] F. Angione *et al.*, "Test, reliability and functional safety trends for automotive system-on-chip," in *2022 IEEE European Test Symposium (ETS)*, 2022, pp. 1–10.

[41] R. Sun, J. Zhan, and W. Jiang, "An insight into fault propagation in deep neural networks: Work-in-progress," in *2020 International Conference on Embedded Software (EMSOFT)*, 2020, pp. 20–21.

[42] G. Li, S. K. S. Hari, M. Sullivan, T. Tsai, K. Pattabiraman, J. Emer, and S. W. Keckler, "Understanding error propagation in deep learning neural network (dnn) accelerators and applications," in *SC17: International Conference for High Performance Computing, Networking, Storage and Analysis*, 2017, pp. 1–12.

[43] J. E. R. Condia, J.-D. Guerrero-Balaguera, F. F. Dos Santos, M. S. Reorda, and P. Rech, "A multi-level approach to evaluate the impact of gpu permanent faults on cnn's reliability," in *2022 IEEE International Test Conference (ITC)*, 2022, pp. 278–287.

[44] F. F. Dos Santos, P. Rech, A. Kritikakou, and O. Sentieys, "Evaluating the impact of mixed-precision on fault propagation for deep neural networks on gpus," in *2022 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2022, pp. 327–327.

[45] A. Horé and D. Ziou, "Image quality metrics: Psnr vs. ssim," in *2010 20th International Conference on Pattern Recognition*, 2010, pp. 2366–2369.

[46] A. Ruospo, G. Gavarini, C. D. Sio, J. Guerrero, L. Sterpone, M. S. Reorda, E. Sanchez, R. Mariani, J. Aribido, and J. Athavale, "Assessing convolutional neural networks reliability through statistical fault injections," in *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2023, [In press].

[47] G. Gavarini, A. Ruospo, and E. Sanchez, "Sci-fi: a smart, accurate and unintrusive fault-injector for deep neural networks," in *2023 European Test Symposium*, 2023, In press.

Appendix 12

XII

M. H. Ahmadilivani, A. Bosio, B. Deveautour, F. F. Dos Santos, J. D. Guerrero-Balaguera, M. Jenihhin, A. Kritikakou, R. L. Sierra, S. Pappalardo, J. Raik, et al., "Special session: Reliability assessment recipes for dnn accelerators," in 2024 IEEE 42nd VLSI Test Symposium (VTS), pp. 1–11, 2024.

Special Session: Reliability Assessment Recipes for DNN Accelerators

Mohammad Hasan Ahmadilivani[1], Alberto Bosio[2], Bastien Deveautour[2], Fernando Fernandes dos Santos[3], Juan-David Guerrero-Balaguera[4], Maksim Jenihhin[1], Angeliki Kritikakou[3], Robert Limas Sierra[4], Salvatore Pappalardo[2], Jaan Raik[1], Josie E. Rodriguez Condia[4], Matteo Sonza Reorda[4], Mahdi Taheri[1], and Marcello Traiola[3]

[1]Tallinn University of Technology, Tallinn, Estonia
[2]Ecole Centrale de Lyon, CPE Lyon, INL, Ecully, France
[3]Univ Rennes, CNRS, Inria, IRISA - UMR 6074, F-35000 Rennes, France
[4]Politecnico di Torino, Turin, Italy

2024 IEEE 42nd VLSI Test Symposium (VTS) | 979-8-3503-6378-4/24/$31.00 ©2024 IEEE | DOI: 10.1109/VTS60656.2024.10538707

***Abstract*—Reliability assessment is mandatory to guarantee the correct behavior of Deep Neural Network (DNN) hardware accelerators in safety-critical applications. While fault injection stands out as a well-established, practical and robust method for reliability assessment, it is still a very time-consuming process. This paper contributes with three recipes for optimizing the efficiency of the reliability assessment: a) hybrid analytical and hierarchical FI-based reliability assessment for systolic-array-based DNN accelerators; b) mixing techniques for the reliability assessment of in-chip AI accelerators in GPUs; c) reliability assessment of DNN hardware accelerators through physical fault injection. The experimental results demonstrate the efficiency of the proposed methods applied to their target DNN HW accelerator platforms.**

***Index Terms*—deep neural networks, approximate computing, fault simulation, error emulation, reliability, resiliency assessment**

I. Introduction

Deep Neural Networks (DNNs) are a powerful tool assisting different aspects of human life, e.g., healthcare, transportation, security, IoT and edge applications [1] [2]. They are characterised by the extremely complex computational kernels (i.e., several giga operations per second) and the high amount of parameters to be transferred from and to the memory (i.e., in the order of gigabytes). To achieve target energy efficiency and high performance, several types of DNN Hardware Accelerators (DNN-HAs) for the execution of DNN kernels were proposed. [3].

Fig. 1 sketches a system-level view of a DNN-HA. The latter is controlled by a microprocessor host and has to access the main memory to retrieve data. The most widely used DNN-HAs can be classified as Systolic Array (i.e., TPU), GPU and FPGA. Independently from the type, employing DNN accelerators in safety-critical applications raises hardware reliability concerns. For example, in compliance with the ISO 26262 functional safety standard for road vehicles, the FIT (Failures In Time) rate of particular hardware components has to be 10 failures in 1 billion hours of operation at maximum to meet the target safety integrity level, which necessitates very circumspect design [4], [5].

*The authors are sorted in alphabetic order.

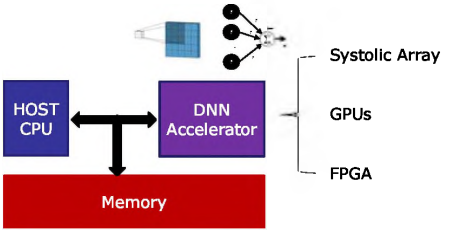


Fig. 1: DNN HW accelerator system

The reliability of DNN accelerators is boosted by their intrinsic ability to function correctly even in the presence of environment-related faults (soft errors, electromagnetic effects, temperature variations) or faults in the underlying hardware (manufacturing defects, process variations, nanoelectronics aging effects) [6]. DNNs are known to be resilient to faults due to their numerous interconnected layers and the ability to mask faults [7] [8]. Unfortunately, assessing the reliability of a DNN accelerator is not a trivial task [9], [10]: it depends on several factors, such as the training set, the data type, and the quality of the test set [11]. On top of that, we need to consider the hardware that performs the computations since specific platforms have specific faults [12].

There are three main methodologies for DNNs' reliability assessment radiation-based, platform-based, and simulation-based [13]:

- **Simulation-based:** The injection process is carried out without relying on the physical device finally running the NN. Moreover, depending on the abstraction level, they can be further ranked.
  - **Software Level:** The injections are performed on a high-level model of the NN, not considering any details of the actual hardware architecture.
  - **Hardware Level:** The injections are performed on a more accurate model of the NN that simulates the target hardware architecture. E.g., this can be described at the register transfer level (RTL) or gate level.
- **Platform-based:** The measurements and the analyses are performed directly on a physical device that emulates the final implementation of a design using FPGAs or on

physical platforms running the NN, e.g., CPUs and GPUs.
- **Radiation-based:** The reliability assessment is performed in the actual platform running the NN under assessment by means of external electromagnetic interference, such as ionizing particle incidence through accelerated radiation test campaigns.

The goal of this paper is to present advanced methods to assess the reliability of the three types of DNN-HAs, i.e., Systolic Array, GPU and FPGA, through different methodologies.

The paper is organized as follows: Section II discusses a hybrid approach for Systolic Array DNN accelerator. Section III-B presents a method for evaluating the impacts of faults in GPU by considering its low-level micro-architecture description and its functional operation. Section IV leverages neutron beam experiments to extract practical – and non-obvious – information about GPUs and FPGA DNN accelerators. Finally, Section V concludes the paper.

II. Hybrid Analytical and Hierarchical FI-based Reliability Assessment for Systolic-Array-Based DNN Accelerators

A. Motivation and Related Work

Simulation-based FI is less expensive in terms of equipment, but at the same time, it implies the most resource-intensive computations and is very time-consuming. On the other hand, analytical and hybrid approaches are proposed to reduce the complexity of exploiting FI for the reliability assessment of DNNs [12]. Analytical approaches attempt to provide mathematical approaches to estimate reliability, nonetheless, their evaluation accuracy is challenging to address and they are mostly hardware-agnostic. Whereas hybrid FI-analytical approaches can take advantage of both FI and analytical approaches in terms of scalability, accuracy and hardware-based analysis [12]. To our knowledge, there is no work assessing the reliability of Systolic Arrays (SAs) running DNNs using hybrid methods to accelerate the analysis process.

This section introduces a novel hybrid analytical and hierarchical simulation-based reliability assessment for systolic-array-based DNN accelerators based on FI. This methodology is tailored to significantly accelerate the fault injection process on systolic-array-based DNN hardware accelerators. The systolic-array core of the DNN accelerators is modeled using a Uniform Recurrent Equation (URE) system [14]. The proposed injection flow has been implemented based on an SA simulator [15], thus offering the advantage of being more precise than a hardware-agnostic tool, yet much faster than traditional RTL-level simulations. An analytical method [16] is used to prune the fault space to further optimize the tool and speed up the reliability assessment process.

B. Proposed method

The methodology for the proposed framework is illustrated in Fig. 2. After providing the trained network parameters and architecture, in Step 1, the fault list is generated. Possible fault locations can be defined by the user or can be a random fault list generated based on the network parameters by the framework. In this work, we consider random transient faults in the registers of the systolic array's processing elements.

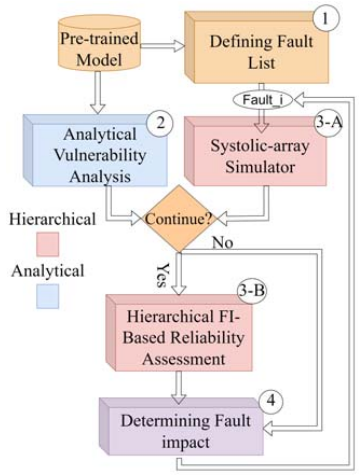


Fig. 2: The proposed methodology for hybrid analytical and hierarchical reliability assessment for systolic array DNN-HAs

In Step 2, to prune the fault space, we adopt and extend the DeepVigor methodology presented in [16] that provides vulnerability analysis for DNNs and QNNs, respectively. To this end, we find error values for each output Feauture Map (FMap) that misclassifies the network output. Let $\delta_k^l(X_i)$ be an added positive or negative error value to an output FMap by a fault in the k-th neuron at layer l with input data X_i. For each neuron, we find the minimum positive and maximum negative $\delta_k^l(X_i)$ that misclassifies the output from the golden classification. This value is obtained for all input data X and aggregated over them. The aggregation leads to a vulnerability value range for each neuron, as shown in Fig. 3. It is labeled as follows:

- **Vulnerable** (red area): if a fault deviates the output of a neuron as in this range, it will certainly lead to misclassification for any input.
- **Non-Vulnerable** (green area): if a fault deviates the output of a neuron as in this range, it will not change the output classification for any input.
- **Semi-vulnerable** (grey area): if a fault deviates the output of a neuron as in this range, it might or might not lead to misclassification for any input.

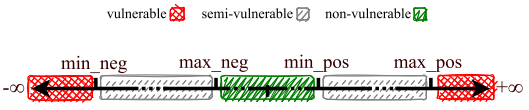


Fig. 3: Vulnerability ranges for fault space pruning

This analysis enables pruning the fault space in a way that we map the deviations at the erroneous outputs of neurons induced by fault to the obtained vulnerability ranges for neurons. *If the error corresponds to the red or green areas, we immediately classify them respectively as critical or non-critical, and do not continue the fault simulation.* Otherwise, if the error corresponds to a semi-vulnerable range, the fault simulation is required to be performed.

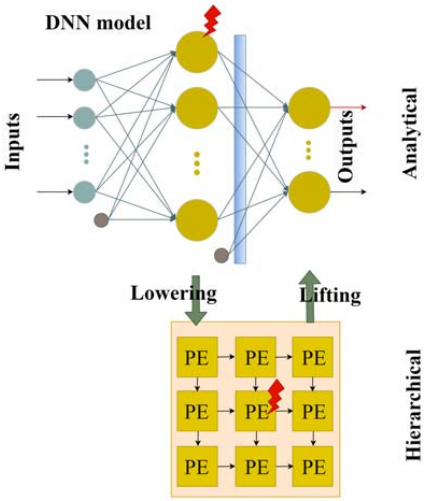


Fig. 4: Fault injection across hierarchical and analytical abstractions

In Step 3-A, the simulation is performed from the beginning of the DNN to the fault's location to obtain the erroneous output of the corresponding neuron. The error is mapped to the vulnerability values of the corresponding neuron (shown in Fig. 3). If the fault is pruned (corresponding to green or red areas), the fault impact is determined (Step 4). Otherwise, the simulation continues (Step 3-B) to obtain the fault impact (Step 4).

In Steps 3-A and 3-B, switching between high-level API and systolic-array simulator is done by solving the URE system mentioned before; this step is described later in this section and Fig. 4. In Step 4, the reliability of the network and the impact of the faults are reported by different metrics. After evaluating the impact of one fault, the next fault is selected for evaluation.

The hardware simulation in Step 3-A is based on a formalization of the problem to solve through a URE system. Such a system can describe the problem to be solved by the architecture through a set of recurrence relations. In our specific case, we are interested in solving the problem of matrix multiplication. We associate the following system to such a task.

$$\begin{aligned} c(i,j,k) &= c(i,j,k-1) + a(i,j-1,k) \times b(i-1,j,k) \\ a(i,j,k) &= a(i,j-1,k) \\ b(i,j,k) &= b(i-1,j,k) \end{aligned}$$

The generalization of this process is described in [14]. This equation system can be associated with actual hardware processing by projecting the iteration space to the physical space. The iteration space contains all the points (i, j, k) of the equation system. The physical space describes where (i.e., which processing element) and when (i.e., at what clock cycle) each computation happens in the real world. To achieve that, the iteration space is projected twice: the first time, the resulting points will correspond to the spatial arrangement of the processing elements; the second projection determines isotemporal planes, identifying operations that are computed during the same clock cycle but on different processing elements;

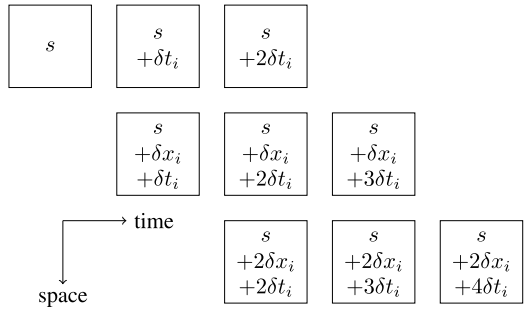


Fig. 5: Fault propagation in systolic array. When injecting element s, the fault is propagated in time (thus affecting elements $s + \delta t_i$ and $s + 2\delta t_i$) and in space (forwarding the faulty value to neighboring elements $s + \delta x_i + \delta t_i$, $s + 2\delta x_i + \delta t_i$ and so on).

each plane corresponds to a different clock cycle. The space-projection matrix P and the temporal dimension vector π are used later.

In order to perform the simulation, it is sufficient to solve the system shown above. Nevertheless, this method gives the possibility of injecting faults in the values in a hardware-aware fashion. To achieve the injection, it is sufficient to change the values a, b and c at specific iterations (i, j, k). The faulty values must then be propagated to the following Processing Elements (PEs). Given the system of equations, the propagation can be computed easily, taking into account the transformation matrix. Figure 5 shows the concept. In this case, an injection in the element $s = (x, y, t)$ on the generic line i is done between times 0 and ∞. The injected elements are visible in the figure. Specifically, the fault will propagate in time, thus injecting also $s + \delta t_i$ and $s + 2\delta t_i$. In the same way, this fault will propagate in the space to the element cascading from s. The value propagation only happens after each clock cycle, which means that the next injected element will be also displaced in time, thus injecting element $s + \delta x_i + \delta t_i$. In the same way, the latter will propagate to the following element on the following clock cycle, thus injecting element $s + 2\delta x_i + 2\delta t_i$ and so on. The injected points can be translated into iteration vectors i, j, k using the inverse transformation. Formally, it is sufficient to consider the injection as a function h applied to one of the three values, e.g. $a(i, j, k) = h(a(i, j - 1, k)$

C. Experimental Results

To evaluate the methodology, experiments were performed using a 16-bit quantized LeNet-5 trained on the MNIST dataset. The network was injected with random transient faults (assuming that they are caused by Single-Event Upsets). More specifically, a fault affects a random bit in the weight register of a random processing element. The fault causes an error, which manifests itself by fixing the value of a bit to either 0 or 1. The target architecture is an Output-Stationary Systolic Array. In this experiment, faults are injected only in the first layer of the network. It is simulated using the URE system to obtain its output values. The output is compared against the vulnerability ranges obtained by the analytical approach, which determines whether the injected fault is vulnerable, semi-vulnerable or non-vulnerable (see Fig. 3). If the fault happens to be semi-vulnerable, the simulation must be carried out until the end to

determine whether the fault produced misprediction or not; in the other two cases, the output is pre-determined, and there is no need to complete the simulation until the end. It is worth mentioning that the vulnerability ranges are obtained in less than one minute on an NVIDIA 3090 GPU. This methodology allows to accelerate the process of fault injection simulations.

The produced fault list includes 964 random faults, each simulated with 100 random input images.

TABLE I: Channel-wise analysis of the fault injection speedup. The last column (su) indicates the percentage of vulnerable and non-vulnerable simulations with respect to the total.

channel	non-vulnerable	semi-vulnerable	vulnerable	speed-up (%)
0	13673	3127	0	81.38
1	12606	3594	0	77.81
2	13044	2956	0	81.52
3	10392	5508	0	65.35
4	11811	3589	0	76.69
5	13173	2927	0	81.82

Table I shows the number of simulations in each category. The simulations are grouped by channel for better understanding. The first column indicates the channel in which the faults are injected. The "non-vulnerable", "semi-vulnerable", and "vulnerable" columns indicate the number of simulations with the corresponding vulnerability. The "speed-up" column indicates the percentage of simulations that do not need to be fully performed because of the vulnerability ranges gathered in the previous step. More specifically, speed-up is the percentage of pruned simulations.

As observed in Table I, no fault was pruned as vulnerable using the analytical approach. This is because the analytical approach did not provide any red area for the first layer of LeNet-5, yet it provides an effective green area in which up to 81.52% of faults are pruned. Table I shows that consistently, for each layer, at least 65% of the simulations are predetermined using the analytical approach leading to a remarkable time-saving for simulations.

If we assume that x is the time needed to perform a single hardware simulation (computing a convolution between a kernel and a feature map), we would need a total time t for a single inference as follows:

$$t = x \times \sum K_{i,j,l} \qquad \forall (i,j,l) \in NN_f \subseteq NN$$

where K is always equal to 1 and indicates a single convolution and (i,j,l) is the triple $(fmap_{input}, fmap_{out}, \text{layer})$ in the neural network NN. NN_f represents subset of the convolutions affected by the fault f.

For example, in our LeNet-5, the set NN has 6 elements for the first layer: $(1,1,1),(1,2,1),\dots(1,6,1)$. The second layer has 6×16 elements: every FMap of the output of the first layer has to be convoluted with every kernel of the second layer, so we will have $(1,1,2),(1,2,2),\cdots(1,16,2),\cdots(2,1,2),\cdots(6,16,2)$. Note that the first layer only has one input FMap: the input image itself, this we only have 6 convolutions in that case. The subset NN_f corresponds to all those convolutions whose input FMap is different than the golden and thus needs to be recomputed, taking into account the errors introduced by the fault. In our case, with the type of fault chosen, we would have an error affecting only one value in the first channel. Such an error is propagated to the following layers, although it may be masked. Especially when considering permanent faults, we would be forced to simulate every convolution until the output to determine the criticality of the fault. In our setup, NN_f would have a total of 102, considering that our systolic array only computes convolutional layers. Using the analytical approach made it possible to interrupt the simulation just after the first layer. In the conducted fault injection campaign, about 76% of the total simulations were pre-determined, allowing us to simulate the systolic array only once in 76% of the total inferences. This gave us an average time per simulation $t' = x \times (0.24 \times 102 + 0.76) \approx 0.24 \times t$ and a total 86% of fault injection speed up.

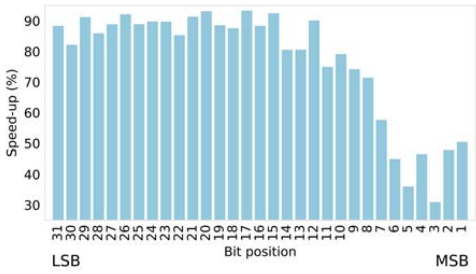


Fig. 6: Speedup per injected bit position

Figure 6 shows the speed-up in percentage concerning the injected bit position. It can be concluded that the most significant bits (0 to 7) are the ones that produce a smaller speed-up in general. Most of these faults fall in the "semi-vulnerable" category, thus it is necessary to propagate the error to infer the fault criticality. The least significant bits (14 to 31), on the other hand, provide great speed-up since the error is small enough for the simulation to be classified as "non-vulnerable".

Finally, the methodology is applied for SDC-1 (i.e., Silent Data Corruption leading to misclassification) analysis.

TABLE II: SDC1 computed over each channel. The second column (misclassified) shows how many "semi-vulnerable" simulations ended up misclassifying the output. The third column (SDC1) shows the metric computed over the whole batch.

channel	misclassified	SDC1 (%)
0	48	0.29
1	50	0.31
2	41	0.26
3	93	0.58
4	69	0.45
5	60	0.37

Using the data in table I, we could compute a lower bound corresponding to the number of "non-vulnerable" simulations. Since there is consistently no "vulnerable" simulation, the lower bound effectively corresponds to the complement of the speed-up. Furthermore, the SDC-1 was calculated for the "semi-vulnerable" faults. Table II shows the metric computed for the network under analysis. The second column shows the number of simulations that ended up in misclassification. The third column shows the metric itself. The network shows great resilience to this type of fault. In total, the SDC1 is 0.37%.

In conclusion, the presented approach is capable of significantly reducing the fault injection simulation time. This

method will be extended to larger DNNs considering various fault models for obtaining reliability evaluation metrics and combining the assessment with selective hardening techniques [17], [18].

III. Mixing techniques for the Reliability Assessment of in-chip AI accelerators in GPUs

A. Motivation and Related Work

Modern GPUs are highly adopted platforms to deploy AI applications, since these devices efficiently exploit the programming flexibility and structural parallelism to speed up the execution of such data-intensive workloads [19]. In particular, modern AI algorithms are characterized by repetitive operations (e.g., convolutions) that are implemented by resorting to convolution mapping algorithms (e.g., GEMM, Winograd, or direct) [20]–[22]. For this purpose, modern generations of GPUs include specially optimized accelerators, called *Tensor Core Units* or TCUs, that resort to compacted 2D/3D arrays of *Dot-Product Units* (DPUs) [23]. TCUs comprise highly regular DPU structures to increase performance and are controlled in the GPU to exploit their implicit multi-threading parallelism [24]. Unfortunately, the vast density of transistors in modern accelerators (e.g., 100 billion) and the characterized amount of data-intensive operations in AI applications (e.g., tens of millions) impede the use of conventional strategies for the fault characterization of structures, as well as the reliability assessment of their running applications [25].

Classical strategies for GPUs, such as simulation- and emulation-based fault analyses, provide fine-grain characterizations and allow the identification of vulnerable structures under focused evaluations [26], [27]. However, evaluations on complete designs and large applications might involve unfeasible evaluation times. Other strategies, including software-based error propagation on GPUs, depend on the error models used, which might induce simplified evaluations and inaccurate analyses [28]. Thus, strategies to effectively characterize fault effects while providing an acceptable trade-off between performance and accuracy are still required.

In particular, the individual adoption of fault characterization strategies is insufficient and unfeasible for large hardware accelerators, such as GPUs with in-chip accelerators (TCUs). Some preliminary works evaluated faults in the TCU's micro-architecture and their propagation effects on the operation's outputs, considering number format impacts [29], and effects on the operand sizes [30]. However, these works did not consider the evaluation of large workloads, such as CNNs. Similarly, other works [31] evaluated the impact of TCUs in mixed-precision operations, indicating that TCUs seem to be more fault-sensitive than equivalent applications without TCUs. Unfortunately, these analyses hardly identified fault-sensitive structures inside TCUs.

Other works resort to software-based error propagation schemes to instrument the GPUs application's code and represent error corruptions in software from faults affecting the underlying system. Unfortunately, the accuracy of the method directly depends on the targeted units (mostly data path units) and the available error models to represent faults. In literature, some works [32] addressed the software-based characterization of large workloads (CNNs) in GPUs when errors affected functional units under software error models limited to random bit-flips. Authors in [33] analyzed the impact of errors in CNNs and developed error models to represent corruptions on applications, but neglecting the fine-grain micro-architecture of the underlying hardware. In [34], the authors explored a hybrid strategy to represent software errors from faults in GPU controllers. Unfortunately, their analyses were limited to a few structures with considerable evaluation times.

This work proposes a clever mixing method to effectively assess the reliability of TCUs in GPUs for permanent faults, considering the low-level micro-architecture description and their functional operation to determine error patterns, which are then used in the evaluation of large CNN workloads. Our method combines three strategies to provide an affordable trade-off between accuracy and performance evaluation.

B. Assessing the reliability of in-chip GPU accelerators

The proposed reliability assessment method for the TCUs in GPUs combines three strategies to provide accuracy while allowing feasible evaluation times when GPUs execute large applications, such as CNNs, as depicted in Figure 7. The first strategy (*focused low-level micro-architecture fault evaluation*) resorts to fine-grain evaluations of the main structures in TCUs (DPUs) in order to accurately characterize the fault propagation effects on the scalar operations in TCUs. Then, a *functional evaluation* characterizes fault effects on the array operations of TCUs, considering their interaction with other GPU structures. It must be noted that TCUs' execution depends on GPU's scheduling and are operated through GPU's machine instructions. Moreover, TCU cores are sequentially reused to operate matrix's fragments. Thus, the functional characterization allows the identification of accurate spatial fault propagation patterns during the complete execution of the TCUs. Finally, those scalar impacts and the array fault propagation patterns are combined to represent software errors effects on applications. In this case, we exploit an *application-level* error propagation strategy by using software-based error injection schemes to instrument the application's code (e.g. CNNs) with the identified error effects. The combination of both strategies (error patterns + software-based injection) allows the reliability assessment of large applications with accuracy under feasible times. The next subsections describe each strategy for the TCU's assessment.

1) Focused micro-architecture evaluation: consist of a low-level micro-architecture (Gate-level) evaluation of a DPU only focusing on its scalar operations and the fault propagation effects on the results. We use a commercial-grade logic simulator tool (*ModelSim* by *Siemens EDA*) to characterize the fault-free operation of the DPU with typical workloads (e.g., matrix tiles). The output values are used for comparison in the identification of fault effects. Then, a functional safety simulator (*ZOIX* by *Synopsis*) is adapted to exhaustively evaluate the propagation effects of permanent (stuck-at) faults in the DPU. In the characterization, one hardware fault is placed in the DPU and then all operations are independently evaluated, so allowing the identification of faults prone to corrupt individual DPU operations. We used normally distributed random matrix tiles with values in ranges from ± 1.0 to ± 100.0 to represent CNN operations [35]. The output results are then evaluated to identify scalar corruption patterns. This analysis also allows

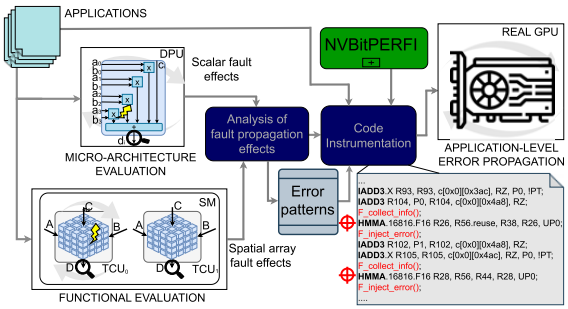


Fig. 7: A general scheme of the methodology for fault characterization of TCUs in GPUs.

us to identify vulnerable structures to faults for the further development of countermeasures and mitigation mechanisms.

The fault effects are categorized based on their impact on scalar results as follows: *i)* Silent Data Corruptions (SDCs) occurs when faults in the DPU alter the results. *ii)* Detected Unrecoverable Errors (DUEs) refer to faults that halt, collapse, or hang the DPU's operation, including faults generating Not-a-Number (NaN) and infinite (Inf) values, and *iii)* Masked effects encompass benign effects that leave the DPU's results unchanged despite the presence of faults.

2) Functional evaluation: consist on the characterization of the spatial corruption effects on the output arrays (matrix tiles) by faults in the TCUs. This evaluation aims to analyze the operational features in TCUs to process large matrix tiles, since TCU's are reused and managed by machine instructions. Moreover, this assessment considers the interaction with other GPU structures (i.e., schedulers and register files).

We resort to an instruction-accurate architectural simulator of the TCUs in GPUs (*PyOpenTCU*) [36] that integrates the scheduling and the memories as in real TCUs. A pin-level fault injector tool in *PyOpenTCU* supports the evaluation of hardware faults on the TCU's architecture. The injector tool places one or more faults in the inputs, outputs, or internal structures of the TCUs. Then, a complete set of input stimuli (matrix tiles) is evaluated using the set of sequential machine instructions for TCUs and the distribution of input operands among the threads and warps in the GPU. After the fault evaluation of the TCUs, the output results are compared with a fault-free operation using the same input stimuli. In our evaluation, we employ the same stimuli used in the micro-architecture evaluation from representative CNNs. The corruption effects are classified as SDCs, DUEs and masked effects.

3) Application-level error propagation: consist on the characterization of fault propagation effects on complete applications by resorting to a combined representation in software of the identified error patterns from the micro-architecture and functional evaluations of the TCUs (i.e., scalar and spatial errors patterns as *Error Masks*). Thus, *Error Masks* represent software errors by corrupting operations, and values are used to better describe the impacts of hardware faults while allowing acceptable evaluation times for large applications.

We adapt a Hardware-Injection through Program Transformation (HIPT) framework [37] (*Nvbit* [38]) to instrument the code of parallel programs for GPUs and inject software errors as permanent faults from the TCU's hardware. Our tool, called

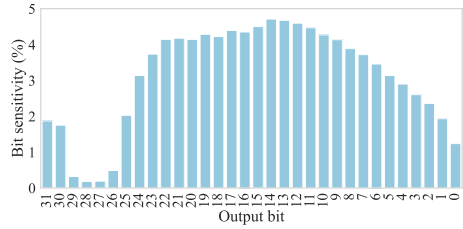


Fig. 8: Fault sensitivity on the outputs of the DPU operations.

NvBITPERFI [34], represents errors by instrumenting the application's assembly code controlling the TCU cores (e.g., *MMA*) with custom functions that apply the previously identified scalar and spatial error patterns. The scalar error patterns are injected to corrupt output values to mimic the equivalent effect of a faulty TCU. In addition, the spatial error patterns are used to indicate the number of corrupted threads and warps, as well as the spatial distribution of the corruptions.

The error propagation evaluation starts with the execution of an error-free operation for the complete application (e.g., *CNN*) on a real GPU. The overall results are collected and stored as a reference for comparison. Then, *NvBITPERFI* instruments the application's code with one error pattern and starts its execution, collecting the output results for corruption classification. It is worth noting that the instrumented functions are used each time the TCUs are used in the application. The evaluation is repeated, injecting each one of the identified error patterns, and the error corruptions in the application are evaluated by comparing the results from the execution with errors and the reference one.

C. Experimental Results

For the experiments, we evaluated the typical configuration of a GPU with two TCUs per SM and 16 DPUs per TCU. In detail, the DPU operates four 16-bit wide inputs. The synthesis of the micro-architecture DPU resorts to a 15nm open-source technology library [39]. Furthermore, *PyOpenTCU* is configured to compute 16x16 input matrix tiles per warp, so a complete matrix tile requires a sequence of 8 *MMA* instructions.

For the fine-grain and functional fault characterizations, we used four typical input matrix tiles that account for a total of 4,096 input stimuli for the DPU core. One exhaustive fault injection campaign evaluates 82,912 faults on the DPU for each individual input stimulus. Similarly, four fault injection campaigns perform the exhaustive functional evaluation of the TCUs by injecting an overall of $2.29x10^5$ hardware faults (57,344 faults per campaign). Then, for the evaluation of the error propagation on the applications, we resort to three typical CNNs (*ResNet50*, *AlexNet*, and *MobileNetv3*) on the *ImageNet* data set by injecting 1,024 errors on the TCU core per CNN. These CNN models are deployed on TensorRT with TCU acceleration using FP16 data computations. All experiments were performed on a workstation HP Z2 G5 with an Intel Core i9-10800 CPU with 20 cores, 32BG of RAM, and one NVIDIA Ampere 3070ti GPU. In our experiments, we targeted the TCU_0 inside the SM_0 for the functional evaluation and the error propagation on a real GPU. In the end, the evaluation required around 415.6 hours, which accounts for 264hrs of fine-grain evaluation of the DPU, 128hrs of functional evaluation of the TCUs, and 23.6 hours for the error propagation on applications.

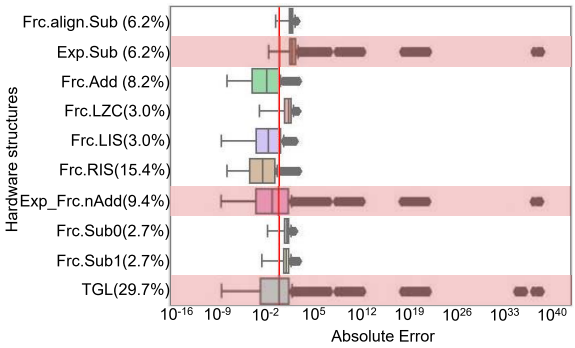


Fig. 9: Distribution of errors among the structures of the ADD inside the DPU.

First, we analyze the fine-grain fault impacts on the scalar operations on the DPU. The results indicate that a considerable percentage of faults (around 87%) are prone to propagate and cause corruptions on the output results as SDCs, while other faults are masked (13%). Our evaluation did not identify scalar DUE effects, which can explained when considering that DPU units are used in the data path of a GPU. In addition, a cautious analysis of the results reveals that most DPU output corruptions mostly caused one bit-flip (around 46%), two bit-flips (about 18%) and three bit-flips (in around 8%). Then, we analyzed the scalar impacts as errors corrupting bit-fields on the DPU operations. Figure 8 illustrates the distribution of errors on the output bit-fields, indicating that most effects in the DPU operations are corruptions on the mantissa part (bits 22-13) and the lower part of the exponents (bits 30, 25-23). Interestingly, corruptions in the exponent fields (around 7% of all observed errors) are responsible for large-error magnitudes and prone to propagate effects on the application. In contrast, corruptions in the mantissa or fraction (around 93% of errors) only caused effects with magnitudes lower than 1.0 and might be neglected as critical error sources in most CNN scenarios. An additional evaluation reveals that infrastructures calculating the exponent of an operation for the sub-units of addition (*ADD* or adder tree) and multiplication (*MUL*) in the DPU (around 14.6% of the DPU's area) are highly vulnerable and prone to cause large-magnitude errors, as depicted in the distribution of errors (*Exp.Sub*, *Exp_Frc.nAdd*, and *TGL*) for the DPU's ADD in Figure 9. We observed that for both structures (ADD and MUL) produced similar error distributions with around 48% of the corruptions caused by the ADD and around 13% produced by faults inside any of the 4 MUL cores in the DPU.

The functional evaluation aims to indicate the distribution of spatial corruption effects among the DPUs in the TCU. Figure 10 reports the error patterns and their spatial distribution for an output matrix tile (16x16). The results indicate that a faulty DPU in a TCU can corrupt from 1 to 8 output elements. The spatial distribution of errors in terms of relative distance between corrupted elements, and the localized effects (i.e., left-part of the output) is explained by a deterministic distribution of matrix fragments to operate among the TCUs and its scheduling policy, respectively. An analysis of the intermediate results (during the *MMA* instruction's execution) reveals that the sequential use of TCUs contributes to accumulate errors and increases in one the number of corrupted bits on the output value.

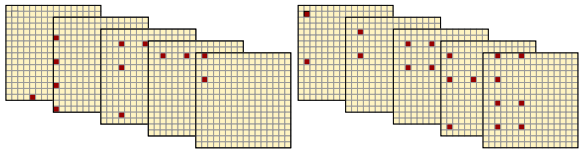


Fig. 10: Distribution of output's corruptions by a faulty TCU.

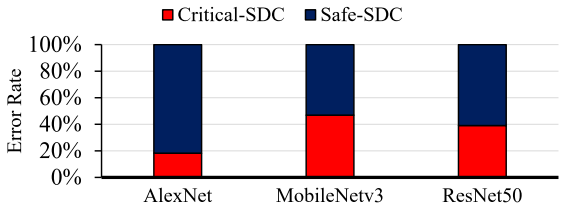


Fig. 11: Corruption effects on the CNNs by TCU hardware faults.

Finally, both sets of error patterns (*scalar* and *spatial*) are used to instrument the applications. This evaluation considers one error injected per application execution. Thus, an error (i.e., error mask) is only applied on those threads using the instructions to access a targeted unit (TCU_0). Figure 11 illustrates the fault rate on the outputs of the evaluated CNNs. The results indicate that all the application level errors induce Silent Data Corruption (SDC) effects, but a considerable percentage of error effects (from 18% to 45%) critically caused wrong outcomes at the CNN's outputs. These results suggest that accumulated errors from the reuse of faulty TCUs are highly prone to corrupt several operations in CNNs and propagate their effects, so increasing the probability of data corruption on the outputs. Since errors affect several elements in the layer's outputs, the implicit resilience of a CNN is affected by these corruption effects. In fact, we identified that during the inference of one image, around 25% of the kernels are reused, and from those kernels, around 80% rely on the TCU computations. These features of the CNN implementation cause that a faulty TCU corrupt multiple layers by the accumulation of error during the CNN operation. Indeed, the results show that the average error activation rate reaches ≈90,000, ≈ 300,000, and ≈1 million for *AlexNet*, *ResNet50*, and *MobileNet*, respectively.

An analysis of the workload's distribution in the GPU shows that each kernel uses fixed configurations to distribute their tasks among the SMs and use the TCUs. The workload distribution in the GPU influences the error propagation when a faulty TCU is in the system, which indicates that a clear understanding of micro-architecture and the algorithms (i.e., convolutions) used to map CNNs in the GPU's TCUs are required to analyze and develop effective countermeasures and mitigation mechanisms correctly. The other parameter contributing to the error propagation in CNNs is the total amount of SMs; since the CNNs are distributed among all TCUs in a system, their error vulnerability is proportionally inverse to the number of TCUs and SMs in a system.

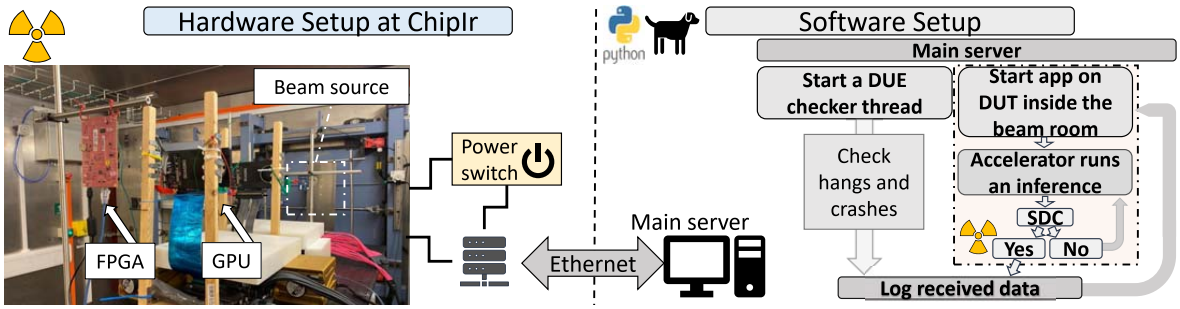


Fig. 12: Beam setup mounted at ChipIR beam line and the software setup.

IV. Reliability Assessment of DNN Hardware Accelerators Through Physical Fault Injection

A. Motivation and Related Work

Evaluating the sensitivity of DNN accelerators for transient events is crucial for identifying errors that compromise DNN accuracy. Fault simulation allows researchers to pinpoint specific fault sites and track how faults propagate through both the hardware and the application [37], [40], [41]. This can be implemented at various levels of abstraction. Low-level hardware fault simulations (e.g., gate level) offer detailed insights but are time-consuming, while higher-level hardware or software simulations (e.g., microarchitectural or software) provide faster but less accurate results. Additionally, selecting appropriate fault models for simulation is critical to prevent inaccurate conclusions [42]. In contrast, physical fault injection using radiation exposes the system to a beam of ionizing particles, offering a means to estimate realistic error rates. However, this method does not support tracking fault propagation, as faults are identified only when they lead to failures [43], [44].

By performing radiation experiments, recent studies have demonstrated that DNNs accelerators are highly susceptible to transient faults induced by radiation [31], [45]–[47]. Recent research data suggests that while GPUs are mostly affected because of the high amount of available resources [48], [49] and the possibility of having multiple output elements corrupted, which undermines DNN reliability [46], [50], FPGAs, are mostly affected due to the programable hardware characteristics, where a transient fault can change the configuration memory and change the circuit [51]–[53].

In this section, we show how we can extract practical – and non-obvious – information from neutron beam experiments with GPUs and FPGAs running DNNs.

B. Neutron Beam Experiments

Our experiments were performed at the ChipIR facility within the Rutherford Appleton Laboratory in the UK. The setup within the ChipIR facility is depicted in Figure 12. ChipIR delivers a neutron beam with an energy spectrum similar to atmospheric neutrons [54]. The neutron flux available reaches approximately $3.5\times10^6 n/(cm^2/s)$, which is nearly eight orders of magnitude greater than the terrestrial flux at sea level (13 $neutrons/(cm^2 \cdot h)$ [55]).

Considering the low terrestrial neutron flux, encountering more than one corruption during program execution in a realistic application is unlikely. Our experiments are meticulously designed to uphold this characteristic, ensuring observed error rates remain below one failure per 2,000 executions. Each DNN configuration undergoes testing for a minimum of 3 effective hours, excluding setup, result checking, initialization, and recovery from DUE time.

1) System Under Test: We selected the TUL PYNQ-Z2 board based on the 28nm Xilinx Zynq-7000 SoC and the NVIDIA Tesla V100 GPU based on a 12nm Volta microarchitecture as hardware platforms.

For the GPU experiments, we target an evaluation to measure the reliability of multiple floating-point precisions of a General Matrix Multiplication (GEMM), such as FP16, FP32, and FP64 from the cuBLAS libraries [56]. The choice of multiple GEMM configurations is guided as they are the core of the state-of-the-art DNNs. As a study case, we also exposed an object detection DNN, YOLOv3 [57], with two precisions, FP16 and FP32, with the same setup as used in [58].

For the FPGA experiments, we created five ANN accelerators using the HLS4ML tool [59]. HLS4ML is an open-source tool that can translate machine learning models described with libraries such as Pytorch and TensorFlow into HLS-compatible high-level-description parametric models, which can be configured, synthesized, and implemented to run on an FPGA. We used a 2-layer ANN to classify handwritten digits on the MNIST dataset and varied the number of multipliers used per layer. Table III shows the number of multipliers used for each evaluated FPGA configuration. As the number of multipliers increases, the clock cycles decrease, but the area also increases. As discussed in the next section, a larger area may increase the failure rate.

2) Experimental Setup: Figure 12 presents an overview of the experimental setup. We developed a software watchdog composed of Python scripts, which operates on the server computer located externally to the beam room. This watchdog oversees the DUT by monitoring its activity, executing programs,

TABLE III: Configurations generated by HLS4ML by varying the number of multipliers used in each layer. 39,200 multiplications are performed in the first layer and 500 in the second.

	Multipliers Used (layer 1× layer 2)	Clock Cycles
Opt_1	1x1	39,804
Opt_2	2x2	19,960
Opt_3	10x5	4,154
Opt_4	50x20	947

logging events, and effectively recovering from any device malfunctions. When the program becomes unresponsive, it is terminated and restarted based on a predefined timeout period. Notably, the timeout duration for each code is meticulously set, extending up to 10× the anticipated execution time, tailored to the specific requirements of the code's execution duration.

For every DUT configuration, we designate a host system that is responsible for managing the DNN accelerator hardware. For the GPU setup, the host system is an x86 Intel i3 CPU, whereas for FPGA SoC configurations, the ARM CPU on the PYNQ SoC is employed. Following each iteration performed within the device, the host compares the kernel's output against a predefined constant golden value. Any discrepancy between the actual output and the golden value prompts the host to relay pertinent information back to the server outside the beam room, before initiating a restart of the process. It is important to emphasize that our error rate analysis only considers discrepancies observed in the kernel output.

The experiments conducted at ChipIR allow the estimation of the *Failure In Time (FIT)* rates, representing the error rate of a given accelerator. FIT rate is directly derived from the application *cross-section* (σ), calculated by dividing the number of observed errors by the received particle fluence ($neutrons/cm^2$). This cross-section is then multiplied by the $flux$ under which the system operates to yield the FIT rate.

While the FIT rate is valuable for estimating system failure rates, it overlooks the execution time variations stemming from different HLS and GPU hardware optimizations. To address this, we compute the *Mean Executions Between Failures (MEBF)* to gauge how many correct executions can be performed before a failure occurs, as defined by equation 1.

$$MEBF = \frac{MTBF}{ExecutionTime} \tag{1}$$

Where the $MTBF$ is the inverse of the cross-section multiplied by the $flux$ under which the device will operate (i.e., $MTBF = FIT^{-1} = (\sigma * flux)^{-1}$), and the $ExecutionTime$ is the application execution time.

C. Experimental Results

Figure 13 shows experimentally measured FIT rate and MEBF. Both FIT and MEBF are presented relative to the configuration exhibiting the worst performance for each code. Specifically, we present the experimental data relative to FP64 GEMM and FP32 YOLOv3 for the GPU and Opt_1 for the FPGA. The data in Figure 13 is presented with a 95% confidence interval on a Poisson distribution.

1) FIT: Smaller precision results in a lower overall FIT rate for both applications evaluated on GPUs (Figure 13a). Specifically, the FIT rate of FP32 is 21.5% lower than that of FP64, while the FIT rate of FP16 FIT is 38.43% lower than that of FP16. Similarly, for YOLOv3, the FIT rate of FP16 is 46.29% lower than that of FP32. Using smaller float precisions reduces the probability of a fault that leads to a failure (SDC or DUE) as it uses fewer hardware resources. Furthermore, compared to the FP32 version, the FP16 version of YOLOv3 results in fewer critical SDCs (misdetections) with 21.43% and 27.27%, respectively. The lower representation capacity of FP16 reduces the chances of very large corrupted values, which are known to impact the reliability of DNNs [45], [60], thus changing the detection result of YOLOv3.

The FPGA FIT rate exhibits different behavior compared to the GPU results. The Opt_2 FIT rate decreases compared to Opt_1, while the same does not happen for Opt_3 and Opt_4. Specifically, the FIT rate for Opt_2 is 82.81% of Opt_1, while the FIT rates for Opt_3 and Opt_4 are 98.35% and 107.74% of Opt_1 FIT rate, respectively. It is important to note that no built-in fault detection functionalities, such as scrubbing and/or the activation ECC, were utilized in any of the experiments. As a result, smaller accelerators running for longer might accumulate errors in the extensively reused hardware, and since they experience less DUE stopping the execution, faults can propagate to outputs, leading to more SDCs. Additionally, the DUE FIT increases for larger accelerators. The larger the accelerator area (i.e., more multipliers per layer), the higher the chance that a bit flip corrupts a critical circuit resource, leading to the application's hang or crash.

The Critical SDCs for the FPGAs experiments correspond to 18.26%, 21.92%, 28.52%, and 28.81% of the SDC FIT for Opt_1, Opt_2, Opt_3, and Opt_4, respectively. The chance of experiencing critical SDCs is higher for larger accelerators than smaller ones due to the usage of hardware resources. The amount of work done remains constant, but larger accelerators have more area to improve performance, which leads to a higher probability of critical computation at any given time.

2) MEBF: When comparing the ANN accelerators, relying on the FIT rate alone is insufficient, as it does not provide information on the application's execution time [61]. Therefore, we must correlate error rates with performance to make the comparison more holistic.

Figure 13b shows the MEBF for the different configurations we evaluate. For the GEMM codes on GPU, we used all SDCs and DUEs to calculate the MEBF, while for YOLOv3 and MNIST ANN, we used the Critical SDCs (misclassification or misdetection) and the DUE to calculate the MEBF. By using only the Critical SDCs (+DUEs) to calculate the MEBF, we can quantify the failures that will actually negatively affect the accelerator utilization (i.e., crash or wrong classification) if no fault mitigation is adopted.

Unsurprisingly, the GPU MEBF always increases when comparing the larger precisions GEMM FP64 and YOLOv3 FP32 to the lower precisions versions. For instance, the GPU MEBF for GEMM is 2.52× higher for FP32 and 7.33× higher for FP16 than FP64. Also, the YOLOv3 FP16 produces 1.89× more correct predictions than the FP32 version. Optimized hardware not only delivers higher performance but also ensures more correct operations than slower hardware.

The results of the FPGA configurations show a significant improvement trend from Opt_1 to Opt_4, in contrast to the FIT rate results. The MEBF demonstrates a linear correlation with the reuse parameters, and compared to Opt_1, it increases by 2.72× for Opt_2, 6.35× for Opt_3, and 26.25× for Opt_4. In general, the data suggests that fast and large accelerators are capable of performing more inferences between two failures than slow and small ones. However, when they fail, they are more likely to experience a critical SDC.

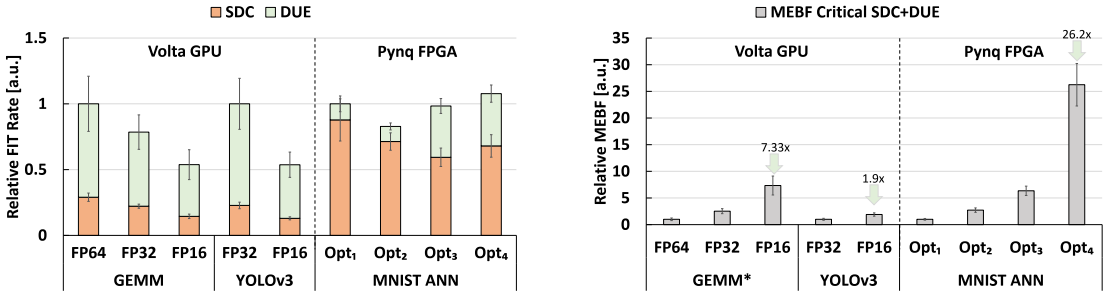


Fig. 13: Experimentally measured relative SDC and DUE Failure In Time (FIT) rates, as well as the relative Mean Executions Between Failure (MEBF). FIT and MEBF values are expressed relative to the configuration exhibiting the worst performance. *While for the GEMM code, we consider all the SDCs on the MEBF calculation, in the YOLOv3 and MNIST ANN, we consider only the SDCs classified as a misclassification or misdetection.

V. Conclusions

The paper presents advanced methods to assess the reliability of the three types of DNN-HAs, i.e., Systolic Array, GPU and FPGA, through different tailored methodologies. These are a) hybrid analytical and hierarchical FI-based reliability assessment for systolic-array-based DNN accelerators; b) mixing techniques for the reliability assessment of in-chip AI accelerators in GPUs; c) reliability assessment of DNN hardware accelerators through physical fault injection. The experimental results demonstrated the efficiency of the proposed methods applied to their target DNN HW accelerator platforms.

Acknowledgments

This work was supported in part by the Estonian Research Council grant PUT PRG1467 “CRASHLES”, by Estonian-French PARROT project “EnTrustED”, by the ANR-21-CE24-0015 “RE-TRUSTING” project, and by the National Resilience and Recovery Plan (PNRR) through the National Center for HPC, Big Data and Quantum Computing.

References

[1] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, “Efficient processing of deep neural networks: A tutorial and survey,” *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.

[2] M. Taheri, “Dnn hardware reliability assessment and enhancement,” in *27th IEEE European Test Symposium (ETS)*, 2022.

[3] *Artificial Intelligence and Hardware Accelerators*. Springer International Publishing, 2023. [Online]. Available: http://dx.doi.org/10.1007/978-3-031-22170-5

[4] A. Nardi and A. Armato, “Functional safety methodologies for automotive applications,” in *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2017, pp. 970–975.

[5] M. Jenihhin, M. S. Reorda, A. Balakrishnan, and D. Alexandrescu, “Challenges of reliability assessment and enhancement in autonomous systems,” in *2019 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, 2019, pp. 1–6.

[6] M. Shafique, M. Naseer, T. Theocharides, C. Kyrkou, O. Mutlu, L. Orosa, and J. Choi, “Robust machine learning systems: Challenges, current trends, perspectives, and the road ahead,” *IEEE Design & Test*, vol. 37, no. 2, pp. 30–57, 2020.

[7] A. Bosio, P. Bernardi, A. Ruospo, and E. Sanchez, “A reliability analysis of a deep neural network,” in *2019 IEEE Latin American Test Symposium (LATS)*. IEEE, 2019, pp. 1–6.

[8] M. Taheri, N. Cherezova, M. S. Ansari, M. Jenihhin, A. Mahani, M. Daneshtalab, and J. Raik, “Exploration of activation fault reliability in quantized systolic array-based dnn accelerators,” *arXiv preprint arXiv:2401.09509*, 2024.

[9] M. H. Ahmadilivani, M. Barbareschi, S. Barone, A. Bosio, M. Daneshtalab, S. D. Torca, G. Gavarini, M. Jenihhin, J. Raik, A. Ruospo, E. Sanchez, and M. Taheri, “Special session: Approximation and fault resiliency of dnn accelerators,” in *2023 IEEE 41st VLSI Test Symposium (VTS)*, 2023, pp. 1–10.

[10] M. Taheri, M. Riazati, M. H. Ahmadilivani, M. Jenihhin, M. Daneshtalab, J. Raik, M. Sjōdin, and B. Lisper, “Deepaxe: A framework for exploration of approximation and reliability trade-offs in dnn accelerators,” in *24th International Symposium on Quality Electronic Design*. https://doi.org/10.48550/arXiv.2303.08226, 2023.

[11] M. Taheri *et al.*, “Appraiser: Dnn fault resilience analysis employing approximation errors,” in *2023 26th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS)*, 2023, pp. 124–127.

[12] M. H. Ahmadilivani *et al.*, “A systematic literature review on hardware reliability assessment methods for deep neural networks,” *ACM Computing Surveys*, vol. 56, no. 6, pp. 1–39, 2024.

[13] A. Ruospo *et al.*, “A survey on deep learning resilience assessment methodologies,” *Computer*, vol. 56, no. 2, pp. 57–66, 2023.

[14] P. Quinton, “Automatic synthesis of systolic arrays from uniform recurrent equations,” *ACM SIGARCH Computer architecture news*, vol. 12, no. 3, pp. 208–214, 1984.

[15] M. Taheri, M. Daneshtalab, J. Raik, M. Jenihhin, S. Pappalardo, P. Jimenez, B. Deveautour, and A. Bosio, “Saffira: a framework for assessing the reliability of systolic-array-based dnn accelerators,” 2024.

[16] M. H. Ahmadilivani, M. Taheri, J. Raik, M. Daneshtalab, and M. Jenihhin, “Deepvigor: Vulnerability value ranges and factors for dnns’ reliability assessment,” in *2023 IEEE European Test Symposium (ETS)*. IEEE, 2023, pp. 1–6.

[17] M. Taheri *et al.*, “Adam: Adaptive fault-tolerant approximate multiplier for edge dnn accelerators,” 2024.

[18] M. H. Ahmadilivani, M. Taheri, J. Raik, M. Daneshtalab, and M. Jenihhin, “Enhancing fault resilience of qnns by selective neuron splitting,” in *2023 IEEE 5th International Conference on Artificial Intelligence Circuits and Systems (AICAS)*. IEEE, 2023, pp. 1–5.

[19] B. Dally, “Hardware for deep learning,” in *IEEE Hot Chips 35 Symposium (HCS)*. IEEE Computer Society, 2023, pp. 1–58.

[20] X. Liu, J. Pool, S. Han, and W. J. Dally, “Efficient sparse-winograd convolutional neural networks,” *CoRR*, vol. abs/1802.06367, 2018.

[21] A. Vasudevan, A. Anderson, and D. Gregg, “Parallel multi channel convolution using general matrix multiplication,” in *2017 IEEE 28th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, 2017, pp. 19–24.

[22] A. Lavin and S. Gray, “Fast algorithms for convolutional neural networks,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[23] B. R. Boswell *et al.*, “Generalized acceleration of matrix multiply accumulate operations,” Jul. 2 2019, u.S. Patent 10,338,919.

[24] M. Raihan *et al.*, “Modeling deep learning accelerator enabled gpus,” in *IEEE Int. Symp. on Performance Analysis of Systems and Software (ISPASS)*, mar 2019, pp. 79–92.

[25] J. E. R. Condia, J.-D. Guerrero-Balaguera, F. F. Dos Santos, M. S. Reorda, and P. Rech, “A multi-level approach to evaluate the impact of gpu

permanent faults on cnn's reliability," in *2022 IEEE International Test Conference (ITC)*, 2022, pp. 278–287.

[26] J. E. R. Condia, F. F. dos Santos, M. S. Reorda, and P. Rech, "Combining architectural simulation and software fault injection for a fast and accurate cnns reliability evaluation on gpus," in *2021 IEEE 39th VLSI Test Symposium (VTS)*, 2021, pp. 1–7.

[27] F. F. d. Santos, J. E. R. Condia, L. Carro, M. S. Reorda, and P. Rech, "Revealing gpus vulnerabilities by combining register-transfer and software-level fault injection," in *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2021, pp. 292–304.

[28] G. Papadimitriou and D. Gizopoulos, "Demystifying the system vulnerability stack: Transient fault effects across the layers," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, 2021, pp. 902–915.

[29] R. L. Sierra, J.-D. Guerrero-Balaguera, J. E. Rodriguez Condia, and M. Sonza Reorda, "Analyzing the impact of different real number formats on the structural reliability of tcus in gpus," in *2023 IFIP/IEEE 31st International Conference on Very Large Scale Integration (VLSI-SoC)*, 2023, pp. 1–6.

[30] R. Limas Sierra, J.-D. Guerrero-Balaguera, J. E. R. Condia, and M. Sonza Reorda, "Exploring hardware fault impacts on different real number representations of the structural resilience of tcus in gpus," *Electronics*, vol. 13, no. 3, 2024.

[31] P. M. Basso, F. F. d. Santos, and P. Rech, "Impact of tensor cores and mixed precision on the reliability of matrix multiplication in gpus," *IEEE Transactions on Nuclear Science*, vol. 67, no. 7, pp. 1560–1565, 2020.

[32] J.-D. Guerrero-Balaguera, R. L. Sierra, and M. S. Reorda, "Effective fault simulation of gpu's permanent faults for reliability estimation of cnns," in *2022 IEEE 28th International Symposium on On-Line Testing and Robust System Design (IOLTS)*, 2022, pp. 1–6.

[33] C. Bolchini, L. Cassano, A. Miele, and A. Toschi, "Fast and accurate error simulation for cnns against soft errors," *IEEE Transactions on Computers*, vol. 72, no. 4, pp. 984–997, 2023.

[34] J. D. Guerrero Balaguera, J. E. R. Condia, F. Fernandes Dos Santos, M. Sonza Reorda, and P. Rech, "Understanding the effects of permanent faults in gpu's parallelism management and control units," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '23, 2023.

[35] D. Mallasén, R. Murillo, A. A. D. Barrio, G. Botella, L. Piñuel, and M. Prieto-Matias, "Percival: Open-source posit risc-v core with quire capability," *IEEE Transactions on Emerging Topics in Computing*, vol. 10, no. 3, pp. 1241–1252, 2022.

[36] R. L. Sierra, J.-D. Guerrero-Balaguera, J. E. R. Condia, and M. Sonza Reorda, "Analyzing the impact of different real number formats on the structural reliability of tcus in gpus," in *2023 IFIP/IEEE 31st International Conference on Very Large Scale Integration (VLSI-SoC)*, 2023, pp. 1–6.

[37] T. Tsai, S. K. S. Hari, M. Sullivan, O. Villa, and S. W. Keckler, "Nvbitfi: Dynamic fault injection for gpus," in *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2021, pp. 284–291.

[38] O. Villa, M. Stephenson, D. Nellans, and S. W. Keckler, "Nvbit: A dynamic binary instrumentation framework for nvidia gpus," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '52, 2019, p. 372–383.

[39] M. Martins, J. M. Matos, R. P. Ribas, A. Reis, G. Schlinker, L. Rech, and J. Michelsen, "Open cell library in 15nm freepdk technology," in *Proceedings of the 2015 Symposium on International Symposium on Physical Design*, 2015, p. 171–178.

[40] J. E. R. Condia, B. Du, M. S. Reorda, and L. Sterpone, "Flexgripplus: An improved gpgpu model to support reliability analysis," *Microelectronics Reliability*, vol. 109, p. 113660, 2020.

[41] A. Mahmoud *et al.*, "Pytorchfi: A runtime perturbation tool for dnns," in *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*. IEEE, 2020, pp. 25–31.

[42] G. Papadimitriou and D. Gizopoulos, "Demystifying the System Vulnerability Stack: Transient Fault Effects across the Layers," in *IEEE ISCA*, 2021, p. 902–915. [Online]. Available: https://doi.org/10.1109/ISCA52012.2021.00075

[43] H. Quinn, "Challenges in testing complex systems," *IEEE Transactions on Nuclear Science*, vol. 61, no. 2, pp. 766–786, 2014.

[44] P. R. Bodmann, G. Papadimitriou, R. L. R. Junior, D. Gizopoulos, and P. Rech, "Soft error effects on arm microprocessors: Early estimations versus chip measurements," *IEEE Transactions on Computers*, vol. 71, no. 10, pp. 2358–2369, 2021.

[45] G. Li, S. K. S. Hari, M. Sullivan, T. Tsai, K. Pattabiraman, J. Emer, and S. W. Keckler, "Understanding error propagation in deep learning neural network (dnn) accelerators and applications," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2017, pp. 1–12.

[46] Y. Ibrahim, H. Wang, J. Liu, J. Wei, L. Chen, P. Rech, K. Adam, and G. Guo, "Soft errors in dnn accelerators: A comprehensive review," *Microelectronics Reliability*, vol. 115, p. 113969, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0026271420308003

[47] R. L. Rech Junior, S. Malde, C. Cazzaniga, M. Kastriotou, M. Letiche, C. Frost, and P. Rech, "High energy and thermal neutron sensitivity of google tensor processing units," *IEEE Transactions on Nuclear Science*, vol. 69, no. 3, pp. 567–575, 2022.

[48] D. Tiwari, S. Gupta, J. Rogers, D. Maxwell, P. Rech, S. Vazhkudai, D. Oliveira, D. Londo, N. DeBardeleben, P. Navaux, L. Carro, and A. Bland, "Understanding gpu errors on large-scale hpc systems and the implications for system design and operation," in *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, 2015, pp. 331–342.

[49] M. B. Sullivan, N. Saxena, M. O'Connor, D. Lee, P. Racunas, S. Hukerikar, T. Tsai, S. K. S. Hari, and S. W. Keckler, *Characterizing And Mitigating Soft Errors in GPU DRAM*, 2021, p. 641–653. [Online]. Available: https://doi.org/10.1145/3466752.3480111

[50] F. F. dos Santos, P. F. Pimenta, C. Lunardi, L. Draghetti, L. Carro, D. Kaeli, and P. Rech, "Analyzing and increasing the reliability of convolutional neural networks on gpus," *IEEE Transactions on Reliability*, vol. 68, no. 2, pp. 663–677, 2018.

[51] F. Libano, B. Wilson, J. Anderson, M. J. Wirthlin, C. Cazzaniga, C. Frost, and P. Rech, "Selective hardening for neural networks in fpgas," *IEEE Transactions on Nuclear Science*, vol. 66, no. 1, pp. 216–222, 2018.

[52] P. Maillard, Y. P. Chen, J. Vidmar, N. Fraser, G. Gambardella, M. Sawant, and M. L. Voogel, "Radiation-Tolerant Deep Learning Processor Unit (DPU)-Based Platform Using Xilinx 20-nm Kintex UltraScale FPGA," *IEEE Transactions on Nuclear Science*, vol. 70, no. 4, pp. 714–721, October 2023.

[53] M. Traiola, F. F. Dos Santos, P. Rech, C. Cazzaniga, O. Sentieys, and A. Kritikakou, "Impact of high-level-synthesis on reliability of artificial neural network hardware accelerators," *IEEE Transactions on Nuclear Science*, pp. 1–1, 2024.

[54] C. Cazzaniga and C. D. Frost, "Progress of the scientific commissioning of a fast neutron beamline for chip irradiation," *Journal of Physics: Conference Series*, vol. 1021, p. 012037, may 2018. [Online]. Available: https://doi.org/10.1088/1742-6596/1021/1/012037

[55] JEDEC, "Measurement and Reporting of Alpha Particle and Terrestrial Cosmic Ray-Induced Soft Errors in Semiconductor Devices," JEDEC Standard, Tech. Rep. JESD89A, 2006.

[56] C. Nvidia, "Cublas library," *NVIDIA Corporation, Santa Clara, California*, vol. 15, no. 27, p. 31, 2008.

[57] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *CoRR*, vol. abs/1804.02767, 2018. [Online]. Available: http://arxiv.org/abs/1804.02767

[58] F. Fernandes dos Santos, C. Lunardi, D. Oliveira, F. Libano, and P. Rech, "Reliability evaluation of mixed-precision architectures," in *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2019, pp. 238–249.

[59] J. Duarte *et al.*, "Fast Inference of Deep Neural Networks for Real-Time Particle Physics Applications," *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 305–335, February 2019.

[60] L. Roquet, F. Fernandes dos Santos, P. Rech, M. Traiola, O. Sentieys, and A. Kritikakou, "Cross-Layer Reliability Evaluation and Efficient Hardening of Large Vision Transformers Models," in *Design Automation Conference (DAC)*, San Fracisco, United States, Jun. 2024. [Online]. Available: https://hal.science/hal-04456702

[61] G. Reis, J. Chang, N. Vachharajani, S. Mukherjee, R. Rangan, and D. August, "Design and evaluation of hybrid fault-detection systems," in *32nd International Symposium on Computer Architecture (ISCA'05)*, 2005, pp. 148–159.

Appendix 13

XIII

M. Jenihhin, M. Taheri, N. Cherezova, M. H. Ahmadilivani, H. Selg, A. Jutman, K. Shibin, A. Tsertov, S. Devadze, R. M. Kodamanchili, et al., "Keynote: Cost-Efficient Reliability for Edge-AI Chips," in 2024 IEEE 25th Latin American Test Symposium (LATS), pp. 1–2, 2024.

2024 IEEE 25th Latin American Test Symposium (LATS) | 979-8-3503-6555-9/24/$31.00 ©2024 IEEE | DOI: 10.1109/LATS62223.2024.10534610

Keynote: Cost-Efficient Reliability for Edge-AI Chips

Maksim Jenihhin, Mahdi Taheri, Natalia Cherezova, Mohammad Hasan Ahmadilivani, Hardi Selg, Artur Jutman, Konstantin Shibin, Anton Tsertov, Sergei Devadze, Rama Mounika Kodamanchili, Ahsan Rafiq, Jaan Raik, Masoud Daneshtalab

Department of Computer Systems, Tallinn University of Technology, Estonia
maksim.jenihhin@taltech.ee

***Abstract*—Very recently, Artificial Intelligence started undergoing a remarkable transformation by moving closer to the source of data, thus establishing the Edge AI concept. This trend sets new reliability requirements for the related hardware chips used for safety- and mission-critical applications. The key research and engineering challenges stem from the limited computing and energy resources of the edge devices. Furthermore, the compute-efficiency and the cost of the reliability of the Edge-AI chips are becoming enabling factors for their way to the market. The talk discusses techniques for soft-error and lifetime reliability assessment and enhancement for Deep Learning accelerators. It advocates the role of approximate computing and looks into specifics of the systolic-array-, data-flow-based and industry-grade accelerator architectures for ASICs and FPGAs.**

***Index Terms*—DNNs, HW accelerators, edge computing, edge AI, AxC, soft errors, reliability assessment and enhancement.**

I. Challenges and Trends

Edge AI is gaining momentum across various industries and services by public authorities. It is forecasted by Gartner and other roadmaps that *"by 2025, more than 55% of all data collected will be processed by Edge AI"*, while this figure was less than 10% just a couple of years ago. Yet, the advancements in applications face critical research and engineering challenges in the hardware domain. The real-time inference using massive computations, performed by the safety- and business-critical Edge AI applications on the strongly resource-constrained delicate nanoscale-technology-based hardware, still sets very high expectations for reliability against in-field faults (soft errors, aging, etc.). The boundaries should be pushed not only for compute-efficiency of Deep Neural Network HW Accelerators (DNN-HA) but also for their reliability-efficiency. As a response, promising research endeavours include: *analytical techniques*, *selective hardening*, *cross-layer solutions*, *in-field system health management*, *lightweight monitoring and mitigation*, and *approximation- or dynamic-inference-based trade-offs*.

II. Selected approaches

Tailored cost-efficient reliability enhancement for DNN HW accelerators starts with its assessment. A recent survey [1] categorizes the state-of-the-art DNN reliability assessment methods into *fault injection*, *analytical* and *hybrid* methods. Fault Injection (FI) approaches strongly dominate yet vary by different fault models, injection approaches and target platforms. The survey concludes that analytical and hybrid methods are lightweight, sufficiently accurate, yet their potential is under-explored by the community.

A semi-analytical technique and tool *DeepVigor* [2] for DNN fault resiliency assessment provides vulnerability value ranges for DNN neurons' outputs. The vulnerable and non-vulnerable ranges for each neuron are a base for novel techniques for DNN-HAs' reliability enhancement. E.g., it is a base for QNNs' model-level fault resiliency enhancement by selected critical neuron splitting [3]. Furthermore, it is applied for a hybrid analytical and hierarchical FI-based reliability assessment for systolic-array-based DNN-HAs [4]. The core idea is an analytical pruning of fault space that allows for even further acceleration of a hierarchical tool *SAFFIRA* [5], which employs a system of Uniform Recurrent Equations for systolic array modelling and fault simulation.

The reliability assessment can also be accelerated by an unconventional use of the errors produced by *Approximate Computing (AxC)*. The technique in [6], [7] uses approximate arithmetic circuits to agilely emulate errors in the target hardware without performing fault injection on the DNN. Naturally, AxC has a strong potential to facilitate cost-efficient reliability by releasing a share of DNN-HA resources for fault-tolerance needs or by enhancing reliability themselves [8].

Techniques for in-field immediate error correction in DNN-HAs imply signals monitoring, e.g. for MAC outputs' out-of-range values [9] or an instrumented multiplier outputs [10] with corresponding on-the-fly adjustment of the values. The latter technique uses adaptive fault-tolerant approximate multiplier architecture (*AdAM*) relying on an unconventional use of input Leading one Detector (LOD) and enabling "negative-overhead" reliability.

Industry-scale solutions for today's (FPGA-)SoCs demand a comprehensive cross-layer analysis in the systems fault-tolerance *vendor-* and *user-spaces* [11]. Here, the process for fault detection, localization [12] and recovery may be sophisticated and application-dependant, implying on-chip monitors' data analysis and SoC health management protocols [13].

979-8-3503-6555-9/24/$31.00 ©2024 IEEE

ACKNOWLEDGMENTS

The work was supported in part by the Estonian grant PUT PRG1467 “CRASHLESS” and by the Estonian-French PARROT project “EnTrustED”.

REFERENCES

[1] M. H. Ahmadilivani, M. Taheri, J. Raik, M. Daneshtalab, and M. Jenihhin, “A systematic literature review on hardware reliability assessment methods for deep neural networks,” *ACM Comput. Surv.*, 2024.

[2] M. H. Ahmadilivani *et al.*, “Deepvigor: Vulnerability value ranges and factors for dnns’ reliability assessment,” in *ETS’23*.

[3] M. H. Ahmadilivani, M. Taheri, J. Raik, M. Daneshtalab, and M. Jenihhin, “Enhancing fault resilience of qnns by selective neuron splitting,” in *AICAS’23*.

[4] M. H. Ahmadilivani, A. Bosio, B. Deveautour, F. F. dos Santos, J.-D. Guerrero-Balaguera, M. Jenihhin, A. Kritikakou, R. Limas Sierra, S. Pappalardo, J. Raik, J. E. Rodriguez Condia, M. Sonza Reorda, M. Taheri, and M. Traiola, “Special session: Reliability assessment recipes for dnn accelerators,” in *VTS’24*. (in press).

[5] M. Taheri, P. Salvatore, M. Jenihhin, A. Bosio, M. Daneshtalab, B. Deveautour, and J. Raik, “Saffira: a framework for assessing the reliability of systolic-array-based dnn accelerators,” in *DDECS’24*.

[6] M. Taheri, M. H. Ahmadilivani, M. Jenihhin, M. Daneshtalab, and J. Raik, “Appraiser: Dnn fault resilience analysis employing approximation errors,” in *DDECS’23*.

[7] M. H. Ahmadilivani, M. Barbareschi, S. Barone, A. Bosio, M. Daneshtalab, S. D. Torca, G. Gavarini, M. Jenihhin, J. Raik, A. Ruospo, E. Sanchez, and M. Taheri, “Special session: Approximation and fault resiliency of dnn accelerators,” in *VTS’23*.

[8] M. Taheri, M. Riazati, M. H. Ahmadilivani, M. Jenihhin, M. Daneshtalab, J. Raik, M. Sjõdin, and B. Lisper, “Deepaxe: A framework for exploration of approximation and reliability trade-offs in dnn accelerators,” in *ISQED’23*.

[9] M. Taheri, N. Cherezova, M. S. Ansari, M. Jenihhin, A. Mahani, M. Daneshtalab, and J. Raik, “Exploration of activation fault reliability in quantized systolic array-based dnn accelerators,” in *ISQED’24*.

[10] M. Taheri, N. Cherezova, S. Nazari, A. Rafiq, A. Azarpeyvand, T. Ghasempouri, M. Daneshtalab, J. Raik, and M. Jenihhin, “Adam: Adaptive fault-tolerant approximate multiplier for edge dnn accelerators,” in *ETS’24*. (in press).

[11] N. Cherezova, K. Shibin, M. Jenihhin, and A. Jutman, “Understanding fault-tolerance vulnerabilities in advanced soc fpgas for critical applications,” *Microelectronics Reliability*, vol. 146, 2023.

[12] H. Selg, M. Jenihhin, P. Ellervee, and J. Raik, “Ml-based online design error localization for risc-v implementations,” in *IOLTS’23*.

[13] K. Shibin, M. Jenihhin, A. Jutman, S. Devadze, and A. Tsertov, “On-chip sensors data collection and analysis for soc health management,” in *DFT’23*.

Appendix 14

XV

N. Cherezova, S. Pappalardo, M. Taheri, M. H. Ahmadilivani, B. Deveautour, A. Bosio, J. Raik, and M. Jenihhin, "Heterogeneous Approximation of DNN HW Accelerators based on Channels Vulnerability," in IEEE International Conference on Very Large Scale Integration (VLSI-SOC), 2024.

Heterogeneous Approximation of DNN HW Accelerators based on Channels Vulnerability

Natalia Cherezova[1], Salvatore Pappalardo[2], Mahdi Taheri[1], Mohammad Hasan Ahmadilivani[1], Bastien Deveautour[2], Alberto Bosio[2], Jaan Raik[1], and Maksim Jenihhin[1]

[1]Tallinn University of Technology, Tallinn, Estonia
[2]Univ Lyon, ECL, INSA Lyon, CNRS, UCBL, CPE Lyon, INL, Ecully, France

***Abstract*—Deep Neural Networks (DNNs) are widely used in our everyday life and there is a push towards embedded and edge devices with a context where the power and computational resources are heavily constrained. To that end, Approximate Computing (AxC) can be applied to reduce power consumption and execution time, since DNNs gracefully withstand approximation due to its inherent redundancy. In the literature, several works adopted the AxC paradigm to DNNs in the form of quantization, precision reduction, pruning and functional approximation. Despite the promising results demonstrated so far, most of the existing works have applied homogeneous AxC techniques, meaning that the same degree of approximation has been applied to the entire DNN. However, different DNN components (i.e., channels, filters, layers, neurons) have different resiliency levels. Therefore, instead of using a uniform degree of approximation throughout the DNN, it would be potentially beneficial to exploit the heterogeneous AxC technique by adapting the approximation degree according to the resiliency level of each DNN component. This paper presents a framework for applying heterogeneous AxC to DNN hardware accelerators. The framework is based on the identification of channel resilience and applying a tailored degree of approximation per channel. Preliminary results carried out on LeNet-5 model show that by using the proposed framework it is possible to decrease resource utilization by 65.2% and power consumption by 53.4% at the cost of a marginal drop of accuracy from 98.87% to 98.03%.**

***Index Terms*—deep neural networks, data-flow architecture, hardware accelerator, approximate computing, circuit design**

I. Introduction

Being able to make complex predictions through learning from a vast amount of data, DNNs are the new frontier of machine learning and artificial intelligence. They are capable of being more accurate than humans in certain applications [1]. Furthermore, DNNs are employed in many different fields, due to their outstanding computational capabilities. The current trend is pushing research and industry to deploy DNNs in embedded systems to be used at the edge in which power and computational resources are limited. This constitutes a challenge since DNNs are energy-hungry using large amounts of energy to compute even a single inference. For this reason, specialized hardware has been developed to achieve power efficiency and performance. For example, in [2] the authors designed an Application Specific Integrated Circuit (ASIC) to reduce power consumption and area for a real-time image classification application. In [3], the authors present a novel DNN accelerator design for both ASIC and FPGA. To address the power and latency issue, the Approximate Computing (AxC) paradigm can be used as an effective paradigm to reduce the power and the area required by the circuit since it proved to be an effective technique [4].

In the literature, several works adopted the AxC paradigm to DNNs in the form of quantization, precision reduction, pruning and functional approximation [4]. Despite the encouraging results, most of the existing works applied homogeneous AxC techniques, meaning that the same degree of approximation has been applied to the DNN. However, DNN components (i.e., channels, filters, layers, neurons) show different resiliency levels. Therefore, instead of using a uniform degree of approximation to all components, it would be beneficial to exploit the heterogeneous AxC technique by adapting the approximation degree according to the resiliency level of each DNN component.

This paper presents a framework for applying heterogeneous AxC at the channel level to DNN hardware accelerators. The framework is based on identifying channels' resilience and applies a specific degree of approximation per channel. In particular, we leverage precision reduction, which corresponds to neglecting the Least Significant Bits (LSBs) of the multiplier's input and effectively using a smaller bit-width.

The rest of this paper is structured as follows. In Section II, a brief overview of existing works is given. Section III describes the proposed methodology. Section IV presents the experimental setup and discusses the obtained results. Finally, Section V concludes the paper and proposes future directions.

II. Related Works

The advantages of implementing and deploying DNNs on FPGAs are advocated in several recent works [5]–[7]. The existing FPGA-based toolchains to map Convolutional Neural Networks (CNNs) are presented in the surveys [8]–[11]. Data-flow is an important computation architecture for customized hardware accelerators, which enables the parallel temporal execution of multiple coarse-grained tasks [12]. The FINN framework [13] is released by Xilinx to explore quantized CNNs' inference on FPGAs that also provide customized data-flow architectures for each network.

Research work [14] provides Register-Transfer Level (RTL) models using conventional synthesis tools, e.g., Vivado HLS, where the outputs can be directly synthesized on an FPGA.

Heterogeneous systems are another design strategy in the automated toolchains that propose hardware-software co-design [14]–[16]. In these designs, computational units, e.g., addition or multiplication, are mainly implemented on Processing Logic (PL) controlled by a CPU control unit using a dedicated framework, e.g., OpenCL [17].

To further enhance the performance of DNN accelerators, several techniques are introduced in the literature (e.g., quantization and approximate computing). Using Fixed-point (FxP) data type instead of Floating Point (FP) is becoming popular due to the more optimized resource utilization while keeping the output accuracy degradation at an acceptable level [14], [18], [19]. In [20] a fully automated framework capable of applying various quantization-aware techniques, and hardware implementation is introduced to measure hardware parameters.

On the other hand, multiplication is one of the primary arithmetic operations widely used in DNNs. Approximate computing is a promising technique for designing digital circuits with lower area and power consumption while achieving a higher working frequency, particularly when the target application has some error resiliency [21]. Various approximate multipliers are proposed in the literature. DeepAxe is a framework that enables the selective approximation for data-flow DNN accelerators to provide a design space exploration for the efficiency and accuracy of DNNs [22].

Vulnerability analysis is mainly performed by analytical approaches [6], [23] due to their short execution time. Several methods are proposed in the literature [24]–[26]. DeepVigor is an accurate analytical method providing detailed and fine-grain vulnerability value ranges for each neuron in a DNN which is extended for QNNs in [27]. Vulnerability value ranges indicate minimum error values induced at the neurons' output that can misclassify the DNN's golden result. These results can be exploited in the approximation computing scheme to obtain an optimal approximation for computing units in DNNs' channels.

To the best of our knowledge, there is no work presenting customized heterogeneous channel-wise approximations for hardware accelerators leading to high efficiency in terms of resource utilization and power consumption. The approach proposed in this paper goes beyond the state-of-the-art by establishing a comprehensive methodology for enabling heterogeneous approximation of DNN HW accelerators based on channels' vulnerability analysis.

III. Methodology

In this section, the methodology to achieve heterogeneous approximation for DNN hardware accelerators is presented. Fig. 1 depicts the proposed method in this work which gets a CNN model, test data and hardware architecture and outputs an approximated version of the accelerator executing the CNN with a high accuracy with remarkably higher efficiency in terms of power and area.

The purpose of the method is to systematically provide a hardware accelerator for CNNs in a way that its multipliers are heterogeneously approximated through each layer, at the channel level, to achieve an efficient accelerator with highly accurate CNNs. In this regard, a vulnerability analysis for CNNs' channels is performed to enable error-aware approximation. Afterwards, an optimum point for the heterogeneous approximation throughout the channels and layers is obtained. Finally, the generated approximated multipliers are synthesized on an FPGA and hardware results are reported.

A. Initialization

The method receives a trained CNN model, test data, and the data-flow hardware accelerator as inputs. The method is presented for quantized CNNs, where the data-flow accelerator consists of a series of components connected according to the architecture of the selected neural network. Each component implements one layer of the network. Therefore, it can be optimized based on the layer parameters, e.g., the number of input and output channels, kernel size, etc. Outputs of the layers are stored in the intermediate memory buffers as the data is streaming through those components.

B. Channel Vulnerability Analysis

To obtain the maximum errors at the output of each channel/neuron leading to the minimum possible accuracy drop due to approximation, we adopt and extend the DeepVigor methodology presented in [26] and [27] that provides vulnerability analysis for DNNs and QNNs, respectively.

Let δ_k^l be a deviation that is an added positive or negative error value to an output Feature Map (FMap) by approximation for a k-th neuron at layer l with input data x. Let Δ_k^l be the minimum absolute deviation added to the corresponding neuron by approximation that misclassifies the input image x from its golden classification in the exact DNN. This deviation is defined as follows:

$$\Delta_k^l = min(|\delta_k^l|), \mathcal{E}_t < \mathcal{E}_i, i \neq t; \quad i,t \in C \tag{1}$$

where C is the set of all the classes of the classifier, i and t are elements belonging to C (more specifically t is the top class of the exact DNN) and $\mathcal{E}_t$ and $\mathcal{E}_i$ are the deviated output logits corresponding to the respective output classes. Thus, Δ_k^l represents the maximum absolute deviation of a neuron's output from its error-free value due to the approximation that would not misclassify the DNN.

Approximation design exploits Δ_k^l to identify the bits that can be approximated in the calculations. Δ_k^l can be interpreted as the set of bits for each neuron to be approximated without misclassification. In the integer data type, approximating the LSB corresponds to $\Delta_k^l = 1$. As an example, if $\Delta_k^l = 16$, all the 4 least significant bits can be approximated, since any deviation less than this value would not affect the classification output. Therefore, the set of bits to be approximated is obtained by:

$$Bits\ to\ be\ approx. = [int(log_2(\Delta_k^l)) - 1 : 0] \tag{2}$$

Since approximation design is performed at the channel level of the DNN, the obtained Δ_k^l for each FMap in convolutional layers are aggregated channel-wise. To provide a preliminary

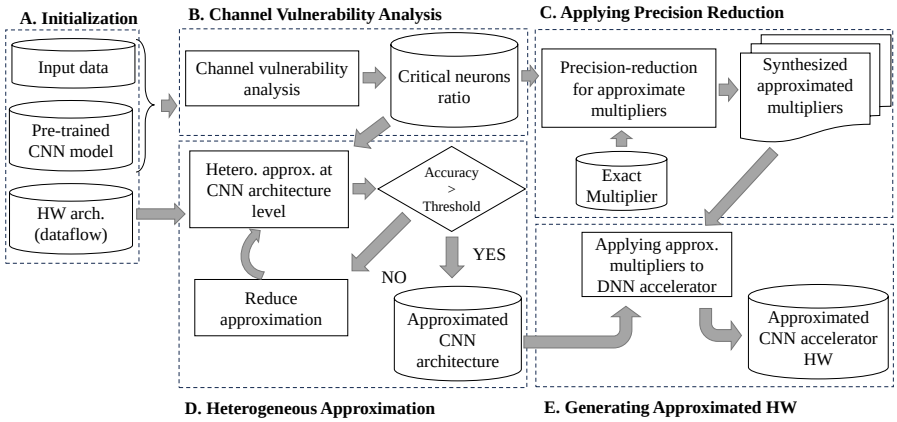


Fig. 1: Proposed flow to achieve heterogeneous approximation for data-flow CNN hardware accelerators

analysis of the effect of approximating channels on the accuracy, we explore the level of approximation regarding each bit corresponding to the obtained Δ_k^l for the neurons inside that channel. This analysis expresses what portion of neurons in a channel has a higher Δ_k^l than the approximation error, which is called *critical neurons ratio*.

In this paper, we consider the *critical neurons ratio* to be lower than 5% as a safe approximation level. In this regard, the approximation in the multipliers of a channel is applied to the least significant bits up to the bit corresponding to the safe approximation level. It is noteworthy that each channel in a layer has its own safe approximation level, leading to a heterogeneous approximation within layers throughout the CNNs.

C. Applying Precision Reduction to Multipliers

As mentioned earlier, the multipliers approximation is based on precision scaling [28]. Specifically, the lower-order bits of the input operands of the multiplier are neglected (Fig. 2. The number of neglected bits is denoted with n. As an example let us consider an exact 16-bit multiplier. It has two 16-bit inputs and a 32-bit output. In the n-approximated 16-bit multiplier, the resulting multiplier will only consider the $16-n$ higher-order bits of both operands and will produce a $32-2n$ output, while the remaining $2n$ lower-order bits will be set to 0.

This technique has three major advantages:

- an entire library of approximate multipliers can be generated at practically zero cost;
- the level of approximation has a clear meaning, depending on the parameter n, and consequently a different approximation level can easily be associated with a channel.

As mentioned, the generation of a multiplier with a given approximation level is straightforward since there is a direct correspondence between the approximation level and the introduced error. However, any other customized library of approximate multipliers can be used. The approximation level is defined as the number n of lower-order bits that are disregarded (as in Fig. 2) and, consequently, the introduced Worst-Case Error (WCE) corresponds to 2^n-1 for the inputs and $2^{2n}-1$ for the outputs.

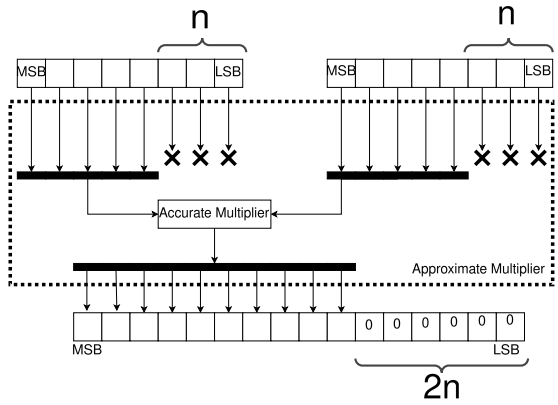


Fig. 2: Precision scaled approximate multiplier

D. Heterogeneous Approximation

This step attempts to identify an optimal configuration for a channel-wise heterogeneous approximated hardware accelerator. Algorithm 1 presents how we obtain the corresponding configuration. First, a separate approximation to each channel in a layer is applied to find a configuration in which the accuracy of the CNN does not drop more than 1% compared to the baseline exact CNN accuracy (lines 1-13). To this end, the aggressive channel-wise approximation configuration is applied initially

to a layer. Then, the approximation continues to increase by 1 bit for all channels until the accuracy drop is less than 1%.

After finding the best channel-wise approximation for each layer separately, all channels are approximated throughout the CNN based on the obtained configurations (line 14). Then, we reduce the level of channel-wise approximation from the first to the last layer, one by one, until the accuracy of the approximated CNN is not 1% less than the accuracy of the exact CNN (lines 15-24). The output of this algorithm is an optimal point for approximation where the accuracy of the approximated CNN is negligible.

Algorithm 1 Identifying an optimal heterogeneous approximation for HW CNN accelerator

Input: Exact CNN model, test dataset, configuration for bit approximation in each channel for all layers.

Output: Optimal configuration for heterogeneous configuration.

Assume: $l \in 1, 2, \cdots, L$ is the set of layers in the CNN, $Accuracy_{ex}$ is the accuracy of the exact CNN before approximation, $model.forward(test\ data)$ performs a forward pass of the CNN to get its accuracy.

1: **for** $l \in 1, 2, \cdots, L$ **do**:
2: Apply safe approximation to channels in this layer;
3: **while** True **do**:
4: $Accuracy^l_{ap} = model.forward(test\ data)$;
5: **if** $Accuracy_{ex} - Accuracy^l_{ap} < 1$ **then**:
6: Store current approximation in $Approx^l$;
7: Increase approximation for 1 bit;
8: **else**:
9: Save $Approx^l$ in $Approx_{layers}$;
10: Break;
11: **end if**
12: **end while**;
13: **end for**
14: Apply approximation $Approx_{layers}$ to all layers;
15: variable $i = 1$;
16: **while** True **do**:
17: $Accuracy_{ax} = model.forward(test\ data)$;
18: **if** $Accuracy_{ex} - Accuracy_{ax} > 1$ **then**
19: Reduce approximation for 1 bit in layer i;
20: $i = i + 1$;
21: **else**:
22: Return the approximated CNN;
23: Break;
24: **end if**
25: **end while**

E. Generating Approximated Hardware

As mentioned earlier, each component in the data-flow DNN accelerator is dedicated to a specific layer of the network and, therefore, can be optimized according to the layer parameters. The configuration obtained using Algorithm 1 defines what approximate multipliers should be used for each channel. The obtained channel-wise heterogeneous approximated DNN accelerator is depicted in Fig. 3. Different colors denote different approximate multipliers in the Processing Elements (PE). The generated DNN accelerator is compared with the baseline accelerator, which utilizes exact multipliers, in terms of resource utilization, number of Look-Up Tables (LUT) and flip-flops (FF), and power consumption.

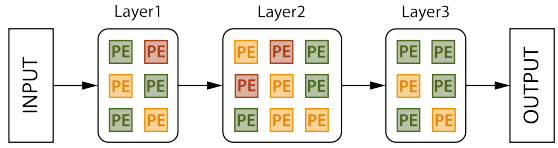


Fig. 3: Heterogeneous data-flow DNN accelerator: different colors denote different approximate multipliers in the Processing Elements (PE)

IV. Experimental Results

A. Experimental Setup

A common CNN benchmark LeNet-5 is used as a case study. The structure of the network is presented in Table I and consists of two convolutional layers, followed by max-pooling layers, and three fully-connected (dense) layers. ReLU (Rectified Linear Unit) is used as an activation function. The network was trained on the MNIST dataset of handwritten digits [29] with PyTorch using full precision, i.e., 32-bit floating point. After the training, the network was quantized to a 16-bit integer data type. The accuracy of the quantized network was 98.87%.

TABLE I: Case-study DNN architecture

Layer	Input size	Kernel size	Output size
Conv1	$32 \times 32 \times 1$	$5 \times 5 \times 6$	$28 \times 28 \times 6$
Max pool	$28 \times 28 \times 6$	2×2	$14 \times 14 \times 6$
Conv2	$14 \times 14 \times 6$	$5 \times 5 \times 16$	$10 \times 10 \times 16$
Max pool	$10 \times 10 \times 16$	2×2	$5 \times 5 \times 16$
FC3	$5 \times 5 \times 16$	—	120
FC4	120	—	84
FC5	84	—	10

A data-flow accelerator for the LeNet-5 is developed considering the AMD-Xilinx FPGA platform as a target. As mentioned, in the data-flow architecture, each layer is implemented as a separate component. This way, each layer is mapped to the component optimized for that layer's parameters. Each convolutional layer consists of computational modules for each output channel. Thus, all output channels are processed in parallel. Each output channel, in turn, consists of computational modules for each input channel.

The input channel computational module includes a multiplier and an adder. Each fully-connected layer consists of computational modules for each output neuron. The output neuron computational module includes a multiplier and an adder as well. This way, Conv1 layer utilizes $6 * 1 = 6$ multipliers, Conv2 layer $16 * 6 * 1 = 96$, FC3 layer 120, FC4 layer 84, and FC5 layer 10 multipliers. Zynq UltraScale+

MPSoC ZCU104 Evaluation Board and Vivado 2021.2 are used for the experiments.

B. Results of Channel Vulnerability Analysis

As mentioned in subsection III-B, an exploration is performed to obtain the portion of neurons in each channel of a layer such that their Δ_k^l is smaller than the error induced by approximation. The vulnerability of convolutional and fully-connected layers is analyzed since only those layers involve multiplication. The last fully-connected layer is excluded from the analysis because it is the output layer and any changes there will directly affect the output of the CNN.

Table II presents the results of channel vulnerability analysis for the first convolutional layer of LeNet-5. The table reports the percentage of approximated neurons in each channel that result in misclassification. The data is organized such that rows show analyzed channels and columns show the number of bits to approximate. The table is color-coded based on the set thresholds: green shows all the channel-approximation pairs that have less than 5% critical neurons, orange those that have less than 10%, and red all the rest.

TABLE II: Percentage of neurons in each channel of the first layer of the case-study DNN with respect to error induced by approximation

Error induced by approximation	1	2	4	8	16	32	64	128
Channel 1	1.14	1.21	1.46	1.59	2.80	8.10	15.75	32.14
Channel 2	1.21	1.46	1.53	1.59	2.42	4.72	12.05	22.38
Channel 3	1.02	1.02	1.02	1.59	2.29	5.10	11.73	23.72
Channel 4	0.95	1.08	1.14	1.40	2.42	4.71	13.39	28.31
Channel 5	0.89	0.95	1.40	3.12	6.05	10.39	19.77	31.05
Channel 6	0.12	0.12	0.12	0.51	1.21	5.42	14.03	29.27

C. Approximate multipliers

A library of approximate multipliers is generated by applying the precision reduction technique to the exact 16-bit multiplier. The comparison of the resource utilization and power consumption is given in Table III.

TABLE III: Resource utilization of generated multipliers

Multiplier bit-width	**LUT**	**Power (mW)**
16	416	67.803
15	377	62.500
14	301	58.108
13	264	53.649
12	223	49.290
11	171	43.513
10	146	40.567
9	117	36.551
8	81	32.457
7	64	28.361

D. Heterogeneous Approximation at the Architecture Level

Using the data obtained by channel vulnerability analysis and the critical neurons ratio set to be lower than 5%, a *safe approximation level* is derived for each channel in each layer. Safe approximation level defines the number of lower-order bits that can be approximated, in our case negated. First, the effect of individual layer approximation on the accuracy of LeNet-5 is studied using the network implementation in C language. Each layer is successively approximated while the rest of the network is calculated using precise multiplication.

For each layer, several tests are performed by adding an offset to the safe approximation level, which increases the number of negated bits. This way it is possible to observe how the degree of approximation affects the performance of the layer. The results of this study are given in Table IV. As observed, a safe approximation level results in a high accuracy for each single layer. Nonetheless, increasing the approximation level incurs a high accuracy drop. The selected configurations for each layer are **bold** in the table, as indicated in Algorithm 1.

TABLE IV: The effect of individual layer approximation on the accuracy of the case-study DNN

Layer	**Offset**	**CNN accuracy (%)**
Conv1	No offset	98.91
	Offset by 1	**98.23**
	Offset by 2	54.97
	Offset by 3	17.25
	Offset by 4	8.92
	Offset by 5	8.92
	Offset by 6	8.92
Conv2	No offset	98.88
	Offset by 1	98.88
	Offset by 2	98.80
	Offset by 3	98.79
	Offset by 4	**98.50**
	Offset by 5	95.26
	Offset by 6	84.57
FC3	No offset	98.88
	Offset by 1	98.83
	Offset by 2	98.75
	Offset by 3	**98.41**
	Offset by 4	97.27
	Offset by 5	92.52
	Offset by 6	69.18
FC4	No offset	98.85
	Offset by 1	98.88
	Offset by 2	98.89
	Offset by 3	98.82
	Offset by 4	98.79
	Offset by 5	**98.28**
	Offset by 6	96.82

E. Generated Approximate DNN HW Accelerator

An optimal configuration is generated based on Algorithm 1. After applying the selected approximation in Table IV, the accuracy of the approximated CNN is 81.09%. By applying the algorithm, the final approximation for each layer is {Conv1: no offset, Conv2: offset by 3, FC3: offset by 2, FC4: offset by 4} resulting in 98.03% accuracy.

A comparison of the resource utilization and power consumption between the baseline configuration using exact multipliers and the selected configuration using heterogeneous approximate multipliers is given in Table V. It can be seen that by using the proposed methodology it was possible to decrease resource

utilization by 65.2% (i.e. by the factor of 3x) and power consumption by 53.4% with the marginal drop of accuracy from 98.87% to 98.03%.

TABLE V: Resource utilization of generated accelerator

Parameters	Baseline	Selected config
Network accuracy, %	98.87	98.03
LUT	157,155	54,640
LUTRAM	148	148
FF	29,864	19,667
BRAM	77.5	77.5
Power, W	2.176	1.012
Resource savings, %	—	65.2
Power savings, %	—	53.4

V. Conclusions

This paper presents a framework for applying heterogeneous approximation to hardware DNN accelerators. The framework identifies the resilience of channels in convolutional and fully-connected layers and applies a specific degree of approximation per channel. A semi-analytical approach DeepVigor is adopted for vulnerability analysis of individual channels to identify a safe approximation level for each channel. As the approach for approximation, we leverage precision reduction that corresponds to neglecting the lower-order bits of the multiplier's inputs and effectively using a smaller bit-width multiplier. Preliminary results demonstrate possibility to decrease resource utilization by 65.2% and power consumption by 53.4% at the cost of marginal accuracy drop from 98.87% to 98.03%.

VI. Acknowledgement

This work was supported in part by the Estonian Research Council grant PUT PRG1467 "CRASHLESS" and by the Estonian-French PARROT project "EnTrustED".

References

[1] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "Imagenet large scale visual recognition challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[2] P.-Y. Chuang, P.-Y. Tan, C.-W. Wu, and J.-M. Lu, "A 90nm 103.14 tops/w binary-weight spiking neural network cmos asic for real-time object classification," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, 2020, pp. 1–6.

[3] S. K. Venkataramanaiah, S. Yin, Y. Cao, and J.-S. Seo, "Deep neural network training accelerator designs in asic and fpga," in *2020 International SoC Design Conference (ISOCC)*, 2020, pp. 21–22.

[4] A. Bosio, D. Ménard, and O. Sentieys, Eds., *Approximate Computing Techniques*. Springer International Publishing, 2022.

[5] M. A. Talib, S. Majzoub, Q. Nasir, and D. Jamal, "A systematic literature review on hardware implementation of artificial intelligence algorithms," *The Journal of Supercomputing*, vol. 77, pp. 1897–1938, 2021.

[6] M. H. Ahmadilivani, M. Taheri, J. Raik, M. Daneshtalab, and M. Jenihhin, "A systematic literature review on hardware reliability assessment methods for deep neural networks," *ACM Computing Surveys*, vol. 56, no. 6, pp. 1–39, 2024.

[7] M. H. Ahmadilivani, M. Barbareschi, S. Barone, A. Bosio, M. Daneshtalab, S. Della Torca, G. Gavarini, M. Jenihhin, J. Raik, A. Ruospo *et al.*, "Special session: Approximation and fault resiliency of dnn accelerators," in *2023 IEEE 41st VLSI Test Symposium (VTS)*. IEEE, 2023, pp. 1–10.

[8] S. I. Venieris, A. Kouris, and C.-S. Bouganis, "Toolflows for mapping convolutional neural networks on fpgas: A survey and future directions," *ACM Computing Surveys*, vol. 51, no. 3, jun 2018.

[9] K. Guo, S. Zeng, J. Yu, Y. Wang, and H. Yang, "A survey of FPGA-based neural network inference accelerators," *ACM Trans. on Reconfigurable Technology and Systems (TRETS)*, vol. 12, no. 1, pp. 1–26, 2019.

[10] K. Abdelouahab, M. Pelcat, J. Serot, and F. Berry, "Accelerating CNN inference on FPGAs: a survey," *arXiv preprint arXiv:1806.01683*, 2018.

[11] R. S. Molina, V. Gil-Costa, M. L. Crespo, and G. Ramponi, "High-level synthesis hardware design for FPGA-based accelerators: Models, methodologies, and frameworks," *IEEE Access*, vol. 10, pp. 90 429–90 455, 2022.

[12] H. Ye, H. Jun, and D. Chen, "HIDA: a hierarchical dataflow compiler for high-level synthesis," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*, 2024, pp. 215–230.

[13] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre, and K. Vissers, "Finn: A framework for fast, scalable binarized neural network inference," in *ACM/SIGDA international symposium on field-programmable gate arrays*, 2017, pp. 65–74.

[14] S. I. Venieris and C.-S. Bouganis, "fpgaConvNet: mapping regular and irregular convolutional neural networks on FPGAs," *IEEE Trans. on neural networks and learning systems*, vol. 30, no. 2, pp. 326–342, 2018.

[15] A. Ghaffari and Y. Savaria, "CNN2Gate: an implementation of convolutional neural networks inference on FPGAs with automated design space exploration," *Electronics*, vol. 9, no. 12, 2020.

[16] P. G. Mousouliotis and L. P. Petrou, "Cnn-grinder: from algorithmic to high-level synthesis descriptions of CNNs for low-end-low-cost FPGA SoCs," *Microprocessors and Microsystems*, vol. 73, p. 102990, 2020.

[17] J. E. Stone, D. Gohara, and G. Shi, "OpenCL: a parallel programming standard for heterogeneous computing systems," *Computing in science & engineering*, vol. 12, no. 3, p. 66, 2010.

[18] K. Guo, L. Sui, J. Qiu, S. Yao, S. Han, Y. Wang, and H. Yang, "Angel-eye: A complete design flow for mapping cnn onto customized hardware," in *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 2016, pp. 24–29.

[19] H. Sharma, J. Park, D. Mahajan, E. Amaro, J. K. Kim, C. Shao, A. Mishra, and H. Esmaeilzadeh, "From high-level deep neural models to FPGAs," in *49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2016, pp. 1–12.

[20] M. Taheri, N. Cherezova, M. S. Ansari, M. Jenihhin, A. Mahani, M. Daneshtalab, and J. Raik, "Exploration of activation fault reliability in quantized systolic array-based DNN accelerators," in *2024 25th International Symposium on Quality Electronic Design (ISQED)*, 2024.

[21] M. Taheri, N. Cherezova, S. Nazari, A. Rafiq, A. Azarpeyvand, T. Ghasempouri, M. Daneshtalab, J. Raik, and M. Jenihhin, "AdAM: adaptive fault-tolerant approximate multiplier for edge DNN accelerators," in *2024 IEEE European Test Symposium (ETS)*, 2024.

[22] M. Taheri, M. Riazati, M. H. Ahmadilivani, M. Jenihhin, M. Daneshtalab, J. Raik, M. Sjodin, and B. Lisper, "DeepAxe: a framework for exploration of approximation and reliability trade-offs in DNN accelerators," in *2023 24th International Symposium on Quality Electronic Design (ISQED)*, 2023, pp. 1–8.

[23] A. Bosio, M. H. Ahmadilivani, B. Deveautour, F. F. dos Santos, J. D. G. Balaguera, M. Jenihhin, A. Kritikakou, R. L. Sierra, S. Pappalardo, J. Raik *et al.*, "Special session: Reliability assessment recipes for DNN accelerators," *IEEE VLSI Test Symposium (VTS)*, 2024.

[24] A. Mahmoud, S. K. S. Hari, C. W. Fletcher, S. V. Adve, C. Sakr, N. Shanbhag, P. Molchanov, M. B. Sullivan, T. Tsai, and S. W. Keckler, "HarDNN: feature map vulnerability evaluation in CNNs," *arXiv preprint arXiv:2002.09786*, 2020.

[25] C. Schorn, A. Guntoro, and G. Ascheid, "Accurate neuron resilience prediction for a flexible reliability management in neural network accelerators," in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2018, pp. 979–984.

[26] M. H. Ahmadilivani, M. Taheri, J. Raik, M. Daneshtalab, and M. Jenihhin, "DeepVigor: Vulnerability value ranges and factors for DNNs' reliability assessment," in *2023 IEEE European Test Symposium (ETS)*. IEEE, 2023, pp. 1–6.

[27] ——, "Enhancing fault resilience of QNNs by selective neuron splitting," in *2023 IEEE 5th International Conference on Artificial Intelligence Circuits and Systems (AICAS)*. IEEE, 2023, pp. 1–5.

[28] S. Mittal, "A survey of techniques for approximate computing," *ACM Computing Surveys (CSUR)*, vol. 48, no. 4, pp. 1–33, 2016.

[29] C. J. B. Yann, Y. LeCun, and C. Cortes, "The MNIST database of handwritten digits," http://yann.lecun.com/exdb/mnist/, [Online].

Curriculum Vitae

1. Personal data

Name	Mahdi Taheri
Date and place of birth	12 January 1995 Kerman, Iran
Nationality	Iranian

2. Contact information

Address	Tallinn University of Technology (TalTech), School of Information Technologies, Department of Computer Systems, Ehitajate tee 5, 19086 Tallinn, Estonia
E-mail	mahdi.taheri@taltech.ee

3. Education

2021–2024	Tallinn University of Technology, School of Information Technologies, Information and Communication Technology, Ph.D. studies
2018–2021	Shahid Bahonar University of Kerman, Faculty of Electrical Engineering, Electrical Engineering, MSc
2014–2018	K. N. Toosi University of Technology, Faculty of Electrical Engineering, Electrical Engineering, BSc

4. Language competence

Persian	native
English	fluent

5. Defended theses

- 2018, Hardware Acceleration and Implementation of DNA sequencing Alignment Algorithms, MSc, supervisor Prof. Dr. Ali Mahani, Shahid Bahonar University of Kerman, Institute of Electrical Engineering

6. Field of research

- Deep Neural Networks
- Reliability
- Approximate Computing
- Hardware Designs and Implementations
- FPGA and ASIC synthesis

Elulookirjeldus

1. Isikuandmed

Nimi	Mahdi Taheri
Sünnikuupäev ja -koht	12. jaanuar 1995, Kerman, Iraan
Rahvus	Iraani

2. Kontaktandmed

Aadress	Tallinna Tehnikaülikool (TalTech), Infotehnoloogia teaduskond, Arvutisüsteemide instituut, Ehitajate tee 5, 19086 Tallinn, Eesti
E-post	mahdi.taheri@taltech.ee

3. Haridus

2021–2024	Tallinna Tehnikaülikool, Infotehnoloogia teaduskond, Infotehnoloogia ja kommunikatsioonitehnoloogia, doktorikraad
2018–2021	Shahid Bahonari Ülikool, Kermani Elektri- ja Tehnoloogia Teaduskond, Elektritehnika, magistrikraad
2014–2018	K. N. Toosi Tehnikaülikool, Elektri- ja Tehnoloogia Teaduskond, Elektritehnika, bakalaureusekraad

4. Keelteoskus

Pärsia keel	emakeel
Inglise keel	sorav

5. Kaitstud väitekirjad

- 2018, Riistvarakiirus ja DNA järjestamise joondamise algoritmide rakendamine, magistrikraad, juhendaja Prof. Dr. Ali Mahani, Shahid Bahonari Ülikool, Kermani Elektriinseneri Instituut

6. Uuringute valdkond

- Sügavad närvivõrgud
- Usaldusväärsus
- Ligikaudne arvutamine
- Riistvaradisain ja rakendused
- FPGA ja ASIC süntees

ISSN 2585-6901 (PDF)
ISBN 978-9916-80-251-9 (PDF)