TAL TECH

DOCTORAL THESIS

Assessment and Enhancement of Hardware Reliability for Deep Neural Networks

Mohammad Hasan Ahmadilivani

TALLINNA TEHNIKAÜLIKOOL
TALLINN UNIVERSITY OF TECHNOLOGY
TALLINN 2025

TALLINN UNIVERSITY OF TECHNOLOGY
DOCTORAL THESIS
19/2025

Assessment and Enhancement of Hardware Reliability for Deep Neural Networks

MOHAMMAD HASAN AHMADILIVANI

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 January 2, 2025.

Supervisor: Professor Jaan Raik,
Department of Computer Systems, School of Information Technologies,
Tallinn University of Technology,
Tallinn, Estonia

Co-supervisor: Professor Masoud Daneshtalab,
Department of Computer Systems, School of Information Technologies,
Tallinn University of Technology,
Tallinn, Estonia

Opponents: Professor Yanjing Li,
University of Chicago,
Chicago, USA

Professor Luciano Ost,
Loughborough University,
Loughborough, UK

Defense of the thesis: April 4, 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.

Mohammad Hasan Ahmadilivani _____
signature

ISSN 2585-6898 (publication)
ISBN 978-9916-80-275-5 (publication)
ISSN 2585-6901 (PDF)
ISBN 978-9916-80-276-2 (PDF)
DOI https://doi.org/10.23658/taltech.19/2025
Printed by Koopia Niini & Rauam

Ahmadilivani, M. H. (2025). *Assessment and Enhancement of Hardware Reliability for Deep Neural Networks* [TalTech Press]. https://doi.org/10.23658/taltech.19/2025

TALLINNA TEHNIKAÜLIKOOL
DOKTORITÖÖ
19/2025

Riistvara töökindluse hindamine ja täiustamine süvanärvivõrkude jaoks

MOHAMMAD HASAN AHMADILIVANI

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. 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*, 56(6):1–36, 2024.
DOI: https://doi.org/10.1145/3638242

II M. H. Ahmadilivani, M. Taheri, J. Raik, M. Daneshtalab, and M. Jenihhin. DeepVigor: VulnerabIlity Value RanGes and FactORs for DNNs' Reliability Assessment. In *IEEE European Test Symposium (ETS)*, pages 1–6. Venice, Italy, 2023.
DOI: https://doi.org/10.1109/ETS56758.2023.10174133

III M. H. Ahmadilivani, J. Raik, M. Daneshtalab, and M. Jenihhin. Deepvigor+: A Scalable, Accurate and Automated Framework for Resilience Analysis of Deep Neural Networks. *Under review*, pages 1–14, 2024.
DOI: https://doi.org/10.48550/arXiv.2410.15742

IV M. H. Ahmadilivani, M. Taheri, J. Raik, M. Daneshtalab, and M. Jenihhin. Enhancing Fault Resilience of QNNs by Selective Neuron Splitting. In *IEEE 5th International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, pages 1–5. Hangzhou, China, 2023.
DOI: https://doi.org/10.1109/AICAS57966.2023.10168633

V S. Mousavi, M. H. Ahmadilivani, J. Raik, M. Jenihhin, and M. Daneshtalab. ProAct: Progressive Training for Hybrid Clipped Activation Function to Enhance Resilience of DNNs. *Under review*, pages 1–12, 2024.
DOI: https://doi.org/10.48550/arXiv.2406.06313

VI 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. In *The 30th IEEE International Symposium on On-Line Testing and Robust System Design (IOLTS)*, pages 1–6. Rennes, France, 2023.
DOI: https://doi.org/10.1109/IOLTS60994.2024.10616072

VII M. H. Ahmadilivani, J. Raik, M. Daneshtalab, and A. Kuusik. Analysis and Improvement of Resilience for Long Short-Term Memory Neural Networks. In *IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, pages 1–4. Juan-Les-Pennes, France, 2023.
DOI: https://doi.org/10.1109/DFT59622.2023.10313559

VIII B. Parchekani, S. Nazari, M. H. Ahmadilivani, A. Azarpeyvand, J. Raik, T. Ghasempouri, and M. Daneshtalab. Zero-Memory-Overhead Clipping-Based Fault Tolerance for LSTM Deep Neural Networks. In *IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, pages 1–4. Oxforshire, United Kingdom, 2024.
DOI: https://doi.org/10.1109/DFT63277.2024.10753533

Other Publications

IX 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, J. E. Rodriguez Condia, M. Sonza Reorda, M. Taheri, and M. Traiola. Special Session: Reliability Assessment Recipes for DNN Accelerators. In *IEEE 42nd VLSI Test Symposium (VTS)*, pages 1–11. Tempe, United States of America, 2024.
DOI: https://doi.org/10.1109/VTS60656.2024.10538707

X 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 *IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*. Tanger, Morocco, 2024.
DOI: https://doi.org/10.1109/VLSI-SoC62099.2024.10767798

XI 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 *IEEE 24th International Symposium on Quality Electronic Design (ISQED)*, pages 1–8. San Francisco, United States of America, 2023.
DOI: https://doi.org/10.1109/ISQED57927.2023.10129353

XII M. Taheri, M. H. Ahmadilivani, M. Jenihhin, M. Daneshtalab, and J. Raik. Appraiser: DNN Fault Resilience Analysis Employing Approximation Errors. In *IEEE 26th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS)*, pages 124–127. Tallinn, Estonia, 2023.
DOI: https://doi.org/10.1109/DDECS57882.2023.10139468

XIII M. H. Ahmadilivani, M. Barbareschi, S. Barone, A. Bosio, M. Daneshtalab, S. Della Torca, G. Gavarini, M. Jenihhin, J. Raik, A. Ruospo, E. Sanchez, and M. Taheri. Special Session: Approximation and Fault Resiliency of DNN Accelerators. In *IEEE 41st VLSI Test Symposium (VTS)*, pages 1–10. San Diego, United States of America, 2023.
DOI: https://doi.org/10.1109/VTS56346.2023.10140043

XIV I. Dadras, M. H. Ahmadilivani, S. Banerji, J. Raik, and A. Abloo. An Efficient Analog Convolutional Neural Network Hardware Accelerator Enabled by a Novel Memoryless Architecture for Insect-Sized Robots. In *11th International Conference on Modern Circuits and Systems Technologies (MOCAST)*, pages 1–6. Bremen, Germany, 2022.
DOI: https://doi.org/10.1109/MOCAST54814.2022.9837551

XV I. Dadras, S. Seydi, M. H. Ahmadilivani, J. Raik, and M. E. Salehi. Fully-Fusible Convolutional Neural Networks for End-to-End Fused Architecture with FPGA Implementation. In *30th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, pages 1–5. Istanbul, Turkey, 2023.
DOI: https://doi.org/10.1109/ICECS58634.2023.10382831

XVI J. Rostovski, M. H. Ahmadilivani, A. Krivošei, A. Kuusik, and M. M. Alam. Real-Time Gait Anomaly Detection Using 1D-CNN and LSTM. In *Nordic Conference on Digital Health and Wireless Solutions*, pages 260–278. Oulu, Finland, Springer, 2024.
DOI: https://doi.org/10.1007/978-3-031-59091-7_17

Author's Contributions to the Publications

I In I, I was the main author, defined the scope and research questions, conducted a systematic search for the literature review, selected 139 papers to be included, and studied them. I developed a taxonomy for categorizing the identified methods and wrote the manuscript.

II In II, I was the main author, conceptualized the fault resilience analysis, and implemented the method. I conducted the experiments and simulations, analyzed the results, and wrote the manuscript.

III In III, I was the main author, extended the method, and developed the open-source tool. I conducted the experiments and simulations, analyzed the results, and wrote the manuscript.

IV In IV, I was the main author, developed and implemented the methodology. I conducted the experiments and simulations, analyzed the results, and wrote the manuscript.

V In V, I was a co-author and contributed to conceptualizing the methodology. I analyzed the results and wrote the manuscript.

VI In VI, I was the main author, designed and implemented the method for model-level fault tolerance. I conducted the experiments and simulations, analyzed the results, and wrote the manuscript.

VII In VII, I was the main author, I designed and implemented LSTMs for gait analysis as well as fault resilience assessment and enhancement. I conducted the experiments and simulations, analyzed the results, and wrote the manuscript.

VIII In VIII, I was a co-author, initiated the topic, and contributed to designing and implementing the method. I analyzed the results and wrote the manuscript.

Abbreviations

ABFT	Algorithm-Based Fault Tolerance
AI	Artificial Intelligence
ALU	Arithmetic Logic Unit
ASIC	Application-Specific Integrated Circuits
BNN	Binarized Neural Networks
BER	Bit Error Rate
CNN	Convolutional Neural Network
COTS	Commercial Off-The-Shelf
CPU	Central Processing Unit
CVV	Candidate Vulnerability Values
CVF	Channel Vulnerability Factor
DEF	Data Extraction Form
DHA	DNN Hardware Accelerator
DNN	Deep Neural Network
DL	Deep Learning
DMR	Dual Modular Redundancy
DUE	Detected Unrecoverable Error
ECC	Error Correction Code
EDAC	Error Detection and Correction
EDM	Error Distribution Map
FF	Flip Flop
FPGA	Field-Gate Programmable Logic
FI	Fault Injection
FIT	Failure In Time
GAN	Generative Adversarial Network
GPU	Graphical Processing Unit
HDL	Hardware Description Language
HW	Hardware
HLS	High-Level Synthesis
IC	Integrated Circuits
IFMap	Input Feature Map
IoT	Internet of Things
ISA	Instruction Set Architecture
IVF	Instruction Vulnerability Factor
IP	Intellectual Property
KD	Knowledge Distillation
KVF	Kernel Vulnerability Factor
LSTM	Long Short-Term Memory
LCU	Lightweight Correction Unit
LVF	Layer Vulnerability Factor
MAC	Multiply-and-Accumulate
MBU	Multi-bit Upset
MEA	Mean Absolute Error
ML	Machine Learning
MLP	Multi-Layer Perceptron
mPA	mean Average Precision
MTTF	Mean Time to Failure

MWTF	Mean Work To Failure
MPSoC	Multi-Processor System-on-Chip
NPU	Neural Processing Unit
OFMap	Output Feature Map
OSR	Open-Set Recognition
PE	Processing Element
PC	Personal Computer
PVF	Program Vulnerability Factor
QNN	Quantized Neural Network
RNN	Recurrent Neural Network
RTL	Register-Transfer Level
RQ	Research Question
SA	Systolic Arrays
SBU	Single Bit Upset
SDC	Silent Data Corruption
SER	Soft Error Rate
SEU	Single Bit Upset
SFI	Statistical Fault Injection
SIHFT	Software Implemented Hardware Fault Tolerance
SIMD	Single Instruction, Multiple Data
SLR	Systematic Literature Review
SM	Streaming Multiprocessors
SW	Software
SoC	System-on-Chip
TPU	Tensor Processing Unit
TMR	Triple Modular Redundancy
ViT	Vision Transformers
VF	Vulnerability Factor
VVSS	Vulnerability Values Search Space
VVR	Vulnerability Value Range

1 Introduction

Computer science is one of the most prominent achievements of human life throughout history and has revolutionized all aspects of our activities, including but not limited to communications, manufacturing, business, education, transportation, etc. One of its most significant recent outcomes that is increasingly penetrating into various applications is Artificial Intelligence (AI). AI is an outstanding breakthrough in computer science that allows computer machines to acquire intelligence and perform decision-making as human does [172].

Machine Learning (ML) is a primary branch of AI that enables computer systems to gain the ability to learn and solve complex problems efficiently. Deep Neural Networks (DNNs) are brain-inspired algorithms that can learn and analyze complex patterns from data and provide highly accurate results for many complex tasks such as image classification, object detection, regression, signal processing, prediction, etc. [223]. Fig. 1.1 illustrates an overview of AI, ML, and Deep Learning (DL) and generally depicts their different branches and divisions based on [172].

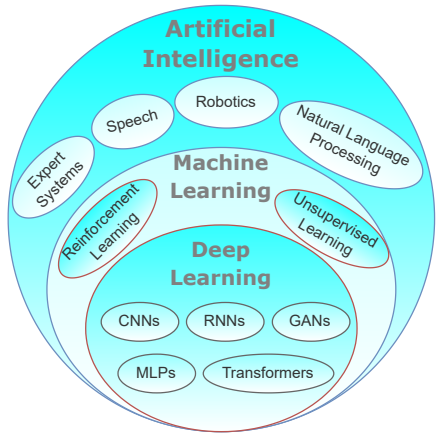


Figure 1.1: An overview of AI, ML, and DL.

With the evolution of powerful processing systems in the recent decade, DNNs have emerged to be larger and deeper, employed in an ever broader extent of domains. Due to the high capabilities of DNNs in solving various tasks with a high accuracy, they are widely adopted in safety-critical applications [41, 84]. Safety is the avoidance of unacceptable risk. Risk is the probability of occurrence and the severity of a physical injury to the health of people as a result of the system failure [133]. Safety-critical applications refer to the use cases in which the failure of a system could result in consequences considered as *unacceptable* and affect the well-being of the people surrounding the system [134].

As AI and DNNs are increasingly leveraged in safety-critical applications such as automotive, space, healthcare, avionics, etc., the significance of safety is drawing more attention. In compliance with the US National AI Initiative Act [2] and the European Union's AI Act [6], any AI system that may pose a threat to people's health and safety must be rigorously evaluated and tested before deployment which reflects the importance of safety assessment for AI systems deployment.

One of the Major concerns in designing a system for safety-critical applications is hardware reliability [190]. Hardware reliability is a characteristic of digital systems expressing the probability of perfectly performing their required function in given conditions within a time interval [37]. Hardware reliability is jeopardized by faults that can be caused by intrinsic characteristics of chips such as manufacturing defects, process variation, aging, or environmental issues such as radiation, temperature variation, and electromagnetic disturbances [41, 173, 213]. Faults influence logic circuits or memory cells of digital devices by altering their functionality during deployment, which may lead to a system failure [173]. By transistor scaling, the intrinsic fault rate of modern digital systems is drastically increasing, exacerbating their vulnerability to faults [120, 212].

DNNs are deployed by various hardware devices and accelerators including Field-Programmable Gate Arrays (FPGAs), Application-Specific Integrated Circuits (ASICs), Graphics Processing Units (GPUs), and Central Processing Units (CPUs) [169, 227]. Hardware reliability of DNNs concerns the ability of DNN accelerators to operate correctly in the presence of faults during the deployment. Hardware faults can modify the operations or parameters of DNNs, producing errors that may propagate to their outputs and lead to misclassification, which may result in a catastrophe if safety-critical applications are concerned. It has been shown in several studies that DNNs are highly susceptible to faults and their accuracy can significantly drop with a few random bitflips [42, 80, 131]. As an example, Fig. 1.2 illustrates an autonomous vehicle possessing digital devices for executing DNNs to perform object detection. During the driving operation, faults influence the operations, and as a result, pedestrians are wrongly detected. Therefore, the vehicle continues driving and leads to a potentially fatal risk situation.

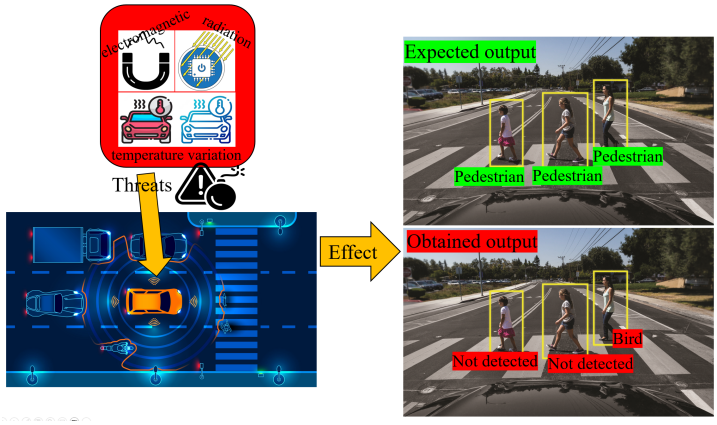


Figure 1.2: Potential impact of faults in DNN accelerators in a safety-critical application.

To address the hardware reliability concerns in DNN deployment, their fault resilience should be first assessed and then enhanced. *Reliability assessment* is the process of measuring the reliability of a modeled or presented system with quantitative evaluation metrics. It is the first step to achieving a fault-tolerant design and a reliable deployment. Generally, the reliability assessment of a system can be conducted by three methods: Fault Injection (FI), analytical, and hybrid methods [83]. In FI methods, faults are injected into the system implemented either in software or hardware, during the simulation or execution. In analytical methods, the function of the system and its reliability is mathematically modeled, based on the target architecture. In hybrid methods, a combination of FI and analytical methods is developed to estimate the reliability. Reliability assessment using

FI methods is generally more realistic and accurate than analytical and hybrid methods; however, FI is a non-scalable process with a high computational complexity and execution time [201].

Reliability assessment not only provides reliability measurement but also identifies the vulnerabilities in a DNN and its accelerator. *Reliability enhancement* is the process of mitigating the vulnerability of DNNs against hardware faults leading to a fault-tolerant design. Redundancy (in time, space, or data) is the key to achieving fault tolerance which can be applied at different levels of system abstraction from the software level down to the transistor level, depending on the application and system constraints [173, 190]. Reliability assessment and enhancement are tightly coupled to each other, necessitating an elegant and accurate process to ensure the safety of the system in deployment. In this thesis, some of the most significant challenges of reliability assessment and enhancement for DNNs are identified and addressed to enable extending the exploitation of DNNs in safety-critical applications with reliable deployment.

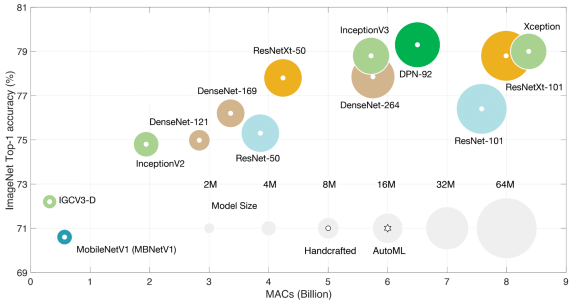
1.1 Motivation

According to IBM's global AI adoption index in 2023 [5], 81% of enterprises in the automotive sector and 72% in the healthcare sector are either actively deploying AI or exploring its integration into their business operations. Furthermore, projections indicated that the AI global market size in the automotive industry will expand from USD 3.22 billion in 2023 to USD 35.7 billion by 2033 [4], while in the healthcare industry, it is expected to grow from USD 19.5 billion in 2023 to USD 490 billion by 2032 [3]. These numbers reflect the increasing need for AI as well as DNN deployment in the markets of safety-critical applications, and therefore, reliability assessment and enhancement are essential issues to address.

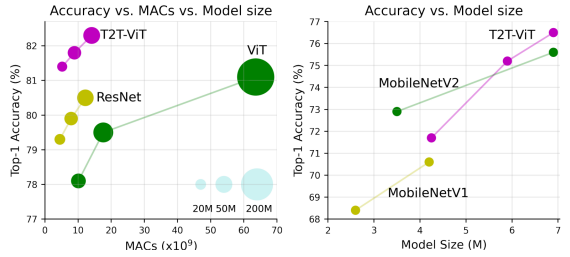
With the growth of DNN exploitation, the size of emerging DNNs in terms of the amount of parameters and computations is rapidly rising. Fig. 1.3 illustrates the size of emerging DNNs in terms of the number of parameters and Multiply-and-Accumulate (MAC) operations for CNNs [75] and Vision Transformers (ViT) [246]. It is observed that emerging DNNs possess billions of parameters and require billions of computations to achieve high accuracy. The gigantic size of DNNs places a huge complexity on their reliability assessment and enhancement, necessitating efficient and innovative solutions to reduce this complexity and overheads.

Several papers have been published on reliability assessment and enhancement in recent years [168, 220]. Due to the variety of DNN models and accelerators, different papers have considered their own customized setups to address their identified challenges. As the extent of DNNs and their accelerators is increasingly widening, their reliability study requires various techniques to enable specialized reliability assessment and enhancement. This diversity has led to an ambiguous space for researchers to comprehend the methods for hardware reliability assessment of DNNs and their accelerators. Therefore, a Systematic Literature Review (SLR) is required to allow comprehension of the essentials of studying the reliability of DNNs. Throughout the literature, there is no survey characterizing existing solutions for the reliability assessment of DNNs, hindering researchers from identifying the challenges and gaps in this area and addressing them.

By overviewing the literature, it can be observed that the hardware reliability of DNNs is mainly assessed by the means of FI [203]. However, given the huge size of emerging DNNs and the complexity of their accelerators, FI is prohibitively complex and impractical. Achieving statistically high-confident results using FI requires weeks and months of simulations by powerful GPUs [200]. Therefore, one of the main challenges in the literature is to tackle the high complexity of FI by devising alternative scalable methods for



(a) Convolutional Neural Networks [75]



(b) Vision Transformers (ViT) [246]

Figure 1.3: Growing size of emerging DNN models regarding their computations and memory requirements.

evaluating and quantifying the fault resilience of DNNs.

DNNs are inherently resilient to faults and they can mask a huge amount of faults, yet their accuracy is considerably compromised by faults [147, 176]. Fault tolerance can be achieved by redundancy either at hardware or software levels [135]. As mentioned, hardware devices, including general-purpose processors (e.g., CPUs and GPUs) and specialized accelerators (e.g., FPGAs and ASICs), are exploited to efficiently deploy DNNs [169]. Safety-critical applications such as autonomous vehicles are characterized as edge applications where the processing of data is performed at the edge of the network instead of exploiting external service providers as in cloud computing [167]. In many cases in edge AI applications, hardware accelerators are not configurable and it is not conceivable to modify the underlying hardware to enhance the fault tolerance of the deployed DNNs; especially with the general purpose accelerators as well as with off-the-shelf Integrated Circuits (ICs) and hard cores. In these cases, fault tolerance can be achieved by Software-Implemented Hardware Fault Tolerance (SIHFT) [98]. In the case of DNNs, it can be obtained by introducing redundancy to the DNN models' architecture (e.g., parameters, functions, and operations) without redesigning the underlying hardware.

On the other hand, achieving fault tolerance through redundancy lays overhead to the system pertaining to the DNN's memory footprint and execution performance [123, 213]. Whereas hardware accelerators in edge computing pose several constraints in terms of latency, power, and memory size [167]. Therefore, the trade-off between DNNs' reliability and memory/performance is a crucial consideration in DNNs fault tolerance. Conventional fault-tolerant techniques such as Triple Modular Redundancy (TMR) or Dual Modular Redundancy (DMR) introduce a massive overhead to DNNs which are already gigantic, making them inefficient compared to their effectiveness. Therefore, there is a need for

cost-effective model-level approaches to enhance the fault resilience of DNNs.

1.2 Problem Formulation and Research Questions

In the frame of this thesis, the objective is to identify the major challenges in the literature concerning hardware reliability assessment and enhancement for DNNs in safety-critical applications. In this regard, four problems (P) are identified as follows:

- **P1. Ambiguity in the reliability assessment literature for DNNs**: Due to the variety of DNNs and accelerators, there exists a wide range of research papers with distinct methods evaluating the reliability of DNNs and their accelerator. Because of the extent of the domain, researchers approach the problem of the reliability of DNNs from various perspectives. We are confronted with a multitude of DNN applications as well as a variety of DNN architectures and accelerators for different tasks. This variety creates an ambiguous research area, restraining researchers from precisely identifying and comprehending the gaps in the literature. Multiple factors affect the variety of reliability evaluations of DNN accelerators leading to different ways of assessment methods and experiments: DNN architectures, hardware accelerators, system abstraction level, and fault models. Existing surveys [123, 168, 203, 213, 220, 229] focus on reliability enhancement methods but there is no comprehensive survey for reliability assessment methods. To tackle this ambiguity in the literature, there is a need for a comprehensive literature review to characterize the existing methods and identify the gaps. The Research Questions (RQ) regarding this problem are as follows:

  - **RQ1.1.** What are the existing methods in this domain?
  - **RQ1.2.** How could the existing methods be characterized?
  - **RQ1.3.** What are the open challenges in this domain?

- **P2. Scalability of reliability assessment:** Fault injection is the major method that is employed for hardware reliability assessment for CNNs. It is a realistic method and provides the possibility of modeling various fault types and injecting them at diverse system abstraction levels from software down to RTL level. Nonetheless, since emerging CNNs are gigantic and their accelerators are complex, FI's non-scalability leads to an obstacle to reliability assessment. Achieving high-confidence and accurate results by FI for CNNs takes weeks and months. On the other hand, existing analytical resilience analysis methods for DNNs [27, 28, 161, 206, 210, 237] could not tackle this gap in the literature. Although they are faster than FI, they do not quantify reliability and are not as accurate as FI. To our knowledge, there is no scalable, fast, and accurate approach to reliability assessment for CNNs. The RQs regarding this problem are as follows:

  - **RQ2.1.** How to quantify the reliability of emerging CNNs as accurately as FI?
  - **RQ2.2.** How to obtain the reliability of fine-grain components of CNNs and identify the more vulnerable ones?
  - **RQ2.3.** How to perform a scalable and fast resilience analysis yet accurate for emerging CNNs?
  - **RQ2.4.** How to apply the resilience analysis outputs for optimized resilience assessment or enhancement methods?

- **P3. Costly fault tolerance:** Fault-tolerant techniques for CNN accelerators are carried out at the architecture and algorithm level [168]. Architecture-level techniques are accelerator-specific and exploit hardware redundancy with performance and memory overhead and they do not apply to general-purpose processors and predesigned IPs. Meanwhile, algorithm-level techniques modify the CNN models in software that any accelerator executes. Throughout the literature, several cost-effective algorithm-level fault tolerance techniques for enhancing the reliability of CNNs are presented [49, 53, 93, 113, 143, 247, 252], yet they are either too complex, or induce considerable overhead, or provide low fault resilience compared to the induced overhead. Therefore, there is a need for low-overhead model-level approaches for CNNs to significantly enhance their deployment reliability. The corresponding RQs concerning this problem are as follows:

  - **RQ3.1.** How to effectively enhance the fault tolerance for off-the-shelf CNN accelerators?
  - **RQ3.2.** How to provide high fault resilience for CNNs with minimal overheads?
  - **RQ3.3.** How to allow trade-off controllability over resilience vs. overhead based on the application's constraints?

- **P4. Missing the study of reliability for Recurrent Neural Networks (RNNs):** Throughout the literature, numerous papers extensively examined the reliability assessment and enhancement of feed-forward DNNs [123, 220]. Nonetheless, the reliability of RNNs, in particular Long Short-Term Memory (LSTM) DNNs, is not explored. However, they are widely deployed in safety-critical applications, particularly in healthcare for diagnosis, treatment, and prediction of diseases and anomalies [165]. Therefore, there is a gap in the literature on the impact of faults on LSTMs and how to improve their fault resilience. The research questions pertaining to this problem are as follows:

  - **RQ4.1.** How vulnerable are LSTM-based DNNs to faults?
  - **RQ4.2.** How to improve their resilience against faults with a low overhead?

1.3 Contributions

This thesis attempts to address the identified problems in the research domain and the raised research questions. Fig. 1.4 summarizes the contributions of this dissertation with respect to the aforementioned problems, that go beyond state-of-the-art as follows:

- **C1. Systematic literature review on hardware reliability assessment for DNNs:** To address *P1* and *RQ1.1-RQ1.3*, the first Systematic Literature Review (SLR) focused exclusively on all methods of reliability assessment of DNNs is conducted. This study establishes a comprehensive picture of the reliability assessment methods for DNNs and systematically reviews the pertinent literature. The primary objective of this review is to explore the methods of reliability assessment for DNNs, generalize and classify them, and identify the existing challenges in the field. To the best of our knowledge, this survey constitutes the first in-depth literature review concerning reliability assessment methods for DNNs. The review encompasses all relevant papers published from 2017 to 2022, identified through a systematic search. The key highlights of this contribution which is based on paper I are:

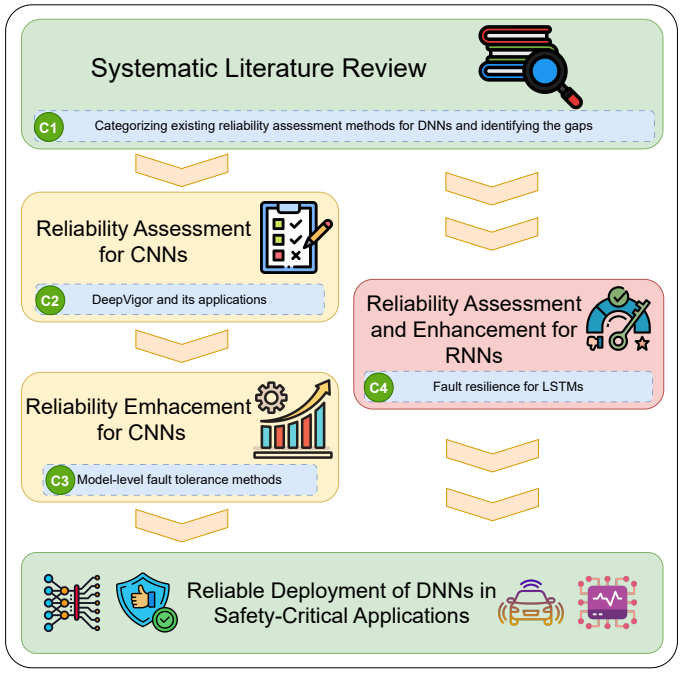


Figure 1.4: An overview of the contributions of the thesis.

- Presenting a thorough and systematic survey on the reliability assessment methods for DNNs;
- Analyzing publication trends across different years and methods;
- Classifying and characterizing reliability assessment methods for DNNs;
- Categorizing fault injection methods based on the DNN hardware platforms;
- Introducing analytical and hybrid reliability assessment methods as alternatives to FI;
- Highlighting open challenges in the field and proposing recommendations for future research.

■ **C2. Scalable, fast, and accurate resilience analysis for CNNs:** This contribution addresses *P2* and *RQ2.1-RQ2.4* by introducing novel semi-analytical methods for reliability assessment of CNNs, DeepVigor and its optimized version DeepVigor+. In this thesis's contribution, the concept of neurons' vulnerability ranges is first introduced in DeepVigor, which indicates whether a fault in the output of neurons would lead to a CNN misclassification. This enables a comprehensive reliability study through a novel resilience analysis method, where Vulnerability Factors (VF) of layers, neurons, and bits within CNNs are derived. The key highlights of this contribution, which is based on paper II, are as follows:

- Proposing DeepVigor, an innovative, metric-oriented, and accelerator-agnostic resilience analysis method for CNNs that is faster than FI with comparable accuracy;

- Introducing and calculating vulnerability ranges for all neurons in CNNs, supported by a fault propagation analysis, resulting in accurate categorization of critical/non-critical faults;
- Providing fine-grain VF as reliability quantification metrics for layers, neurons, and bits in CNNs, validated through comparison with fault injection;

The DeepVigor method is further optimized as DeepVigor+ to overcome the scalability challenges in reliability analysis. DeepVigor+ leverages an optimal fault propagation analysis across neurons and entire CNNs to acquire VFs in an optimized manner. To the best of our knowledge, DeepVigor+ represents the first scalable semi-analytical alternative to FI with a comparable accuracy for resilience analysis of CNNs in the literature. Additionally, DeepVigor+ is made available as an open-source tool, facilitating reliability analysis for emerging CNNs and enabling researchers to quickly evaluate CNNs' reliability and design fault-tolerant CNNs. The key highlights of this contribution which is based on paper III, are as follows:

- Introducing DeepVigor+, a scalable, fast, and accurate resilience analysis for deriving VFs for CNNs' layers and models by analyzing both parameters and activations and utilizing an optimal error propagation analysis;
- Employing a novel statistical approach in DeepVigor+ based on stratified sampling, enabling fast and accurate resilience analysis for emerging CNNs.

Furthermore, QDeepVigor is an extended version of DeepVigor for Quantized Neural Networks (QNNs) and its analysis results are exploited for various purposes:

- Cross-layer reliability enhancement for QNNs accelerators: In this work, which is based on paper IV, the more critical neurons in a QNN are identified by QDeepVigor and are split into two equivalent neurons. Then a Lightweight Correction Unit (LCU) is designed for Systolic Arrays (SAs) to correct faults in the identified critical neurons. The results indicate the same fault resilience as TMR with half overhead;
- A hybrid method for QNNs' reliability assessment: In this work, which is based on paper IX, QDeepVigor is exploited to prune the fault space using the vulnerability value ranges for neurons in QNNs executing on an SA. This hybrid reliability assessment method leads to a significant speed-up in fault simulation.

■ **C3. Cost-effective model-level fault-tolerance for CNNs:** This contribution attempts to address *P3* and *RQ3.1-RQ3.3* by proposing model-level fault tolerance methods for CNNs as SIHFT-based techniques. In the first method, a novel low-cost activation restriction method, called ProAct, is introduced and combined with progressive training based on Knowledge Distillation (KD) [112] to achieve significant resilience in CNNs with minimal memory overhead. The source code for ProAct, along with other state-of-the-art activation restriction methods, is published and made publicly accessible for researchers for the first time. The key highlights of this work which is based on paper V, are as follows:

- Proposing Hybrid Clipped ReLU (HyReLU) activation function, which restricts activation values by trainable threshold parameters in a neuron-wise manner at the last layer and operates layer-wise in the other layers of CNNs;

- Introducing progressive training to acquire clipping thresholds in HyReLU for each layer individually, resulting in more optimal and effective clipping threshold values that ensure high resilience for CNNs.

The second method introduces an innovative model-level hardening solution that modifies the architecture of CNNs to inherently enable fault correction during inference. An efficient error correction mechanism is designed, achieved by selectively duplicating channels within the CNNs structure. In this approach, the vulnerability of the weight channels of CNNs is analyzed, and the most vulnerable ones are duplicated. Subsequently, a correction layer detects and corrects erroneous outputs based on duplicated values. The key highlights of this work, which is based on paper VI, are as follows:

- Proposing a model-level hardening method for CNNs to improve their fault tolerance during inference. The approach involves duplicating parameters in most vulnerable channels and integrating an efficient Error Detection and Correction (EDAC) layer to correct erroneous feature maps;
- Introducing a channel pruning technique based on the parameter vulnerability, allowing significant reductions in the overhead associated with the hardening technique;
- providing a set of user-defined constraints based on the application to manage the trade-off between performance and memory overheads vs. the expected fault resilience.

Eventually, all proposed methods for reliability assessment and enhancement of CNNs in this thesis, are integrated into the SentinelNN framework. This open-source framework obtains channel vulnerability factors for CNN models using DeepVigor+ to perform vulnerability-aware pruning and hardening. SentinelNN hardens the CNN models by incorporating selective channel duplication and correction as well as activation restriction.

- **C4. Low-overhead resilience assessment and enhancement for LSTMs:** This contribution addresses *P4* and *RQ4.1-RQ4.2* by analyzing and improving the fault resilience of LSTMs for healthcare applications. To our knowledge, this gap is addressed for the first time. The key highlights of this contribution which is based on papers VII and VIII, are as follows:

  - Conducting a comprehensive analysis of the resilience of various LSTM-based DNNs using fault injection into their parameters;
  - Proposing multiple fault-tolerant techniques for LSTM-based DNNs, resulting in remarkably mitigating the impact of faults on LSTM-based DNNs;

1.4 Thesis Organization

This thesis consists of 7 Chapters. Chapter 2 presents a thorough background on the theoretical aspects of the dissertation. It discusses the definition and concept of hardware reliability and explains the impacts of hardware faults and soft errors and how they can be measured and mitigated. Furthermore, the basics of DNNs and their accelerators are presented. Chapter 3 presents a systematic literature review on the reliability assessment methods for DNNs. This Chapter thoroughly categorizes the existing methods and identifies the gaps in the literature.

Chapter 4 presents a novel scheme for fault resilience analysis for CNNs, called DeepVigor, and tackles the scalability of conventional methods based on fault injection. DeepVigor is exploited not only for fault resilience but also for fault enhancement of accelerators. Chapter 5 proposes SIHFT-based methods for enhancing the fault resilience of CNNs, beyond state-of-the-art. It presents ProAct for optimal activation restriction with a minimal overhead. Furthermore, this Chapter introduces EDAC layers accompanied by selective channel-duplication in CNNs, which is facilitated by vulnerability-aware pruning for a cost-effective fault tolerance in CNNs. This Chapter also introduces the SentinelNN framework that integrates the proposed methods for reliability assessment and enhancement in this dissertation. In the following, Chapter 6 investigates fault resilience assessment and enhancement for various LSTM-based DNNs. Finally, Chapter 7 concludes the thesis.

2 Background

To convey the essence of this thesis, several concepts must be explained. This Chapter presents the theoretical and technical background for the thesis in detail.

2.1 Hardware Reliability

The rapid increase in software and hardware complexity has been driven by the tremendous advances in electrical and computer technology in the past century. The growth of microelectronics is generally associated with the evolution of the microprocessor, often described as "Moore's Law". Moore predicted that chip complexity would double every two years, which has been on track for the past decades [228]. In this context, transistor miniaturization has resulted in high-performance computing systems, which are also expected to operate safely and reliably.

Since microelectronics have permeated all aspects of our daily lives, it is crucial to employ them safely and reliably. However, several threats can compromise their safety and reliable operation. Hardware faults are a major threat to the safety of digital devices, especially in their safety-critical applications. The following subsections describe the concept of hardware reliability, its associated threats (hardware faults), and methods to evaluate and improve it.

2.1.1 Definition and Concept

Hardware reliability of a system is defined as the probability that no failure or user-visible error occurs in its functional operation over a given period [37, 173, 214]. Reliability is a function of time and represents the likelihood that a system can operate as intended for a specific duration. As stated in reference [173], reliability is expressed as $R(t) = P(T > t)$, where T denotes the lifetime of a system. If a population of N_0 identical systems is considered, $R(t)$ is equal to the fraction of the systems that survive beyond time t. Supposing N_t is the number of survived systems until time t, and $E(t)$ is the number of systems that encountered errors in the interval $(0,t]$, then, reliability can be expressed as Eq. (2.1).

$$R(t) = \frac{N_t}{N_0} = \frac{N_0 - E(t)}{N_0} = 1 - \frac{E(t)}{N_0} \tag{2.1}$$

Differentiating Eq. (2.1) yields the instantaneous error rate i.e., hazard rate $h(t)$ which is defined as the probability that a system experiences an error during a time interval Δt, given that it has prevailed until time t. It means that $h(t)$ is the probability of an error occurring in the time interval $(t, t+t]$. Therefore, we can extend the mathematics as noted in Eq. (2.2) [173].

$$\begin{gathered}
\frac{dR(t)}{dt} = -\frac{\frac{dE(t)}{dt}}{N_0} \\
h(t) = P(t < T < t + \Delta t | (T > t)) = -\frac{\frac{dE(t)}{dt}}{N_t} = \frac{-\frac{E(t)}{N_0}}{\frac{N_t}{N_0}} = \frac{\frac{dR(t)}{dt}}{R(t)} \\
\Rightarrow \frac{dR(t)}{dt} = -h(t)R(t)
\end{gathered} \tag{2.2}$$

The solution to this differential equation is written in Eq. (2.3) [173].

$$R(t) = e^{-\int h(t)dt} \tag{2.3}$$

Assuming that $h(t)$ has a constant failure rate (λ) during its useful lifetime phase and the system is not affected by aging, the reliability of the system can be expressed as Eq. (2.4) [173].

$$R(t) = e^{\lambda t} \tag{2.4}$$

To elaborate on the concept of reliability, we refer to an example from [214]. Consider 50 systems being tested over 1,000 hours, with two failures occurring during the test. The probability of failure, P_f, for this system over 1,000 hours of operation is calculated as $P_f(1,000) = \frac{2}{50} = 0.04$. The probability of success, known as the reliability R, is given by $R(1,000) = 1 - P_f(1,000) = 0.96$. The failure rate, f_r, for this system is $f_r = \frac{2}{50 \times 1,000} = 4 \times 10^{-5}$, representing the probability of failure of an instance for an individual system in 1,000 hours of operation. Using the reliability function in Eq. (2.4):

$$R(1,000) = e^{-4 \times 10^{-5} \times 1,000} = 0.96 \tag{2.5}$$

which is consistent with the previous calculation.

2.1.2 Hardware Faults: Definition, Classification, and Impact

In a computer system, the abstraction layers are generally divided into six broad categories, organized from top (software) to bottom (hardware), as shown in Fig. 2.1: user application, operating system, firmware, architecture, circuits, and process technology. Faults typically originate at the process technology or within the silicon chip itself. Hardware faults can result from manufacturing defects in a silicon chip or environment interactions such as bitflips caused by cosmic ray strikes [173].

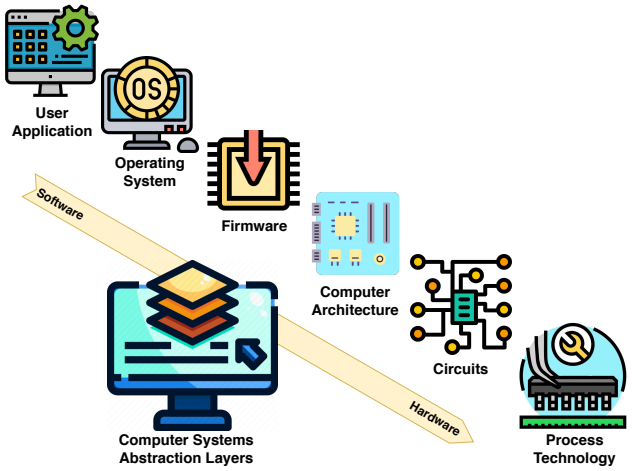


Figure 2.1: Abstraction layers of computer systems, inspired by [173].

Faults are generally categorized into three classes: permanent, intermittent, and transient. Permanent faults persist indefinitely until a correction mechanism is applied, such as oxide wearout leading to a transistor failure in a silicon chip. Intermittent faults repeatedly show up and vanish, as seen in the case of partial oxide wearout. Transient faults, on the other hand, appear briefly and then disappear, such as bitflips in memory or gate malfunctions in logic caused by alpha particles or neutron strikes [173].

Hardware faults may not always result in a user-visible error for two reasons. First, a fault may be masked in an intermediate layer; for instance, a defective transistor caused by oxide wearout might impact performance but not disrupt the correct functioning of the architecture. Second, different layers within the system might be designed with fault tolerance. For example, radiation-hardened cells can detect and recover from faults in transistors. Each abstraction layer in Fig. 2.1 can thus be designed to handle faults originating from lower layers. If a fault is managed at a specific layer, it prevents errors from propagating to higher layers [173].

Errors are the observable consequences of faults. While faults are necessary for errors to occur, not all faults manifest as errors, particularly if they are masked or tolerated. *Failure* refers to a system malfunction that prevents the system from meeting its expected correct outputs. A failure is essentially a specific case where an error becomes visible to the user. Fig. 2.2 illustrates the impact of faults on a system's output. As depicted, when a fault occurs, it is considered *masked* if the affected bit or transistor is not read, or if the corresponding error is corrected. If the fault leads to a user-visible error, it manifests as a *Silent Data Corruption (SDC)*, i.e., failure. If it is detected but not corrected it appears as a *Detected Unrecoverable Error (DUE)*, which may lead to temporary system unavailability [173]. As mentioned in [98], failure's consequences can be categorized in four grades:

- *Benign*: No major impact on the system's task or performance,
- *Significant*: The system's task is disrupted, leading to reduced efficiency of the delivered service,
- *Serious*: The system's task is severely disrupted,
- *Catastrophic*: The system's task is completely halted, resulting in the destruction of the controlled process or causing human injury or death.

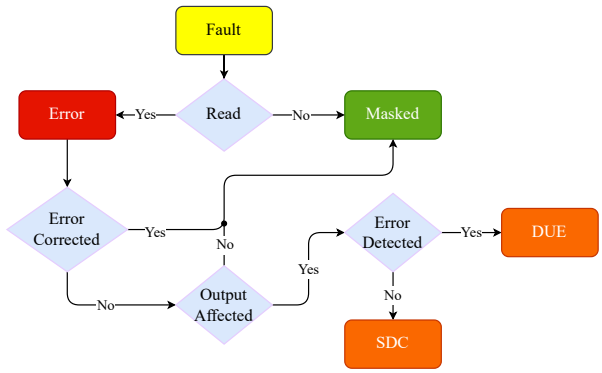


Figure 2.2: Faults impact on the output of a system, inspired by [173].

2.1.3 Soft Errors: Origination and Impact

Transient faults in semiconductor devices can occur due to various factors, including transistor variability, thermal cycling, erratic fluctuations in the minimum operational voltage of a circuit, and soft errors caused by external radiation. Transistor variability stems from random dopant fluctuations, sub-wavelength lithography, and high heat flux amongst the

silicon die. Thermal cycling results from repeated stress due to temperature variations. Erratic fluctuations in the circuit's minimum voltage can occur as a result of gate oxide soft breakdown combined with high gate leakage [173].

Radiation-induced soft errors are a significant threat to system reliability and can lead to circuit malfunctions or system crashes. Soft errors occur when data is corrupted, whereas the device itself remains undamaged. These errors are difficult to trace or identify as the root cause of system malfunctions through failure analysis. Therefore, it is crucial to mitigate their effects in the system design. In terrestrial applications, the primary sources of soft errors are neutrons and alpha particles, produced by cosmic radiation, solar activities, etc. When these high-energy particles strike transistors, they induce energy, generating a current pulse that can result in an upset in transistors [178].

Radiation impacts semiconductors from sea level to space, with neutron flux intensifying at higher altitudes. Consequently, space applications face the highest risk, while terrestrial applications are at a lower risk. It is estimated that the average terrestrial neutron flux that can potentially lead to soft errors is $14\ \frac{neutrons}{cm^2.hour}$. On the other hand, at the altitude of 10 km, neutron flux increases by a factor of 228 compared to sea level. This highlights that digital devices are exposed to soft errors at any altitude, stressing the need for fault tolerance mechanisms, particularly for safety-critical applications [173].

As mentioned, a transistor can collect charge from an alpha particle or a neutron strike. When this charge exceeds the circuit's threshold at the gate or cell level, it results in a bitflip. In memory devices, a transient fault occurs when a bit stored in a cell flips. However, in logic devices, a transient fault is only considered to occur when this fault propagates to a forward latch or storage cell. These bitflips are classified as Single-Event Effects (SEEs), which may impact single or multiple bits in memory cells or latches. For a bitflip and subsequent malfunction to occur, the accumulated charge in a cell or circuit must exceed a minimum threshold, known as the critical charge [173].

Transistor scaling has a remarkable impact on the susceptibility of transistors to soft errors due to its effect on the critical charge. For instance, the Soft Error Rate (SER) in SRAMs increases by a factor of 6-7 from 130 *nm* to 22 *nm* process [120]. This indicates that emerging digital systems are becoming increasingly susceptible to soft errors. SER refers to the rate at which soft errors occur in a device within a specific environment. The SER is measured in Failure In Time (FIT), where one FIT is equal to one failure in 10^9 device hours [178].

2.1.4 Fault Tolerance Techniques

Fault-tolerant computing refers to the ability to maintain correct computation despite the presence of errors in a system, thus enhancing reliability. Achieving fault tolerance primarily involves leveraging and managing redundancy. Redundancy is the property of having an excess of resources beyond the minimum required to perform a task. When failures occur, this redundancy is utilized to mask or bypass the failures, ensuring the system continues to operate at the intended level of functionality [135, 214].

Redundancy can be exploited in different methods at the hardware or software level, in spatial and temporal ways. Hardware redundancy is provided by incorporating extra hardware into the design to either detect or override the effects of a failed component. For example, instead of having a single processor, we can use two or three processors, each performing the same function. By having two processors (Dual Modular Redundancy-DMR), we can detect the failure of a single processor; by having three (Triple Modular Redundancy-TMR), we can use the majority output to override the wrong output of a single faulty processor. These forms of hardware redundancy incur high overheads, and

their use is therefore normally reserved for critical systems where such overheads can be justified [135].

Information redundancy encompasses techniques such as Error Detection and Correction Codes (ECCs) or Algorithm-Based Fault Tolerance (ABFT). ECCs utilize additional bits, i.e., check bits, which are appended to the original data to enable the detection and potential correction of errors. On the other hand, ABFTs integrate redundant computations into algorithms to detect, locate, and correct errors caused by hardware faults during normal operations. Both ECCs and ABFTs are widely employed in memory units and storage devices to protect against benign failures. However, they often necessitate additional hardware to handle the processing of redundant data [135, 235].

Processing components can exploit time redundancy by re-executing the same program, primarily to counteract transient faults. Since most hardware faults are transient, it is unlikely that successive executions will encounter the same fault. This makes time redundancy an effective method for detecting transient faults that might otherwise remain undetected. Additionally, it can be employed when other error detection mechanisms are available, allowing the system to recover from a fault and repeat the computation. While time redundancy imposes significantly lower hardware and software overhead compared to other methods, it lays a substantial performance cost [135].

Software Implemented Hardware Fault Tolerance (SIHFT) refers to a set of techniques that exploit redundancy at the software level to detect and correct hardware faults, without requiring modifications to the hardware design. SIHFT offers a cost-effective alternative to traditional hardware or information redundancy approaches and is particularly beneficial when using Commercial Off-The-Shelf (COTS) microprocessors or predesigned IP cores, which typically lack native or strong error detection and correction capabilities. In SIHFT, the software not only performs its primary designed functions and requirements, but also incorporates monitoring functions to detect, signal, and correct hardware errors when they occur [98, 135].

2.1.5 Evaluation and Metrics

There are many metrics to quantify the reliability of computer systems in relation to hardware faults and their fault tolerance. In this subsection, some of the key metrics that are employed in this thesis are discussed. One of the fundamental metrics to measure is defined by the reliability function, as presented in Eq. (2.4). The error rate of a system is typically expressed in terms of Failure in Time (FIT), where one FIT corresponds to one failure in 10^9 hours. The total FIT rate of a system can be calculated by the summation of the FIT rates of its individual components [173]. It can be expressed as Eq. (2.6) in which FIT_{raw} is provided by the manufacturer, $Size_{component}$ is the total number of the component bits, and $SDC_{component}$ that can be obtained by FI [145].

$$FIT_{accelerator} = \sum_{component} FIT_{raw} \times Size_{component} \times SDC_{component} \tag{2.6}$$

Reliability is also measured by the SDC rate, which reflects the proportion of faults that propagate to the system's output, regardless of its effect. Another essential metric is the Vulnerability Factor (VF), representing the fraction of faults that result in user-visible errors, i.e., failure. VF can be measured across various system levels, from architecture to software programs, by FI simulations or analytical approaches. VF expresses the probability of failure at a given system level, in the case of fault occurrence. These metrics are essential for evaluating the FIT rate calculation of systems [173].

The metric Architectural Vulnerability Factor (AVF), specifically, expresses the probabil-

ity of fault propagating to the output at the architecture level [149]. AVF can be measured through FI, by dividing the number of faults propagated to the output by the total number of injected faults. Furthermore, authors in [150] provide a formula to estimate the cross-section of the configuration memory in (2.7) 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 [150].

$$\sigma = AVF \times (\#UtilizedBits) \times (\frac{\sigma_{static}}{\#MemBits}) \tag{2.7}$$

In radiation experiments, 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 Eq. (2.8) [236]. Cross-section σ is expressed as a unit of cm^2 and is the probability that a particle may cause an observable error [149].

$$\sigma = errors/Flux \tag{2.8}$$

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 Eq. (2.9).

$$SER = \sigma \times \phi \tag{2.9}$$

2.2 Deep Neural Networks

Deep Learning (DL) is a subset of Machine Learning (ML), which focuses on enabling computers to learn how to solve problems without explicit programming. Due to the remarkable learning capabilities of Deep Neural Networks (DNNs), they find applications across a wide range of fields, including image and video processing, data mining, robotics, autonomous vehicles, and gaming [223]. DNNs experience a training phase using a designated dataset before being utilized in their target application during the inference phase.

The training phase is an iterative process conducted once, aimed at updating the parameters of the DNN (e.g., weights, and biases). A loss function (e.g., cross-entropy) is employed during this phase to measure the difference between the expected and the predicted outputs of the DNN. Training function iteratively updates the parameters to minimize the loss function, until the DNN achieves a high accuracy. On the other hand, during the inference phase, representing the deployment of the DNN, the DNN model is executed multiple times using the parameters acquired from the training phase [223].

DNNs consist of numerous interconnected neurons, each of which receives input activations and multiplies them by corresponding weights. The weighted activations are then summed and passed to the neuron's output. A collection of neurons forms a layer, which is typically followed by additional functions such as activation functions (e.g., ReLU, sigmoid), batch normalization, and pooling (e.g., max or average pooling) [223]. DNNs can have various architectures, each suited to different applications. In this subsection, the DNN architectures explored in this thesis are briefly reviewed.

2.2.1 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are widely employed in tasks such as image classification and object detection, consisting of multiple Convolutional (CONV) and Fully-Connected (FC) layers. CONV layers contain a set of multi-dimensional weights, i.e., filters, which extract specific features from the layer's input. A channel represents a set of Input

Feature Maps (IFMaps), which are convolved with the filters to produce Output Feature Maps (OFMaps). FC layers are positioned at the end of CNNs to carry out the classification task [223].

An abstract representation of a neuron in a neural network is illustrated in Fig. 2.3. As shown, inputs enter the network through the input layer. The intermediate or hidden layers determine the network's depth and perform the CNN's core computational functions. The output layer is responsible for decision-making and producing probabilities for each output, i.e., logits or output confidence score. The highest value corresponds to the top-ranked output in a classification task, representing the network's final decision or classification.

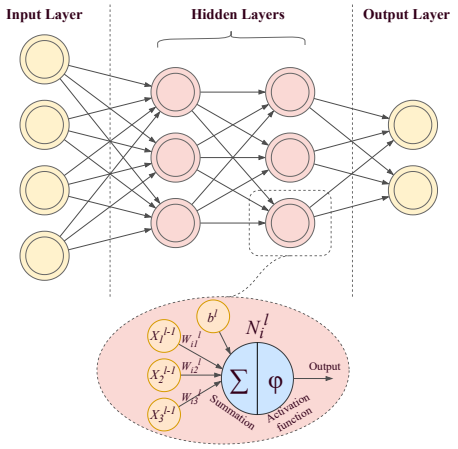


Figure 2.3: Representation of a simple neural network with the detail of a neuron.

Equation (2.10) describes the function of the *i*-th neuron in layer *l* (denoted as N_i^l) which receives input activations from the previous layer *l-1* consisting of *n* outputs (denoted as X^{l-1}). In this equation, *W* and *b* represent weights and bias, respectively, associated with the connections between neurons [223].

$$N_i^l = \phi\left(\sum_{j=0}^{n} X_j^{l-1} \times W_{ij}^l + b^l\right) \tag{2.10}$$

Fig. 2.4 illustrates an abstract view of a CONV layer. As shown, the IFMaps in layer l, with n channels and dimensions $X^l \times X^l$, are convolved with filters that have the same number of input channels and a kernel size of K^l, producing m output channels. During this process, each 3-D filter in an output channel performs a point-wise multiplication with a matching section of IFMaps. The results are summed to generate one value for the respective channel of OFMaps. The 3-D filter is then shifted across the IFMap to produce all values in one 2-D OFMap with the size of $X^l \times X^l$. This operation is repeated with other 3-D filters, resulting in the OFMaps for layer $l+1$.

In the field of CNN research, several models are widely adopted, including LeNet-5 [141], AlexNet [137], VGG [216], and ResNet [110] for image classification, and YOLO [194] for object detection. Prominent datasets that are often utilized for training are MNIST [244], CIFAR [136], and ImageNet [74]. Furthermore, due to the large number of parameters and computations in DNNs, Quantized Neural Networks (QNNs) [119] and Binarized Neural Networks (BNNs) [67] have been presented to reduce complexity, memory foot-

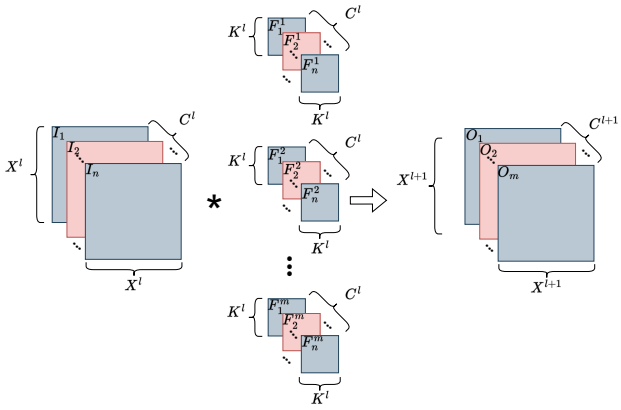


Figure 2.4: Abstract view of a CONV layer

print, and power consumption. These quantized networks reduce the bit-width of DNNs' parameters and computations while maintaining an acceptable accuracy trade-off.

2.2.2 Long Short-Term Memory Neural Networks

Recurrent Neural Networks (RNNs) are a type of DNNs that possess recursive connections across their layers, allowing them to memorize information through time. Long Short-Term Memory (LSTM) neural networks, a subset of RNNs, are particularly proficient at retaining long-term dependencies, making them appropriate for analyzing time-series data [233]. This capability has made LSTMs especially valuable for anomaly detection in healthcare applications, including tasks such as disease diagnosis, treatment monitoring, and the prediction of anomalies [109, 165].

LSTM-based DNNs are composed of multiple cascaded LSTM layers, each containing individual LSTM cells. The operations of a single LSTM cell are depicted in Fig. 2.5 and its functions are mathematically expressed in Eq. (2.11). In this context, x_t is a time-series input at time t, C_{t-1} and h_{t-1} are recursive inputs i.e., the last outputs of the LSTM cell. The parameters U and W represent the weights associated with the input data and recursive inputs, respectively, and B denotes the bias parameters [233].

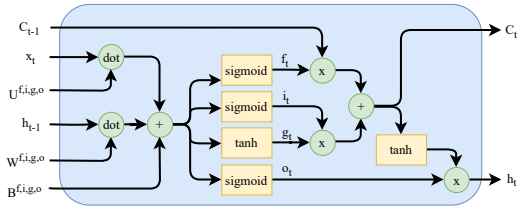


Figure 2.5: Operations in a single LSTM cell (arrows show the data flow).

$$\begin{cases} i_t = sigmoid(x_t U^i + h_{t-1} W^i + B^i) \\ f_t = sigmoid(x_t U^f + h_{t-1} W^f + B^f) \\ g_t = tanh(x_t U^g + h_{t-1} W^g + B^g) \\ o_t = sigmoid(x_t U^o + h_{t-1} W^o + B^o) \\ C_t = f_t \times C_{t-1} + i_t \times g_t \\ h_t = tanh(C_t) \times o_t \end{cases} \tag{2.11}$$

As discussed, an LSTM has two sets of weight parameters: 1) U, which is applied to the input data, to extract features from the current input data, and 2) W, which is applied to the recursive inputs, to manage information from previous outputs over time, enabling the LSTM cells to retain and utilize relevant features. The structure of an LSTM-based DNN varies depending on the specific task. In the case of time-series data classification, an FC layer can be added after the LSTM layer to perform the classification task. Additionally, CONV layers can be incorporated within the LSTM network to enhance feature extraction capabilities [245].

2.3 DNN Hardware Accelerators

DNNs are trained and deployed in their target application using DNN Hardware Accelerators (DHAs). These accelerators are specifically designed to leverage parallelism to enhance the performance of DNN deployment during both training and inference phases. DHAs are typically classified into four main categories: Field-Programmable Gate Arrays (FPGAs), Application-Specific Integrated Circuits (ASICs), Graphics Processing Units (GPUs), and Central Processing Units (CPUs) [169, 227].

FPGAs are the most commonly utilized DHA compared to other platforms, primarily due to their design flexibility across various applications [169, 227]. FPGAs are programmed through configuration bits that define their functionality. The architecture of FPGA-based DNN accelerators typically incorporates a host CPU and an FPGA component, with corresponding interconnections between the two. In this design model, the core functions of DNN are implemented on the FPGA, while the CPU manages the accelerator's instruction as well as data transfer, as both components are integrated with memory. A typical architecture of an FPGA-based DHA is illustrated in Fig. 2.6, which employs a Hardware/Software (HW/SW) co-design approach, effectively separating the DNN implementation on the integrated CPU (software) and the FPGA (hardware) that communicate with each other [105, 117].

ASIC-based DHAs offer superior performance and power efficiency compared to FPGAs; however, they lack flexibility for various applications and involve longer design time. An example of a spatial architecture model is illustrated in Fig. 2.7, which consists of 2D systolic arrays of Processing Elements (PEs) that flow data both horizontally and vertically between input/weight buffers and output buffers. PEs conduct Multiply-Accumulate (MAC) operations on inputs and weights, thus performing the functionality of neurons in DNNs. Off-chip memory is necessary to store DNN parameters and intermediate results generated by the PEs. One notable ASIC-based DNN accelerator is the Tensor Processing Unit (TPU) developed by Google, which utilizes this architectural framework [76, 127].

GPUs are a strong platform for both training and inference of DNNs, incorporating thousands of parallel cores that enhance their efficiency for DNN algorithms, particularly for training. GPUs are designed to execute multiple threads of a program simultaneously. Since DNNs are constructed of numerous independent computations, GPU architecture

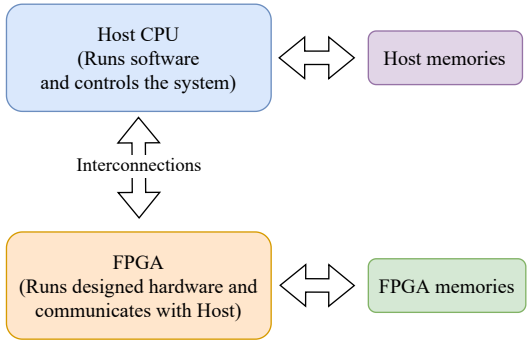


Figure 2.6: Typical structure of an FPGA-based DNN accelerator [105].

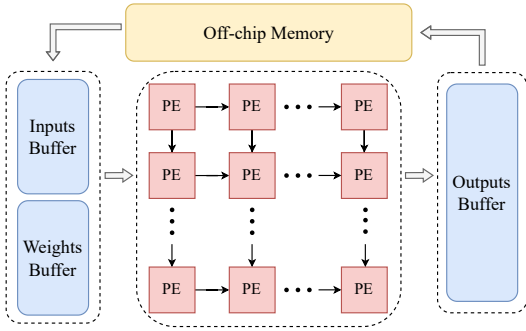


Figure 2.7: An example of a spatial architecture for ASIC-based DNN accelerators [170].

has led to their vast exploitation for accelerating DNN execution. The general architecture of GPUs is depicted in Fig. 2.8. Numerous Streaming Multiprocessors (SMs) are present within a GPU, each containing several cores that share register files and caches. A scheduler and dispatchers manage the distribution of tasks among the SMs and their cores [76, 122, 227].

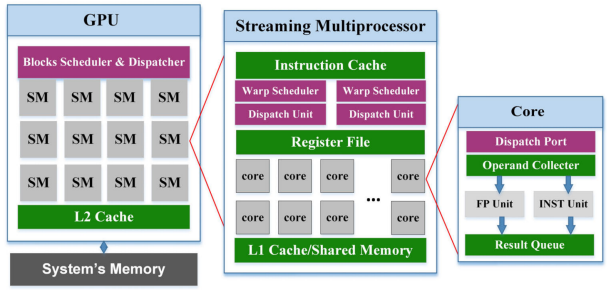


Figure 2.8: General architecture of CUDA-based GPUs [122].

CPUs are general-purpose processors capable of executing a diverse range of algorithms, including DNNs. Multi-core processors are particularly employed for deploying DNNs in edge computing and Internet of Things (IoT) applications. CPUs can execute SIMD (Single Instruction, Multiple Data) instructions, utilizing multiple ALUs (Arithmetic Logic Units) simultaneously. Therefore, they enable parallel computing and low power con-

sumption, thus supporting a broader range of applications for DNNs. However, CPUs are generally not the preferred option for DNNs deployment, due to their relatively lower performance and efficiency compared to specialized hardware accelerators [159, 169, 209]. Although new CPU architectures incorporate Neural Processing Units (NPUs) to accelerate the performance for DNN deployment [51, 189].

Fig. 2.9 illustrates an overview of the attributes of DNN accelerators in terms of flexibility, reconfigurability, and power efficiency [51]. CPUs and GPUs are general-purpose platforms that provide high reconfigurability at runtime, making them suitable for a wide range of DNN applications. However, these platforms suffer from significant power consumption and high data transfer latency between PEs and off-chip memory. In contrast, ASICs and FPGAs can be tailored to optimize performance for specific DNN models, achieving superior power efficiency. This optimization, however, comes at the expense of reduced reconfigurability, limiting their ability to efficiently support diverse DNN models on a single platform.

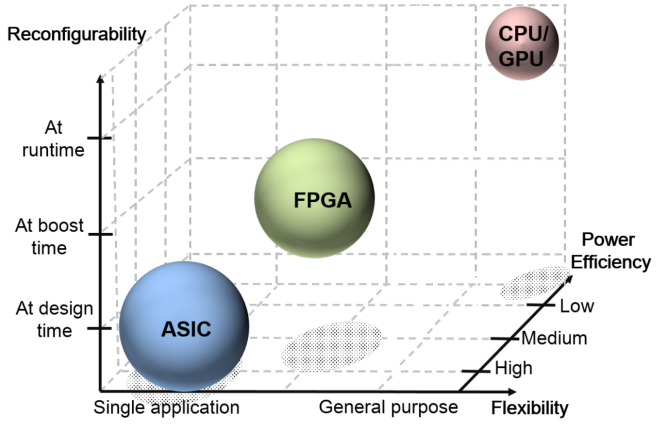


Figure 2.9: An overview of the attributes of DNN accelerators [51].

3 Literature Review on the Reliability Assessment Methods for DNNs

Reliability assessment is the first step towards achieving fault tolerance. As mentioned in Chapter 1, due to the extent of the domain of DNN applications, accelerators, and fault models different papers have studied the reliability of DNNs in a distinct method. This diversity has led to an ambiguous space for researchers to comprehend the methods for hardware reliability assessment of DNNs and identify existing gaps. Therefore, a comprehensive literature review is required to tackle this problem. This Chapter aims to review the literature on hardware reliability assessment methods for DNNs. This Chapter attempts to address P1 which includes RQ1.1-1.3 and presents contributions mentioned in C1, in Chapter 1. This Chapter is based on the following publication:

I 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*, 56(6):1–36, 2024

In the rest of this Chapter, Section 3.1 defines the terminology in this domain, 3.2 explains the adopted methodology to conduct a comprehensive literature review, Section 3.3 presents the characterized taxonomy for existing methods in the literature and outlines the research trend of the field which are discussed in details in Sections 3.4, 3.5, and 3.6. Afterward, Section 3.7 discusses the identified gaps in the literature and eventually, Section 3.8 concludes this Chapter.

3.1 Terminology

The terms 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 thesis does not relate to the reliability and test in software engineering, reliability and robustness of DNN algorithms' accuracy in terms of safely classifying corner cases or out-of-distribution, or security issues e.g., adversarial or bitflip attacks in which an attacker perturbs the inputs or parameters.

3.2 Literature Review Methodology

Systematic Literature Review (SLR) is a standard methodology for reviewing the literature in a recursive process and minimizing bias in the study [59, 140, 227], which is adopted in this Chapter. The SLR methodology determines:

- Specifying the SLR's Research Questions (SLR-RQs),
- Specifying the search method for finding and filtering the related papers,
- Extracting corresponding data from the found papers based on the SLR-RQs,
- Synthesizing and analyzing the extracted data.

Based on the aforementioned steps in SLR methodology, the SLR-RQs that we attempt to answer are:

- **SLR-RQ1:** What is the distribution of the research papers in the domain of DNNs reliability assessment? (To obtain the trend of publications in this domain).
- **SLR-RQ2:** What are the existing methods of reliability assessment for DNNs? (To comprehend the entire variety of methods in this domain).
- **SLR-RQ3:** How could the existing methods be characterized and categorized in terms of reliability assessment methods? (To categorize existing works and provide the taxonomy, and systematic instruction for finding the suitable method for potential applications in this domain).
- **SLR-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 3.1.

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 SLR-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.

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 Chapter, and analyzed them to identify the open challenges.

3.3 Taxonomy and Trends

3.3.1 Characterization of Existing Methods

Fig. 3.1 represents the top-level categorization taxonomy of the study to address SLR-RQ2 and SLR-RQ3 in this Chapter. Reliability assessment of DNNs is categorized into three main methods: Fault Injection, Analytical, and Hybrid.

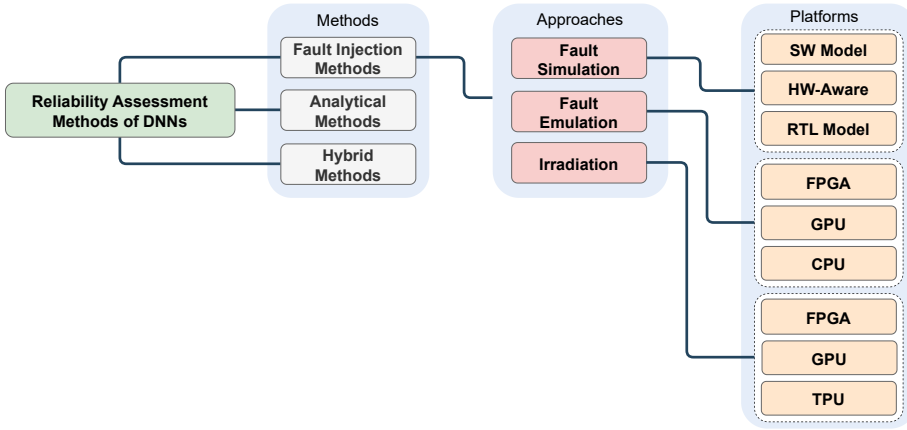


Figure 3.1: Obtained taxonomy of the reliability assessment methods for DNNs in the SLR.

Fault Injection 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 [36, 83, 201, 203]. Therefore, we adapt them for categorizing the related works on DNNs into three approaches depicted in Fig. 3.1. FI methods are categorized into three approaches of FI 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 CPU) 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.

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 employing vulnerability analysis derived by analyses, and eliminate the complexity of simulating/emulating the faults in reliability assessment.

Hybrid Methods: The integration of FI and analytical methods within hybrid methods leverages the strengths of both paradigms. Notably, the incorporation of analytical methods enables the development of mathematical models alongside FI techniques leading to faster reliability evaluation. This synergistic approach enables the derivation of reliability evaluation metrics, mitigation of the complexity associated with extensive FI experiments, and more realistic compared to analytical methods alone.

3.3.2 Research Trends

To address SLR-RQ1, we present the main statistics on the papers included in this study. Fig. 3.2 shows the distribution of the 139 included papers published over the years 2017-2022. Regarding this chart, it can be observed 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. The number of included papers in 2022 is fewer than that of in 2021 because the search for related papers is performed in Feb. 2022, therefore, many accepted papers in 2022 have not yet been indexed in databases.

Fig. 3.3 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 Fig. 3.4 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.

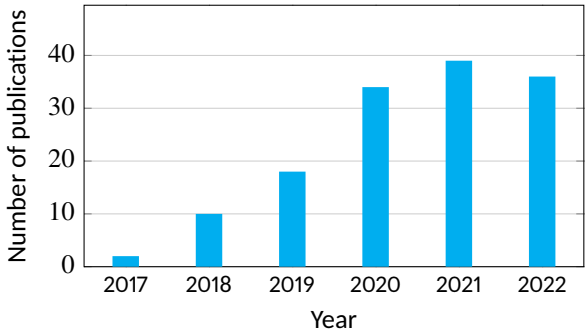


Figure 3.2: Trend of publications related to hardware reliability of DNNs over different years.

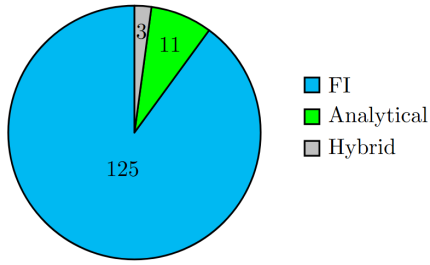


Figure 3.3: Proportion of each method in the reliability assessment of DNNs among included works.

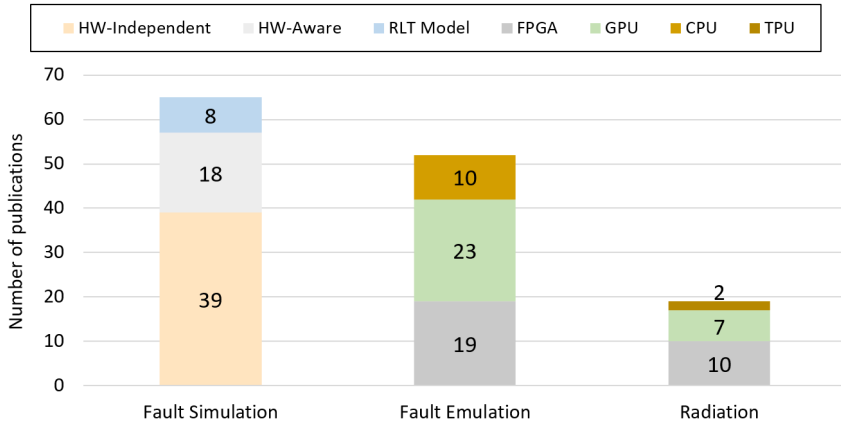


Figure 3.4: Distribution of included publications focused on FI over different approaches and platforms.

3.4 Fault Injection Methods

In FI methods, once the DNN platform and fault model are determined, perturbation and system execution are performed, and the reliability is evaluated. The identified approaches using FI for DNNs' reliability assessment and distinct evaluation metrics are presented in this subsection.

3.4.1 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.

3.4.1.1 Hardware-Independent Platform. In hardware-independent platforms, DNNs are implemented in software DNN frameworks. Therefore, fault injection is performed on top of the frameworks, i.e., PyTorch, Keras, TensorFlow, Caffe, etc. Implementing the DNN in software provides a flexible environment for studying the effect of various fault models. By analyzing the paper, it is observed that both transient and permanent faults are studied in this platform, while most of the works studied transient faults such as soft errors.

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 considered active throughout the entire execution, the value of a bit or variable (in weights, or activations) is set to 0 or 1, as experimented in [40, 113, 198]. 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 DHAs. Hence, random transient faults are injected into random bits of weights as a bitflip in different executions, as experimented in [29, 30, 43, 45, 47, 73, 88, 93, 97, 100, 125, 142, 143, 164, 175, 176, 185, 188, 204, 205, 222, 243, 248].
- 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 [113, 175], and faults in logic or datapath are investigated in [26, 27, 45, 53, 185].

To examine 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 the Hardware-Independent platform inject transient faults into the bits of randomly selected weights. Nearly all works in this class, inject faults based on Bit Error Rate (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 [40, 176, 198, 205].

***Evaluation*:** For evaluating the reliability, different metrics are considered. References [26, 27, 29, 30, 43, 88, 93, 100, 113, 125, 142, 143, 164, 175, 176, 185, 188, 204, 205, 222, 248] 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 [40, 198, 204] inject one permanent fault per experiment and classify them into three classes:

- **Masked:** No difference between the outputs of the faulty and the golden DNN.
- **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 [47, 73] 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 in [40, 198].
- **SDC:** Only affects the output results of the network.
- **Detected as a software exception:** Stops the program execution.
- **Detected by padding check action:** Corrupts the ciphertext.

Burel et al. [45] 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 previously.
- **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 [53, 97] 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 its effect on the outputs, either by changing the output results, by a threshold of accuracy loss, or by system exceptions.

***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 [160], TensorFI [55, 146, 174] and its extension TensorFI+ [138, 139], and Ares [191] 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 [27, 97], TensorFI in [53], and Ares in [205].

Moreover, to enhance the efficiency of the aforementioned tools, additional fault injectors have been introduced. One such injector, known as BinFI [54], is an extension of TensorFI that aims to identify critical bits in DNNs. Another fault injector, namely LLTFI [16], 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 check-point-based fault injector is proposed in [195] that enables studying the impact of SDCs independently of the DNN implementation framework.

3.4.1.2 Hardware-Aware Platform. This platform includes publications that consider an abstract model of the accelerator in their implementation of DNNs in software. They implement the DNN in software frameworks or high-level programming languages. Therefore, they take advantage of simulation in software fault injection while they also apply the reliability assessment to an abstract model of the accelerator.

References [31, 145] 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 [147] implement the DNN and the fault injector in software inspired by an FPGA-based DNN accelerator. Moreover, in [184, 181] 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 [126, 132] evaluate their proposed reliability improvement technique on memories in TensorFlow while injecting transient faults into the weights. PyTorch is used in [182, 183] 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 [96] 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 [42, 44] where the model of the accelerator is adopted from implementing the DNN in an N2D2 framework [1]. Furthermore, authors in [114, 247] use PyTorch and study permanent faults in MAC units of an accelerator while training to improve the reliability at inference. Authors in [231] 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 OFMaps and weights. Weight remapping strategy in memory to decrease the effect of permanent faults is evaluated in [177] using Ares. SCALE-Sim [208], a Systolic Array (SA) simulator for CNNs, is adopted in [253] to study permanent faults in PEs and computing arrays in SA-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 [145, 181, 184].

***Evaluation*:** Nearly all works in this class evaluate the DNN by accuracy loss after FI [42, 44, 96, 132, 147, 177, 181, 182, 183, 184, 215, 231, 247, 253]. References [31] and [126] evaluate the reliability by SDC rate as the proportion of faults that caused misclassification in comparison with the golden model. In addition, authors in [145] differentiate SDCs of injected transient faults into defined classes and calculate FIT for the accelerator by its components based on Eq. (2.6). In addition, in this work SDCs are classified by comparing the faulty and golden model outputs as follows:

- **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.

3.4.1.3 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 [57, 64, 152, 240, 249, 250],
- RTL implementation of DNNs [116],
- Multi-Processor System-on-Chips (MPSoCs) for DNNs [197].

In the first group, a configuration of TPU is utilized in [57, 64, 249, 250], and a model of a 2D systolic array is implemented in [152, 240]. Reference [57] also uses Eyeriss [52] 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 [64] which injects random transient faults into buffers, control and data registers. The second group, which includes [116], 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 [197] 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 [64, 116, 197]. Authors in [197] adopted the classification of [145] which was discussed in the Hardware-Aware platform. 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 [116] classify the faults similar to the general fault classification scheme (masked, SDC, crash) with different terminology.

In addition, [64] 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.

3.4.2 Fault Emulation

In this subsection, research works that assess the reliability of DNNs by emulating FI in hardware accelerators are explored, i.e., FPGA, GPU, and CPU platforms respectively.

3.4.2.1 FPGA Platform. DNNs are implemented fully or partially (e.g., one layer) on FPGAs to perform the inference phase, and faults are emulated at 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 [71, 72, 236].

In this group of works, Zynq-based architecture System-on-Chips (SoCs) [239] 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 [71, 72, 81, 218, 236],
- The on-board embedded processor initializes the faults [35, 89, 129, 130, 131, 149, 150, 157, 166, 241, 242],
- Fault injection module resides inside the hardware design implementation [34, 86, 207].

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 the 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 [218] 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 [71, 72] 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 Fig. 3.5 *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.

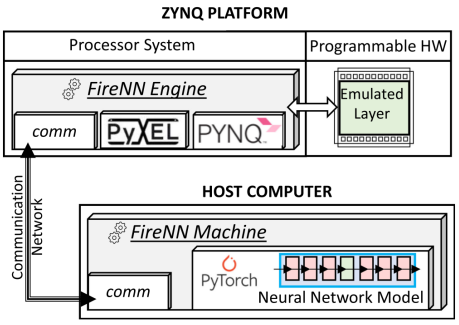


Figure 3.5: An overview of the architecture of the FireNN platform [71, 72].

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 [241, 242] 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 [35, 149, 150] provide a similar design to inject transient faults into configuration bits of the FPGA. The effects of transient faults on both on-chip memories and configuration bits of an FPGA running pruned DNNs are studied in [89]. Authors in [35] 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 [157, 166] 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 into 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 [130] and the underlying method is also used in [131, 129]. This framework is capable of injecting transient faults into both configuration bits of FPGA and on-chip memories. In this method, FINN framework [232] is used to develop and train the BNN, and the proposed framework manipulates the FINN's output to prepare it for the FI campaign. 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. Fig. 3.6 depicts in detail the steps of this FI framework.

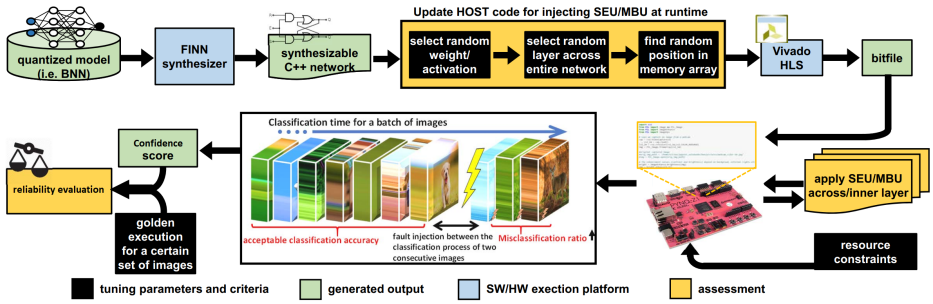


Figure 3.6: Fiji-FIN framework for fault injection into FPGAs [130].

In the third class, references [86] and [207] inject permanent faults and the work in [34] injects transient faults into the hardware implementation of the network. Authors in [86] 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 [207] injects permanent faults into the registers of the RTL model of the network. Authors in [34] explore the effect of transient faults on 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 FPGA, accuracy loss is exploited in [86, 89, 129, 130, 131, 166, 207, 236, 241, 242]. Fault classification is also performed in [71, 72, 81, 129, 131, 149, 150, 242]. References [149, 150] classify SEUs in configuration bits of the FPGA as *critical* if a fault caused misclassification compared to the golden model; otherwise, the fault is *tolerable*. In addition, *Benign Errors* are considered in [150] which

are the faults that caused the true classification of the inputs that were misclassified in the golden model. Another fault classification is presented in [71, 72] 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 [81].

Reliability is evaluated by different metrics considering accuracy loss regarding the application of the target networks in [241, 242]. 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.

[131] classifies the parameters of layers (i.e., weights and activations) by FI to characterize the level of DNN layers' vulnerability. In this work, parameters of layers are labeled as *Low-risk*, *Medium-risk*, and *High-risk*, if FI into the target layers' parameters results in less than 1%, $1\% \sim 5\%$, and more than 5% accuracy loss, respectively.

The metric Architectural Vulnerability Factor (AVF) is adopted in [149, 150] 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 [150] provide a formula to estimate the cross-section of the configuration memory in (3.1) 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 [150].

$$\sigma = AVF \times (\#UtilizedBits) \times (\frac{\sigma_{static}}{\#MemBits}) \tag{3.1}$$

In this regard, [157] estimates the SER of HyperRam saving the weights similar to (3.1) 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 [35] expressed the reliability of the neural network with *n* layers (L_1, L_2, ..., L_n) that are implemented serially as different modules on the FPGA, as an exponential distribution in (3.2), that is an extension of Eq. (2.4).

$$R_{NN}(t) = e^{-(\lambda_{L_1} + \lambda_{L_2} + ... + \lambda_{L_n})t} \tag{3.2}$$

Where $\lambda = \frac{1}{MTTF}$ (MTTF = Mean Time to Failure).

3.4.2.2 GPU Platform. This subsection explores FI in DNNs in which faults are emulated and injected into GPUs. Nearly all works on this platform have studied the effect of transient faults on GPUs. Permanent faults are studied in [63, 101, 102, 103, 104, 156]. To perform FI on GPUs, researchers adopt an FI framework on GPUs; except in [107, 156] which implemented their own FI process on CUDA and TensorRT [66], respectively. FI frameworks in GPUs including FlexGripPlus [62], NVBitFI [230], and CAROL-FI [179] are used in [49, 61, 63, 78, 90, 192], and [79], respectively. Nonetheless, an FI framework is proposed in [102] adapting and customizing NVBitFI for studying permanent faults in GPUs and is leveraged in [101, 103, 104]. Moreover, a cross-layer fault injector framework CLASSES is presented in [39] to inject SEUs at the architecture level, enabling study of the corresponding fault effects in [38]. 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 [13, 77, 79, 80, 122].

SASSIFI [108] is the most frequently used framework for FI into GPUs running DNNs that is used in [13, 14, 15, 77, 80, 121, 122, 124]. 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 bitflip model into the ISA (Instruction Set Architecture) visible states, including general-purpose registers, memory values' predicate registers and condition registers in single or multiple threads.

***Evaluation*:** Reliability evaluation of DNNs in GPUs is carried out more extensively than in other platforms. Nearly all works have classified injected faults [13, 14, 15, 61, 63, 77, 78, 79, 80, 90, 103, 104, 121, 122, 124, 156]. 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 [77, 80])

Furthermore, SDC is also categorized regarding the effect of faults on the accuracy of the DNN for the object recognition task in [13, 122]. 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.

On the other hand, the fault classification of SDCs proposed in [79] 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 [156] 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 Fig. 3.7.

Vulnerability Factors (VFs) are also adopted to analyze the reliability of DNNs on GPU platforms [13, 14, 61, 77, 79, 80, 121, 122, 124]. VF expresses 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)

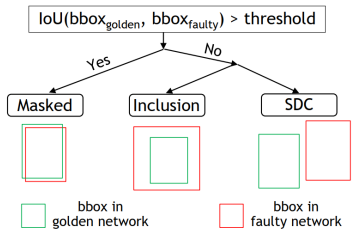


Figure 3.7: Fault classification in the object detection task based on bounding boxes [156].

[13, 121], Layer Vulnerability Factor (LVF) [13, 38, 121], Instruction Vulnerability Factor (IVF) [13, 14, 124], Program Vulnerability Factor (PVF) [13, 77, 80, 122], Operation Vulnerability Factor [90], and Architecture Vulnerability Factor (AVF) [13, 49, 61, 77, 79, 80, 122] have been presented. These metrics provide a thorough understanding of the vulnerability of each location either in DNNs or in GPUs.

3.4.2.3 CPU Platform. DNNs exploit CPUs mostly for IoT and edge applications. The research works in which faults are emulated on single and multicore processors running DNNs are reviewed in this subsection. Soft errors in the register file of ARM processors running DNNs have been studied extensively in [8, 9, 10, 11, 12, 32, 91, 153, 154, 155]. The vulnerability of instructions is studied in [154]. To emulate faults modeling soft errors in target processors, ARM-FI is developed and adopted in [153, 154, 155] and SOFIA [32] is exploited in [8, 9, 10, 11, 12, 32, 91] 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 [11, 32] 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 in Eq. (3.3), where $AVF_{critical-faults}$ is the probability of an erroneous classification due to faults.

$$MWTF = \frac{1}{execution\ \ time \times AVF_{critical-faults}} \tag{3.3}$$

MWTF is adopted as a relationship between performance and reliability in [154, 155]. AVF is obtained as the reliability metric for the register file in [10, 154, 155]. Furthermore, Program Vulnerability Factor (PVF) is leveraged to express the vulnerability of operations and instructions in [154].

3.4.3 Irradiation

The most realistic way of FI is to irradiate the devices with a beam of particles, e.g., neutrons or ions. In this subsection, the publications that study the reliability of DNN accelerators, i.e., FPGA, GPU, and TPU, under radiation are described.

3.4.3.1 FPGA Platform. Zynq SoCs have been examined under radiation tests to assess the reliability of DNNs in [34, 35, 148, 149, 158, 166, 236]. FPGAs are irradiated with neutrons in [17, 34, 35, 85, 148, 149, 236] and with protons in [163]. References [85] and [163] 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 [158, 166]. The research works set up the system's configuration before the experiment, mostly based on HW/SW co-design and save the results for further analysis. Fig. 3.8 shows an example of the setup of the FPGA irradiation.

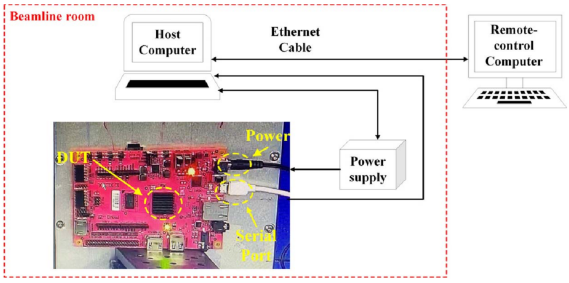


Figure 3.8: Block diagram of the setup of beam experiment in [236].

***Evaluation*:** Radiation experiments enable reliability evaluation by SER or FIT metrics [149, 158, 166, 236]. Cross-section and SER can be obtained according to Eq. (2.8) and Eq. (2.9) respectively. Most research studies on irradiation on FPGAs evaluate the reliability of devices under test using the mentioned metrics. Cross-section is exclusively employed in [17, 85]. In addition, some works classify the faults radiated into FPGA by observing the outputs [148, 149, 163]. Here, both works provide fault classification based on the output confidence scores of the neural network. [149] sets up a 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. [148, 163] 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 [35] as expressed in Eq. (3.2).

3.4.3.2 GPU Platform. Reliability of DNNs on GPUs is assessed under neutron beam radiation in [33, 77, 78, 79, 80, 107, 156]. 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., [156] tests the GPU equivalent to $2,000$ years of exposure to terrestrial neutrons, or [80] reports data that cover more than $110,000$ years of GPU operation. Fig. 3.9 illustrates the radiation test setup in [33, 77, 80].

***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 [77, 80] identify faults that caused SDC and Crash and report their FIT, separately. [78]

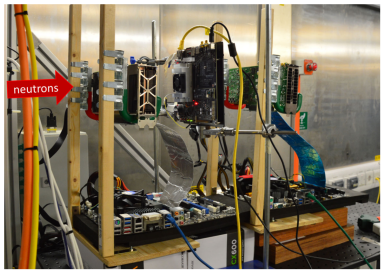


Figure 3.9: Setup of neutron irradiation to GPU [33, 77, 80].

and [79] report FIT of faults caused by SDC and DUE separately in different data representations of the DNN, and in [156] irradiated faults are classified based on Fig. 3.7. SDC rate is also the adopted evaluation metric in [107].

3.4.3.3 TPU Platform. The reliability of Google's Tensor Processing Unit (TPU) is studied under neutron beam radiation in [128, 193]. These works experimented with 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.

3.5 Analytical Methods

Analytical methods in reliability assessment mathematically represent the reliability without performing fault injection into the platform for simulation-based evaluation. 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. Generally, 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 for calculating the reliability of DNNs.

We have identified four approaches in analytical methods:

- Layerwise Relevance Propagation (LRP) based analysis [7, 199, 202, 210, 211],
- Gradient-based analysis [58, 161, 162, 206],
- Estimation-based analysis [161, 162, 187],
- ML-base analysis [92].

In the first approach, DNNs are analyzed based on an algorithm called Layerwise Relevance Propagation (LRP) that leads to obtaining critical scores for neurons/OFMaps. The second approach is based on the gradients of weights/OFMaps 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/OFMaps. 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 [210] 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, meaning it has more influence on the reliability of the network, consequently. Hence, the authors adopted the Layerwise Relevance Propagation (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 in Eq. (3.4).

$$r_{i,j} = \frac{M}{\sum_{m=0}^{M-1} R_{i,j}(y_{0,m}, t_m)} \tag{3.4}$$

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 [211] based on balancing the resilience distribution inside the DNN. Similarly, [7] 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 [202] 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 that explains how the network computes the output by the input is exploited in [206] to obtain the sensitivity of layers and the importance of weights. This work defines the sensitivity of layers in compliance with the difference between 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 is analyzed in [58], which refers to the amount of accuracy drop with bitflip 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 (3.5).

$$Sensitivity_w = gradient_w \times \varepsilon_w \tag{3.5}$$

Sensitivity analysis in this work leads to the allocation of robust hardware to the more sensitive weights.

[161, 162] have presented three gradient-based approaches for vulnerability estimation of OFMaps in a DNN. *Gradient* approach considers the absolute values of OFMaps' gradients with respect to the cross-entropy loss at the output in a backpropagation as the vulnerability of OFMaps. *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 the 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 OFMaps in a DNN. Authors in [161, 162] also presented three estimation-based approaches for the vulnerability of OFMaps. They estimate the relative OFMaps' vulnerability by calculating the *max neuron value*, *OFMap range*, and *average L2* over the input samples. They have provided approximate yet scalable and fast approaches to estimate the vulnerability of OFMaps.

[187] 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 (3.6) provides a representation of this estimation.

$$SERN = \frac{Crit\_OPs_i + \sum_{i+1}^{n} OPs}{\sum_{i}^{n} OPs} \tag{3.6}$$

An ML-based approach for analytical reliability analysis is presented in [92] 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 efficacy is 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 compared to FI. It is shown that analytical methods can evaluate/estimate the vulnerability of different components of DNNs including neurons, OFMaps, and weights. Analytical methods are significantly faster and less complex than FI and are accelerator-agnostic, and 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.

3.6 Hybrid Methods

In hybrid methods, both FI and analytical methods are carried out to assess the reliability of DNNs. To that end, [111] proposes a reliability assessment framework called Fidelity based on a hybrid method. This framework studies the transient faults in both data and control paths 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 [65], i.e., an open-source NVIDIA's DNN accelerator. To further improve this method, a software model for NVDLA is proposed in [234] to enable the 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. [251] 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 that when it reaches a high accuracy, it can be used for vulnerability estimation of the entire set of parameters.

3.7 Discussion: Qualitative Comparison and Open Challenges

In this Section, we highlight the pros and cons of the categorized methods in the SLR. Thereafter, we present a qualitative comparison of different reliability assessment methods for DNNs. Lastly, we discuss the open challenges as well as major potential research directions for the future.

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 which 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 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. Moreover, FI is not scalable and with the increasing size of emerging DNNs in the number of parameters and computations, utilizing FI is a major challenge for reliability assessment. The advantages and disadvantages of FI methods are investigated in detail, as follows:

- In fault simulation approaches, the design time is low for high-level software implementations and they are adaptable for various DNNs, DHA models, and fault models, providing various reliability evaluation metrics. Software simulation enables the reliability study of variations of DNNs under approximation, quantization, encryption, etc. There are multiple open-source frameworks available for high-level software simulation without the need for any special facilities and it is possible to run on regular PCs. On the other hand, fault simulation possesses a high time complexity to achieve a sufficient confidence level and it is non-scalable for emerging DNNs. Moreover, fault simulation is not a realistic model of fault effects in high-level software implementations and its results are inaccurate at high-level software implementations.

- Fault emulation provides realistic reliability analysis for DHAs by conducting experiments with real conditions of DHA operation, accessing all possible locations of the DHA for FI, and deriving several evaluation metrics and fault classifications. However, its design and development process is time-consuming and it requires the physical DHA. Also, different platforms need their specific design and development to perform FI and they need platform-specific frameworks for FI, making fault emulation methods even more challenging. It is worth mentioning that fault emulation

has a lower observability of fault propagation compared to fault simulation.

- Fault radiation is the most realistic approach as real physical faults are injected into the chip, and is suitable for developing fault models. It enables the study to validate simulation and emulation approaches and allows observing the real behavior of the DHA when faced with a physical effect. However, performing radiation testing requires specialized facilities, expertise, and significant expenses. During a radiation experiment, there is a low control over the accuracy of fault injection in terms of the number and location of occurred faults, and also a lack of observability of fault propagation.

Analytical methods have been proposed as an alternative to FI to cope with its high complexity. 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. However, current analytical methods do not consider the accelerator models, and there is a gap in the use of reliability evaluation metrics and their resilience estimation accuracy. 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, quantitative, and realistic evaluation of reliability. 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 3.1 presents a qualitative comparison between the categorized methods of reliability assessment for DNNs regarding the reviewed papers included in this Chapter.

The analysis of statistics presented in Fig. 3.3 highlights that the majority of the identified papers 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 Fig. 3.2. 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 [50], 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;

Table 3.1: Qualitative analysis comparing different reliability assessment methods for DNNs.

	Fault injection	Analytical	Hybrid
Time Complexity	High	Low to Moderate	Moderate
DHA-aware	Yes	No	Yes
Leading to fault-tolerant design	Yes	Yes	Yes
Fault models variety	All fault models	Few fault models	Few fault models
Fault propagation observability	Low (radiation) Moderate (emulation) High (simulation)	High	High
Implementation system level	Software & 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

- 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. 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 to 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 have 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, especially for Large Language Models (LLMs) and Vision Transformers (ViTs).

3.8 Chapter Conclusions

In this Chapter, a comprehensive and systematic literature review is carried out on 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, addressing RQ1.1 and RQ1.2 in Chapter 1. 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, addressing RQ1.3 in Chapter 1. It is outlined that FI is non-scalable for emerging DNNs, therefore analytical and hybrid methods are proposed to tackle this challenge. Based on the analysis of this Chapter, 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.

4 Reliability Assessment for CNNs

To ensure a reliable DNN deployment, the first step is to extensively evaluate the functionality of pre-trained CNNs against hardware faults. Reliability assessment is the process of representing the target DNN accelerator or model and measuring its reliability with respect to the corresponding quantitative evaluation metrics. Chapter 3 presented and discussed the existing methods to assess the reliability of DNNs. As mentioned, 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. However, given the growing size of emerging DNNs and their accelerators [48, 169], obtaining a fully precise evaluation using FI approaches, is unfeasible and impractical.

Throughout the literature, multiple research works are presented to significantly reduce the simulation complexity of FI experiments for DNNs while maintaining statistical accuracy, especially for software simulation [54, 160, 200, 238]. Nonetheless, any FI experiment requires a certain level of statistical confidence, which is obtained by a considerably high number of repetitions and a multitude of fault locations [200]. In addition, most faults in an FI experiment on DNNs are masked [40] and are thus unnecessarily examined. Therefore, FI-based approaches for reliability assessment are inherently non-scalable and can take days to weeks depending on the DNN complexity.

On the other hand, analytical approaches are presented to tackle the drawbacks of FI, particularly its scalability issue. Nonetheless, they cannot sufficiently explain how a fault propagates through the network and influences its outputs. In addition, they do not provide reliability-oriented metrics, including Vulnerability Factors (VFs), which is an essential metric for reliability evaluation. Moreover, the accuracy of analytical methods is not comparable to that of FI-based methods.

To the best of our knowledge, there is no accelerator-agnostic resilience analysis method for DNNs that can compete with FI in terms of scalability and accuracy of reliability evaluation while providing fine-grain metrics enabling different reliability improvement techniques. In this Chapter, we introduce the DeepVigor method, its extensions and applications that address the corresponding gaps in the literature. This Chapter attempts to address P2 which includes RQ2.1-2.4 and presents contributions mentioned in C2, in Chapter 1, based on the following publications:

I M. H. Ahmadilivani, M. Taheri, J. Raik, M. Daneshtalab, and M. Jenihhin. DeepVigor: VulnerabIlity Value RanGes and FactORs for DNNs' Reliability Assessment. In *IEEE European Test Symposium (ETS)*, pages 1–6. Venice, Italy, 2023

II M. H. Ahmadilivani, J. Raik, M. Daneshtalab, and M. Jenihhin. Deepvigor+: A Scalable, Accurate and Automated Framework for Resilience Analysis of Deep Neural Networks. *Under review*, pages 1–14, 2024

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

IV 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, J. E. Rodriguez Condia, M. Sonza Reorda, M. Taheri, and M. Traiola. Special Session: Reliability Assessment Recipes for DNN Accelerators. In *IEEE 42nd VLSI Test Symposium (VTS)*, pages 1–11. Tempe, United States of America, 2024

In the rest of this Chapter, Section 4.1 presents the DeepVigor method in which a novel paradigm of resilience analysis for DNNs is introduced. Section 4.2 presents DeepVigor+, which tackles the scalability issue of DNNs' resilience analysis. In Section 4.3 DeepVigor is extended for QNNs leading to their fault tolerance and fast fault resilience assessment, and Section 4.4 concludes the Chapter.

4.1 DeepVigor: VulnerabIlity Value RanGes and FactORs

This section introduces DeepVigor, an innovative, metric-oriented, and accelerator-agnostic resilience analysis method for DNNs that derives fine-grain VF as reliability quantification metrics for layers, neurons, and bits.

4.1.1 Fault Model

In this section, the fault propagation analysis is performed at the outputs of DNN neurons. This fault propagation model covers 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 [54].

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). It has been shown that high fault coverage obtained using the single-bit model results in a high fault coverage of multiple-bit faults [46]. 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.

4.1.2 Fault Propagation Analysis

Fig. 4.1 depicts an overview of the rationale behind the DeepVigor method. The figure illustrates a neural network with 2 hidden layers, 6 neurons, inputs, golden (fault-free) activation values (inside of neurons), and weights (on the arrows). The golden classification output is $class_1$. A fault changes the neuron's output by δ, which is the difference between the golden and faulty activation values. The δ which can have either a negative or a positive value is propagated to the output layer, and may change the classification result. The fault propagation differs in each output class as Δ_1 and Δ_2. Misclassification happens when the value of the output $class_2$ gets higher than that of the $class_1$.

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. 4.1. 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 method.

4.1.3 The DeepVigor Method

The steps of the proposed DNNs' resilience analysis method (DeepVigor) and its validation are illustrated in Fig. 4.2. As shown, an analysis is performed on a set of data (i.e., set_1, 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., set_2, test set) to validate the outcomes of the analysis. The steps of DeepVigor are as follows:

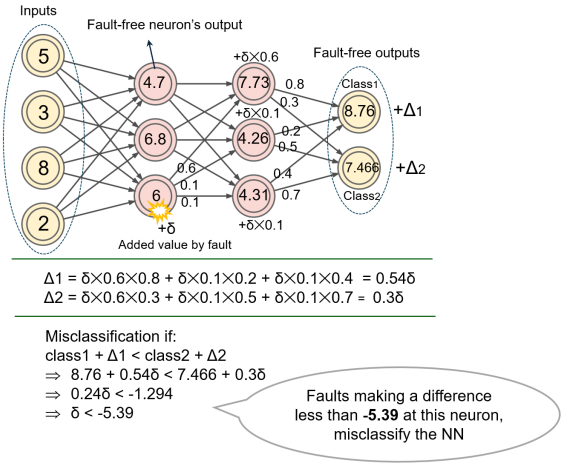


Figure 4.1: An example of fault propagation analysis model and finding the vulnerability value ranges for a neuron with a given input.

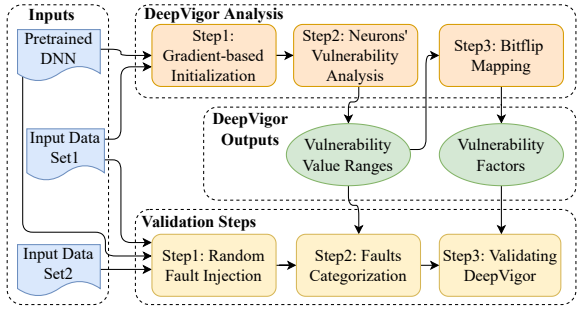


Figure 4.2: Steps of the DeepVigor method for DNNs' reliability assessment and its validation.

Step 1 - 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 $\mathscr{L}$ is defined in Eq. (4.1) as:

$$\mathscr{L} = sigmoid(\sum_{j=0}^{N}(\mathscr{E}_{c_t} - \mathscr{E}_{c_i})) \tag{4.1}$$

where c_t is the golden top class and $\mathscr{E}_{c_t}$ and $\mathscr{E}_{c_i}$ 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.

$\mathscr{L}$ represents the impact of the neuron's erroneous output on the golden top class of the network. When the gradient of $\mathscr{L}$ 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 $0 < \delta_k^l < \varepsilon_k^l$ 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.

Step 2 - 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:

$$\begin{cases} r_{upper} = min(\delta_k^l), \delta_k^l > 0, \mathscr{E}_{c_t} < \mathscr{E}_{c_i}, i \neq t \\ r_{lower} = max(\delta_k^l), \delta_k^l < 0, \mathscr{E}_{c_t} < \mathscr{E}_{c_i}, i \neq t \end{cases} \tag{4.2}$$

where c_t and c_i are the golden top class and any other output class, respectively, and $\mathscr{E}_{c_t}$ and $\mathscr{E}_{c_i}$ are the erroneous output values corresponding to the respective classes.

Eq. (4.2) 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, Eq. (4.2) 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:

$$\begin{cases} R_{NV}(l,k) = \bigcap_{j=1}^{T} R_{NV}(l,k,x_j) \\ R_{VV}(l,k) = \bigcap_{j=1}^{T} R_{VV}(l,k,x_j) \end{cases} \tag{4.3}$$

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. 4.3 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. 4.3);
- **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. 4.3);
- **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. 4.3). Cases *d-f* in Fig. 4.3 happen when the portion of zero gradients in *step1* is less than the *threshold* and more than $1 - threshold$.

Step 3 - 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

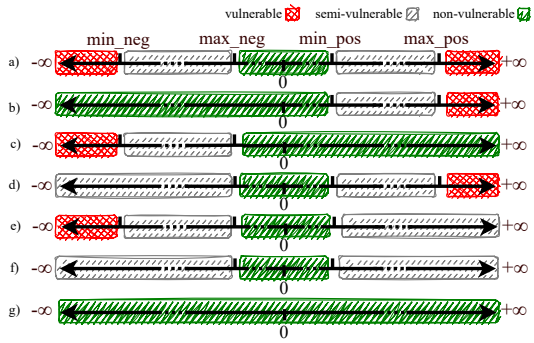


Figure 4.3: Different possible cases of vulnerability ranges for each class in a neuron.

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 (4.4), (4.5), and (4.6), respectively. Vulnerability factors express the probability of misclassifying the network in case of the occurrence of a bitflip at the target element.

$$LVF = \frac{\#vulnerable\ bits\ in\ layer}{\#inputs\ \times \#layer's\ neurons \times word\ length} \times 100 \tag{4.4}$$

$$NVF = \frac{\#vulnerable\ bits\ in\ neuron}{\#inputs\ \times word\ length} \times 100 \tag{4.5}$$

$$BVF = \frac{\#vulnerable\ times\ for\ bit}{\#inputs} \times 100 \tag{4.6}$$

4.1.4 Validating DeepVigor By Fault Injection

As illustrated in Fig. 4.2, 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:

Step 1 - 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 [144]. The required number of faults is obtained by Eq. (4.7) where $N = word\ length \times \#layer's\ neurons$ that represents the total number of bits in the output of a layer.

$$\#layer's\ random\ faults = \frac{N}{1 + (0.01^2 \times \frac{N-1}{1.96^2 \times 0.5^2})} \tag{4.7}$$

Step 2 - 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.

Step 3 - 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.

4.1.5 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 (4.4)-(4.7) is 32 bits. For validation, a layer-wise statistical random FI is performed that satisfies a 95% confidence level and 1% error margin, according to Eq. (4.7).

In the first step of DeepVigor, ε_k^l is considered $\pm 10,000$ 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 set_1) and test (as the input data set_2) data.

4.1.6 Results and Validation

We analyze all neurons of the representative DNNs with training sets as the input data set_1 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 from 66.63% to 99.42% of faults as non-critical over different layers of analyzed networks.

Table 4.1: 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%

For validation, Table 4.1 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 4.1.3.

We have also experimented with FI on the test sets (input data set_2) to see the validity of the analysis on different sets reported in Table 4.2. As it can be seen, similar high accuracy values to input data set_1 are obtained.

Table 4.2: 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. 4.4 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. 4.5 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.

4.1.7 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 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. 4.3) for one neuron can be obtained by 9.1 times feed-forward execution per neuron on average.

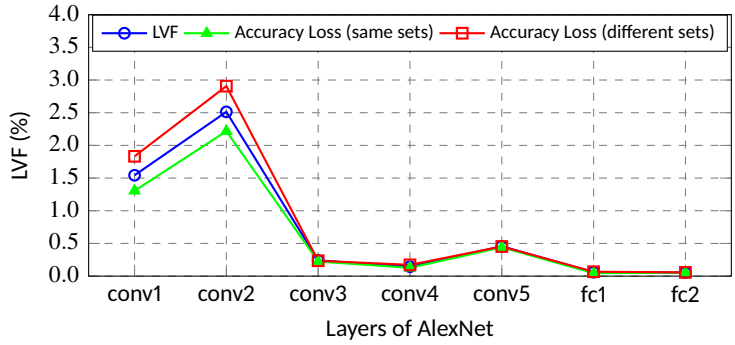


Figure 4.4: Correlation between LVF and accuracy loss.

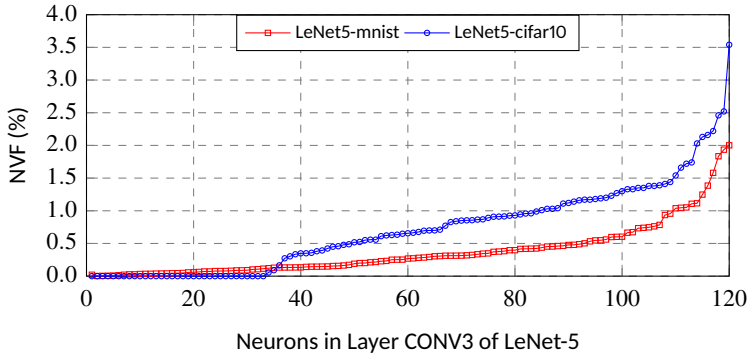


Figure 4.5: NVF of neurons in CONV3 for LeNet5-mnist and LeNet-cifar10.

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:

- Two backpropagation execution per neuron (one for positive and one for negative numbers) in *step1*;
- Feed-forward executions based on the number of searches for finding a positive or negative δ_k^l per neuron, in which the best case is 0 search (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. 4.1 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.

4.1.8 Discussion

The 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. 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 VF metrics for bits, neurons, and layers;
- Fault-aware mapping for neurons on the processing elements of accelerators as in [202, 210];
- Applying range restriction for neurons' or layers' outputs for preventing faults propagation as in [53, 93, 113].

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. This method enables an accelerator-agnostic analysis for DNNs and results can be applied to different accelerators.

4.2 DeepVigor+: Scalable and Accurate Fault Resilience Analysis

DeepVigor is demonstrated to be faster than FI due to its novel fault propagation analysis method, however, it requires a high execution time to obtain VFs. The reason is that DeepVigor attempts to analyze all neurons in a CNN as well as to find the vulnerability values in a huge space of numbers. This section proposes DeepVigor+, a new method to quickly obtain VF metrics for layers and DNN models addressing both scalability and accuracy of fault resilience analysis in the literature. DeepVigor+ employs an optimized error propagation analysis in neurons, assuming that a single fault might happen either at its inputs or weights, thus leading to effectively shrinking the search space for vulnerability values. Moreover, DeepVigor+ proposes a stratified sampling to further accelerate the process of resilience analysis without a tangible analysis accuracy mitigation, leading to obtaining VF in a few minutes, even for deep and large emerging DNNs.

4.2.1 Fault Model

A Single Event Upset (SEU) caused by soft errors can impact memory cells, leading to unintended bit flips in stored values. In DNN accelerators, SEUs can affect either the model's parameters (i.e., weights) or its activations (i.e., layer inputs), as these values are stored in memory elements such as registers or on-chip memory [123].

For reliability analysis, the multi-bit fault model is more realistic, however, it requires a huge fault space to consider all combinations, i.e., 3^{n-1} combinations where n is the number of bits. On the other hand, it is demonstrated that the single-bit fault model provides a high fault coverage of multiple-bit faults [46]. Therefore, analyzing the reliability of DNNs based on a single-bit fault model is valid for obtaining the VF of models.

Therefore, in DeepVigor+, we build our analysis based on the single-bit fault model in weights and activations. We assume that a single parameter or a single input to a layer is faulty which is propagated to the output of the corresponding layer. By mathematical

analysis, we model the fault propagation to the output of neurons and DNNs, resulting in calculating VF for them.

4.2.2 Fault Propagation Model

4.2.2.1 Single Fault Analysis in 32-bit Floating-Point To model the behavior of faults for the analysis, we assume that at an inference, a single fault may influence the value of a single input activation to a neuron or a single parameter in a CNN. First, we analyze the effect of a single bitflip on the 32-bit floating-point data representation to comprehend how a value might change due to a single fault. IEEE-754 32-bit floating-point data type is shown in Fig. 4.6. It contains 1 sign bit, 8 exponent bits, and 23 fraction bits, and represents a number based on Eq. (4.8).

$$number = (-1)^{sign} \times 2^{E-127} \times (1 + \sum_{i=0}^{i=23} b_{23-i} \times 2^{i}) \tag{4.8}$$

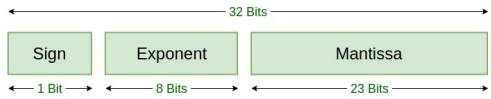


Figure 4.6: 32-bit floating point IEEE-754 data representation.

In this regard, we consider the following lemmas:

- *Lemma 1:* Any bitflip from 1 to 0 in a value decreases the value as ε while $\varepsilon < 0$, and any bitflip from 0 to 1 increases it as ε while $\varepsilon > 0$.

- *Lemma 2:* In the 32-bit floating-point data representation, an error (ε) induced to a value (x) by a bitflip in bit i can be represented as in Eq. (4.9), whether the bitflip is in sign bit ($2 \times x$), exponent bits ($2^i \times x$), or mantissa bits (2^i), as stated in [82, 248].

$$\begin{cases} x_{faulty} = x + \varepsilon \\ \varepsilon \in \{2 \times x, 2^{i-23} \times x, 2^i\} \end{cases} \tag{4.9}$$

- *Lemma 3:* To unify the analysis of the error induced to a value by a single bitflip, we can approximate ε by representing it as a power of two, for each section of the data representation, as follows:

  1. Sign: approximate ε as $\pm 2^{log_2(2 \times x)}$,
  2. Exponent: approximate ε as $\pm 2^{i-23}$, assuming that x is a small value,
  3. Mantissa: approximate ε as $\pm 2^i$.

  In all cases, ε can be approximated as the nearest value of the power of two to the actual ε. This approximation can lead to a unified representation for ε. To that end, ε might be negative or positive (based on Lemma 1) and might be small or big (based on Lemma 2). Eventually, we can express it as a unified representation as shown in Eq. (4.10).

$$\varepsilon \approx \pm 2^{\rho}; \rho \in \{\pm 1, \pm 2, \pm 3, ...\} \tag{4.10}$$

- *Lemma 4:* When a faulty value is used in a multiplication operation in CNNs, the erroneous output can also be approximated. It has been observed that the parameters in CNNs are mainly distributed around 0 and in the range of $[-1,1]$ [118]. In order to approximate the error of multiplying two values when one of them is faulty, we can analyze it based on Eq. (4.11), where x and y are fault-free values and x' is the faulty value after a bitflip in x.

$$\begin{aligned} let: \;\; & x' = x + \varepsilon \\ then: \;\; & x' \times y = x \times y + \varepsilon \times y \\ & \Rightarrow x' \times y = x \times y + \delta \end{aligned} \tag{4.11}$$

  In Eq. (4.11), when x is a small value, $\delta \approx 0$. Since most values in CNNs' operations are close to 0, the erroneous values in multiplications in a CNN can be approximated based on the unified error representation, as shown in Lemma 3 and Eq. (4.10).

4.2.2.2 Single Fault Error Propagation in CNNs We analyze the single bitflip error propagation in CNNs considering their effect on the values of numbers. Each neuron in a convolutional (CONV) layer operates as shown in Eq. (4.12), where the Output Feature Map (OFMap) in lth layer and kth channel is obtained by the summation of multiplications between weights ($\hat{w}$) and Input Feature Map (IFMap, $\hat{x}$) plus bias (b). In CONV layers, $\hat{w}$ and $\hat{x}$ are 3-Dimensional (3D) $c_{in} \times n \times n$ arrays, where c_{in} is the number of inputs channels to the layer, and n is the kernel size.

$$OFMap_k^l(\hat{x}, \hat{w}, b) = (\sum_{i=0}^{c_{in}} \sum_{j=0}^{n} \sum_{k=0}^{n} x_{ijk} \times w_{ijk}) + b \tag{4.12}$$

Here, we assume that a fault affects a single IFMap in a single neuron, thus, it produces a single erroneous OFMap. Supposing that a fault occurs in an IFMap x_{ijk}, represented as x'_{ijk}. The fault introduces an error ε to the fault-free value of x_{ijk}. Therefore, it can be expressed in Eq. (4.13).

$$x'_{ijk} = x_{ijk} + \varepsilon \tag{4.13}$$

Hence, once a bitflip occurs in $\hat{x}$ in Eq. (4.12), the partial multiplication is computed in Eq. (4.14). In this equation, the term $x_{ijk} \times w_{ijk}$ represents the fault-free multiplication, and $\varepsilon \times w_{ijk}$ is an added error to the output by a fault which can be represented as δ.

$$\begin{aligned} x'_{ijk} \times w_{ijk} &= (x_{ijk} + \varepsilon) \times w_{ijk} \\ &= x_{ijk} \times w_{ijk} + \varepsilon \times w_{ijk} \\ &= x_{ijk} \times w_{ijk} + \delta \end{aligned} \tag{4.14}$$

Consequently, the erroneous OFMap can be expressed in a way that it is the summation of the fault-free OFMap and δ, while the single faulty IFMap in $\hat{x}$ can be in any index of the corresponding 3D array. Considering Lemma 4, the induced error at the neuron's OFMap can be expressed in Eq. (4.15).

$$\begin{aligned} OFMap_k^l(\hat{x}', \hat{w}, b) &= OFMap_k^l(\hat{x}, \hat{w}, b) + \delta \\ \delta &\approx \pm 2^{\rho}; \rho \in \{\pm 1, \pm 2, \pm 3, ...\} \end{aligned} \tag{4.15}$$

On the other hand, when a fault occurs in a weight, it has the same effect on a single neuron's output. However, the same faulty weight in the corresponding CONV layer is used by all neurons in the layer, as filters slide over all IFMaps. Therefore, it results in an output channel in which all OFMaps are erroneous, as shown in Fig. 4.7. Considering Lemma 4, the fault propagation for an output channel can be expressed in Eq. (4.16).

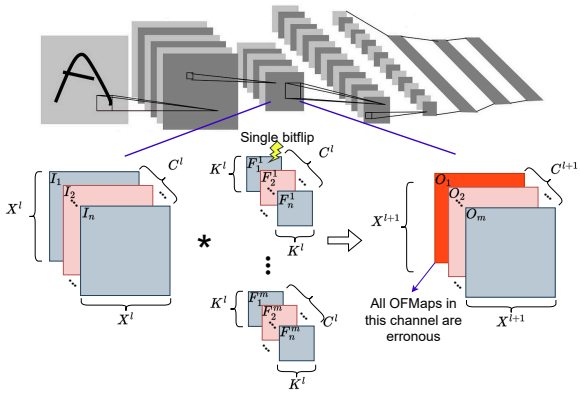


Figure 4.7: Fault propagation in a CNN in the case of a single bitflip in a weight.

$$
\begin{aligned}
Out\_Channel_k^l(\hat{x},\hat{w},b) &= \hat{x} * \hat{w} + b \\
Out\_Channel_k^l(\hat{x},\hat{w}',b) &= \hat{x} * \hat{w} + b + \varepsilon * \hat{x} \\
Out\_Channel_k^l(\hat{x},\hat{w}',b) &= Out\_Channel_k^l(\hat{x},\hat{w},b) + \delta \\
\delta \approx \pm 2^{\rho}; \rho &\in \{\pm 1, \pm 2, \pm 3, \ldots\}
\end{aligned}
\tag{4.16}
$$

Based on the aforementioned theoretical analysis, to analyze the effect of single faults on CNNs, regardless of the fault occurrence in activations or weights, we only need to identify an added value δ (as a power of two) at the output of the target neuron/channel where it misclassifies the golden class of the CNN for each input image.

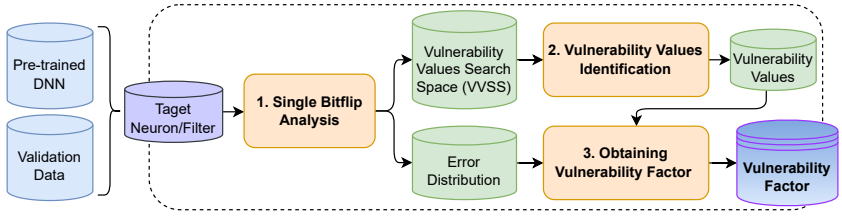


Figure 4.8: An overview of the conducted steps in DeepVigor+.

4.2.3 The DeepVigor+ Method

In this section, the steps of DeepVigor+ are presented. Fig. 4.8 illustrates the steps of the DeepVigor+ methodology for fault resilience analysis of DNNs against single faults. The method's objective is to provide the Vulnerability Factor (VF) for layers and the entire DNN model when a single fault happens in activations or weights.

The inputs of the method are a pre-trained CNN and validation data in a dataset. During the analysis, an OFMap or output channel is targeted, and multiple steps are taken to provide the VF metrics:

1. Single Bitflip Analysis: constructs a search space for possible vulnerability values and produces a distribution of erroneous values based on the approximations in Eq. (4.15) and (4.16),

2. Vulnerability Values Identification: obtains vulnerability values for the target OFMap or channel

3. Obtaining Vulnerability Factor: exploits identified vulnerability values and error distribution to provide VF for the target OFMap or channel.

By obtaining the VF for the analyzed neurons and filters in a DNN, the VF for layers and the entire DNN can be derived. The details of each step are explained in the following. Noteworthy that each step presents a detailed description for analyzing a target neuron and then briefly links it to weights in filters.

Step 1 - Single Bitflip Analysis: As mentioned above, single faults at the inputs of a neuron are considered. This step conducts bitflips in the input activations of the target neuron to construct Vulnerability Values Search Space (VVSS) and Error Distribution Map (EDM) for the target neuron. VVSS is a set of candidate values representing all possible δ in Eq. (4.15). In other words, VVSS represents output errors produced by single bitflips in any bit locations of the inputs of the target neuron. EDM represents the distribution of δ in Eq. (4.15) throughout the approximated δ values with a power of two.

Obtaining VVSS and EDM requires performing a single bitflip for each input and multiplying it by its corresponding weight, as shown in Eq. (4.14). Nonetheless, it is an exhaustive operation with high time complexity. This complexity can be remarkably reduced by leveraging Algorithm 4.1. In this algorithm, for each bitflip, we obtain all possible δ produced at the output of the neuron for all inputs at once.

Since each input might be faulty separately, to produce all errors in a neural operation, we first flip the ith bit in all inputs (line 3) and then convert the binary representation to a value (line 4). Then, the difference of the *erroneous_inputs* with the fault-free input is computed (line 5), which represents all possible errors at inputs (ε in Eq. (4.14)) for all inputs added by a single bitflip in either of them. Then, a point-wise multiplication between the *erroneous_inputs* and *weights* results in producing all possible errors (δ in Eq. (4.15)) at the output (line 6).

Algorithm 4.1 Error Analysis for a Neuron

Input: Target neuron's inputs and weights as 3D matrices;
Output: All possible errors at the output;
Assume: δ is the error added to each golden output; All values are in 32-bit floating-bit;

```
1: binary_representation = float32_to_binary(inputs);
2: for i ∈ [0,31] do:
3:     flip bit i in binary_representation;
4:     erroneous_inputs = binary_to_float32(binary_representation);
5:     input_errors = erroneous_inputs - inputs;
6:     output_errors_list.append(input_errors ⊙ weights);
7: end for;
```

Algorithm 4.1 produces all errors at the output of the target neuron. Thereafter, based on the presented theory in the previous subsection, we generate the distribution of all produced errors as an Error Distribution Map (EDM) based on the Candidate Vulnerability Values (CVVs) which are a set of numbers as a power of two, shown in Eq. (4.17). Based on the experimental observations, we limit the CVV in the analysis between -2^{10} and 2^{10} for large values. The error values between -2^{-10} and 2^{-10} are merged as their propagation effects on CNNs are negligible and masked.

$$CVV = \{\pm 2^{\rho}\}; \rho \in \{0, \pm 1, \pm 2, \pm 3, \cdots, \pm 9, \pm 10\} \tag{4.17}$$

EDM contains the *distribution ratio* of the values in *output_errors_list* in Algorithm 4.1, between each consecutive CVV, as shown in Eq. (4.18). EDM provides the ratio of error distribution with respect to each candidate vulnerability value, demonstrating how much each CVV represents the produced errors at the output of the target neuron. Based on the EDM, each CVV whose *distribution ratio* is not zero, is added to the Vulnerability Value Search Space (VVSS). This means that VVSS contains a subset of CVV that are the approximated errors produced by single faults in the inputs and might lead to misclassification at the output of the CNN.

$$\forall i \in CVV; distribution\_ratio_i = \frac{count(CVV_{i-1} < output\_errors\_list < CVV_i)}{count(output\_errors\_list)} \tag{4.18}$$

The process of single bitflip analysis for a target filter is similar to the one for a neuron. Nonetheless, in the case of a bitflip in a filter, all OFMaps in an output channel are affected (as described in Eq. (4.16)). Therefore, this step is performed separately on every bit of all weights in the target filter, so that *VVSS* and *EDM* for the corresponding filter are obtained.

Step 2: Vulnerability Values Identification: This step exploits VVSS to identify the vulnerability values for the target neuron (i.e., OFMap) by exploring its constructed VVSS. As mentioned, VVSS represents output errors induced by a single fault occurring at the inputs of the target neuron. The objective is to identify the *maximum negative* and *minimum positive* vulnerability values among the existing ones in VVSS for a neuron that misclassify the DNN's outputs from its golden classification for each input data.

To explore the vulnerability values efficiently, we divide them into four different exploration spaces:

1. $VVSS_{[-\infty,-1]}$: contains CVVs between $[-\infty, 1]$ with non-zero distribution ratio,

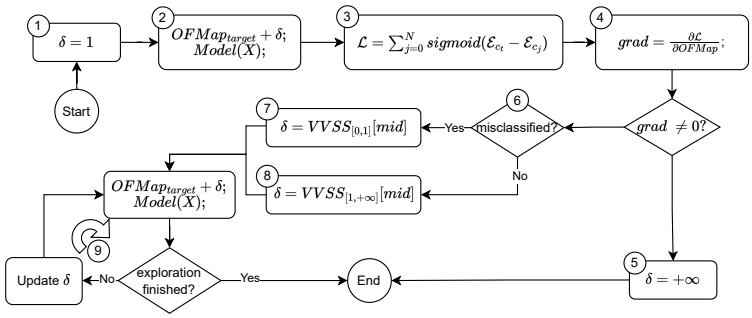


Figure 4.9: Vulnerability value identification for a target neuron with a single input data for positive errors.

2. $VVSS_{(-1,0)}$: contains CVVs between $(-1, 0)$ with non-zero distribution ratio,

3. $VVSS_{(0,1)}$: contains CVVs between $(0, 1)$ with non-zero distribution ratio,

4. $VVSS_{[1,+\infty]}$: contains CVVs between $[1, +\infty]$ with non-zero distribution ratio.

The flowchart in Fig 4.9 illustrates the algorithm of vulnerability value identification for *positive vulnerability values* for a single input data $\mathbf{X}$. In this flowchart, δ is the vulnerability value added to the target OFMap. First, δ equals 1 (*box 1*) is added to the target OFMap. Thereafter, the forward pass of the CNN is performed to obtain its output logits ($\mathscr{E}$) while a neuron is erroneous (*box 2*).

In *box 3*, a loss function $\mathscr{L}$ is calculated to obtain the effect of added δ to the target OFMap on the output classes based on the summation of the differences between all output class's logits from the golden class. In this loss function, $\mathscr{E}_{c_t}$ represents the erroneous output logit of the golden top class and $\mathscr{E}_{c_j}$ is the erroneous output logit of any other class.

In *box 4*, the gradient of loss function $\mathscr{L}$ w.r.t. the target OFMap is calculated to check if the CNN might be misclassified or not, when the target OFMap is erroneous. If the gradient is 0, faults producing positive deviation in the neuron do not lead to misclassification. In this case, the neuron's vulnerability value is considered the biggest CVV assumed value for positive numbers (i.e., 2^{10}) in *box 5*, and the algorithm ends. If the gradient is not 0, therefore, a value can be found for δ in the target OFMap which misclassifies the CNN. The rest of the algorithm attempts to find the minimum positive δ in VVSS.

In *box 6*, it is examined if the CNN misclassifies the input from its golden classification when $\delta = 1$ for the target neuron, determining the initialization for δ in the next steps. If the input is misclassified when $\delta = 1$, it means that its vulnerability value is less than 1 and we should explore $VVSS_{(0,1)}$ (*box 7*), otherwise, $VVSS_{[1,+\infty)}$ should be explored (*box 8*). In the case of exploring $VVSS_{(0,1)}$, δ is set to the middle element of the set, and if it does not misclassify the golden output, the next bigger CVV should be explored. This process continues until the first value which misclassifies the output is found (*loop 9*). The final positive vulnerability value (δ^+) for the target neuron is the minimum value in positive VVSS that does not misclassify the input for CNN from its golden classification.

To identify the maximum negative vulnerability value (δ^-) that does not misclassify the input from its golden classification, the same procedure is conducted to explore negative values in VVSS. As a result, the Vulnerability Value Range (VVR) for the corresponding neuron is obtained and is expressed in Eq. (4.19). VVR represents all induced error values to the outputs of a neuron leading to a misclassification. Noteworthy that since bitflips in 32-bit floating-point might lead to large values, we assumed that any value larger than 2^{10} and less than 2^{-10} are critical for CNNs, therefore, these values represent $+\infty$ and $-\infty$ respectively.

$$VVR = (-\infty, \delta^-] \cup [\delta^+, +\infty) \tag{4.19}$$

The step of vulnerability values identification for a target filter is similar to the one for a neuron. It is conducted similarly to the illustrated flowchart in Fig. 4.9 to obtain VVR for the corresponding output channel resulting in VVR for each target channel in a DNN.

Step 3: Obtaining Vulnerability Factors: As mentioned, an Error Distribution Map (EDM) is obtained for the target neuron in *step 1*, representing the distribution ratio for each vulnerability value. On the other hand, the Vulnerability Value Range (VVR) is obtained in *step 2*. This step aims to map VVR to EDM to provide the VF for the target neuron, expressing the probability of misclassification in the case of a single bitflip in its inputs.

Fig. 4.10 depicts how VF can be obtained by mapping VVR to aggregated EDM. Aggregated EDM represents the summation of the *distribution ratio* for all values less than a

CVV. As shown in Fig. 4.10, all values between (δ^-, δ^+) are non-vulnerable meaning that if an error deviates the OFMap as big as any value in this range, it will not misclassify the golden output. Otherwise, the error value is vulnerable leading to a misclassification.

VF is obtained by subtracting the distribution ratio of the minimum positive vulnerability value ($\delta+$) and the maximum negative vulnerability value (δ^-) as shown in Eq. (4.20). This equation expresses what portion of all errors produced at the output are critical for the CNN in terms of misclassification which is equivalent to the probability that a fault misclassifies the CNN's output.

$$VF_{target\_neuron} = 1 - (distribution\_ratio_{\delta^+} - distribution\_ratio_{\delta^-}) \tag{4.20}$$

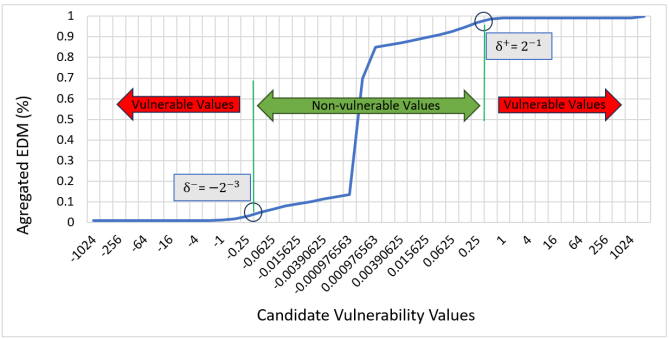


Figure 4.10: Mapping obtained Vulnerability Value Range (VVR) to aggregated Error Distribution Map (EDM) for VF calculation.

The VF for a filter is obtained similarly. The EDM and VVR for the target filter are obtained in *step 1* and *step 2* and they are exploited to provide the VF for the target filter. Deriving the VF for individual neurons and filters leads to obtaining the VF for CONV layers and the entire CNN model. Since the analysis is performed for neurons and filters separately, it can provide separate VF for layers and for the entire model based on filters and neurons. The Layer Vulnerability Factor (LVF) is the average of VF for all activations/filters.

To obtain the total Model Vulnerability Factor (MVF) of the entire model using the obtained detailed VFs based on the activations and filters analysis, Eq. (4.21) is introduced, where L is the total number of layers in a CNN, N_l and W_l are the total number of output activations and weights in layer l. This equation is a layerwise weighted average of LVFs throughout the layers of a CNN including both activations and filters vulnerability analysis, leading to the MVF for the entire model, representing the probability of misclassification if a fault occurs either in activations or weights.

$$MVF_{total} = \frac{\sum_{l=1}^{L}(\frac{N_l}{N_l+W_l} \times LVF_{act_l} + \frac{W_l}{N_l+W_l} \times LVF_{weight_l})}{L} \tag{4.21}$$

The DeepVigor+ analysis can be conducted for all neurons/filters in a CNN. However, performing a complete analysis is obstructive and time-consuming, particularly for emerging CNNs. To address the scalability issue in resilience analysis for huge CNNs, we exploit the stratified sampling concept to reduce the sampling size for statistical DeepVigor+ analysis. Stratified random sampling is a method to increase the accuracy of estimation in sampling by dividing the population into subgroups called *strata* and random samples can be selected within them [217].

In DeepVigor+, the analysis is performed layer by layer. To take the benefits of stratified sampling for reducing the execution time of the analysis, we assume that each output channel in a layer is one stratum. Thereafter, a portion of all output channels specified by *channel sampling ratio* is considered and some random channels are selected. Within each channel, the number of analyzed neurons is determined by the $log_2(\#neurons)$ in the target channel, which is selected randomly. Therefore, the number of analyses can be described with Eq. (4.22).

$$\#analyzed\ neurons = \sum_{l \in layers} \lceil channel\_sampling\_ratio \times out\_channel_l \rceil \times log_2(\#neurons^l_{channel}) \tag{4.22}$$

4.2.4 Experimental Setup

4.2.4.1 DeepVigor+ Implementation DeepVigor+ is fully implemented using Python and the Pytorch library. The source code of DeepVigor+ is fully open-source in https://github.com/mhahmadilivany/DeepVigor as a tool to enable researchers and engineers to adopt it for the resilience analysis of DNNs. The user can obtain the VF for the CONV layers of a target CNN based on analyzing neurons or weights by determining it through some inputs. User can specify the following inputs for the analysis:

- Target pre-trained CNN: the tool loads the pre-trained set of weights, in readable formats for pytorch (e.g., .h5 or .pth).
- Dataset: the tool loads the validation data from the dataset.
- Analysis method: the tool performs resilience analysis for neurons (OFMaps) or filters (weights).
- Sampling method: the tool performs either complete analysis or stratified random sampling based on *channel sampling ratio*.
- Channel sampling ratio: in the case of stratified random sampling, the channel sampling ratio should be specified.

With the determined inputs, the tool performs the analysis and outputs the Layer Vulnerability Factor (LVF) for each layer and the Model Vulnerability Factor (MVF) for the entire CNN model. In this section, we perform the DeepVigor+ analysis on one batch of 100 images in the test set. Note, that all experiments consider 32-bit floating-point data representation.

4.2.4.2 Validating DeepVigor+ by Fault Injection We use Fault Injection (FI) to validate the results of DeepVigor+. To that end, we validate the VF for channels by analyzing filters of CNNs using complete DeepVigor+ and weights FI. In this regard, before an inference, a random 3D filter in a specified layer is selected. In the FI campaign, one bit of one weight in the target filter is flipped considering 32-bit floating-point data representation and the inference is performed on the same data as the DeepVigor+ analysis was conducted. The same FI process is carried out for all bits of weights in the target filter resulting in the Channel Vulnerability Factor (CVF) calculated by the ratio of output misclassifications compared to the golden outcomes. The obtained CVF in FI for each channel is compared with the one in complete DeepVigor+ for the corresponding channel, and their absolute errors are

reported. In all DNNs under the experiment, 15% of all channels throughout the CONV layers are passed through the FI campaign for validation.

To show the accuracy of the result in exploited stratified sampling in DeepVigor+, we compare the obtained VF for channels (CVF) and layers (LVF) in sampling DeepVigor+ vs. complete DeepVigor+ for both activations and filters. The absolute difference between obtained VFs is derived to show the VF estimation in sampling analysis. In order to achieve more accurate results, different *channel sampling ratio*s are experimented including 5%, 10%, 15%, and 20%. Since neurons and channels are selected randomly in the stratified sampling, each sampled analysis is repeated 50 times and the maximum and mean absolute errors are reported.

Furthermore, to show the efficiency of DeepVigor+ analysis, we present the number of simulations for DeepVigor+ and how it outperforms FI. First, we compare the number of simulations in complete analysis of DeepVigor+ against exhaustive FI. Also, we compare the number of simulations for sampling DeepVigor+ with state-of-the-art Statistical Fault Injection (SFI) methods for CNNs proposed in [200], considering layer-wise SFI, data-unaware SFI, and data-aware SFI. Finally, the execution time for the complete and sampling DeepVigor+ is demonstrated on an *NVIDIA A100 GPU* to showcase the efficiency of the proposed method.

The number of simulations in an SFI experiment for conventional computing hardware used to be specified by Eq. (4.23), where *n* is the number of samples, *N* is total fault space, *e* is the error and *t* is determined based on the expected confidence-level [144].

$$n = \frac{N}{1 + e^2 \times \frac{N-1}{t^2 \times p(p-1)}} \tag{4.23}$$

However, it is shown that applying Eq. (4.23) to the entire DNN does not result in statistically accurate results [200]. Authors in [200] introduced multiple methods to improve the accuracy of SFI as well as to reduce the number of simulations. Accordingly, layer-wise SFI, data-unaware SFI and data-aware SFI were presented, which generally apply Eq. (4.23) to each layer separately. Data-unaware SFI and data-aware SFI methods consider FI at the bit level to improve the statistical experiments. In data-unaware $p = 0.5$ for all bits in Eq. (4.23), whereas in data-aware SFI requires a pre-analysis to specify the *p* for each bit position in the data representation.

4.2.4.3 CNNs Under Study In this section, Deepvigor+ is executed and validated on six pre-trained DNNs using various datasets. DNNs under analysis include VGG-11 trained on CIFAR-10, VGG-16, ResNet-18-C and MobileNetV2 trained on CIFAR-100, ResNet-18-I, and ResNet-34 trained on ImageNet. The baseline accuracy, number of channels and neurons for each DNN are shown in Table 4.3. All experiments in this section are performed on an *NVIDIA A100 GPU* accompanied by *AMD EPYC 7742 64-core CPU*.

4.2.5 Results

4.2.5.1 DeepVigor+ Accuracy Compared to FI To analyze the accuracy of the obtained VFs by DeepVigor+, we perform FI experiments on 15% of randomly selected channels in all CNNs in Table 4.3 and present the Mean Absolute Error (MEA), as described in subsection 4.2.4.1. Table 4.4 presents the MEA results comparing the Channel Vulnerability Factor (CVF) by the complete DeepVigor+ filters analysis vs full FI into 15% of channels in each DNN. Noteworthy that the acceptable mean error is 1% [200].

The results in Table 4.4 indicate that the mean absolute error by DeepVigor+ compared to exact FI throughout the DNNs is between 0.819% to 0.978%, i.e., always less than 1%.

Table 4.3: The CNNs under study for DeepVigor+ validation.

DNN	Dataset	Baseline accuracy	# of CONV layers	# of channels	# of neurons
VGG-11	Cifar-10	92.52%	8	2,752	232,448
VGG-16	Cifar-100	66.97%	13	4,224	185,344
ResNet-18-C	Cifar-100	70.26%	20	4,800	666,624
MobileNetV2	Cifar-100	61.27%	54	17,188	854,064
ResNet-18-I	ImageNet	69.19%	20	4,800	2,182,656
ResNet-34	ImageNet	73.04%	36	8,512	3,437,056

These results demonstrate that DeepVigor+ is able to provide precise VFs and meets the expectation of an acceptable error with respect to exhaustive FI experiments.

Table 4.4: Absolute error for CVF in DeepVigor+ and fault injection for 15% of the channels in CNNs.

CNN	VGG-11	VGG-16	ResNet-18-C	Mobile-NetV2	ResNet-18-I	ResNet-34
Mean Absolute Error	0.819	0.938	0.933	0.874	0.978	0.878

The main sources of error in VF calculation in DeepVigor+ are:

1. As discussed in subsection 4.2.2.2, DeepVigor+ approximates the error propagation in neurons based on the values of power of 2. This error approximation introduces an error to the resilience analysis which is also reflected in VF calculation.

2. DeepVigor+ assumes that bitflips resulting in big values are critical for DNNs and the *distribution ratio* for large values are always considered critical. However, in some cases, these values don't result in misclassification. This phenomenon is another reason for a minor difference between the VF by FI and DeepVigor+.

4.2.5.2 Sampling Analysis vs. Complete Analysis This subsection presents the results for sampling DeepVigor+ and compares its VF results against the complete analysis to show how accurate sampling DeepVigor+ is. Channel Vulnerability Factor (CVF) and Layer Vulnerability Factor (LVF) are derived as described in subsection 4.2.3 and the maximum and mean absolute error for obtained CVFs and LVFs in complete and sampling DeepVigor+ for neurons and filters are presented separately. In sampling DeepVigor+, *channel sampling ratio* is explored.

Table 4.5 indicates the absolute errors over various *channel sampling ratio* for DNNs under study, in both neurons and filters analysis. Each sampling analysis is repeated 50 times to observe the effect of random selections in stratified sampling. Based on the results, the minimum absolute error throughout the experiments for both CVF and LVF is very close to 0. As observed, the difference between obtained CVFs and LVFs throughout the results is minimal, demonstrating the effectiveness of the exploited stratified sampling.

In all experiments, the Mean Absolute Error (MAE) for CVF does not vary since the sampling method within channels is similar (i.e., the logarithm of the number of OFMaps).

Table 4.5: Average absolute error analysis over 50 executions for sampling DeepVigor+ compared to complete analysis.

		Activations analysis				Filters analysis			
DNN	channel sampling ratio	MAE CVF	Max error CVF	MAE LVF	Max error LVF	MAE CVF	Max error CVF	MAE LVF	Max error LVF
VGG-11	5%	0.0004%	0.011%	0.0010%	0.012%	0.064%	0.171%	0.019%	0.125%
	10%	0.0004%	0.008%	0.0008%	0.008%	0.064%	0.148%	0.015%	0.097%
	15%	0.0004%	0.004%	0.0007%	0.007%	0.063%	0.130%	0.011%	0.083%
	20%	0.0004%	0.006%	0.0007%	0.007%	0.063%	0.128%	0.008%	0.076%
VGG-16	5%	0.037%	0.141%	0.033%	0.250%	0.045%	0.216%	0.017%	0.183%
	10%	0.038%	0.144%	0.022%	0.188%	0.044%	0.116%	0.010%	0.094%
	15%	0.038%	0.118%	0.018%	0.168%	0.043%	0.137%	0.008%	0.064%
	20%	0.038%	0.130%	0.015%	0.140%	0.043%	0.141%	0.007%	0.065%
ResNet-18-C	5%	0.029%	0.097%	0.054%	0.633%	0.045%	0.153%	0.016%	0.137%
	10%	0.029%	0.092%	0.038%	0.386%	0.045%	0.123%	0.011%	0.076%
	15%	0.029%	0.093%	0.031%	0.272%	0.045%	0.127%	0.010%	0.083%
	20%	0.029%	0.071%	0.026%	0.300%	0.045%	0.124%	0.008%	0.085%
MobileNetV2	5%	0.031%	1.443%	0.079%	2.803%	0.030%	0.893%	0.013%	0.983%
	10%	0.033%	0.841%	0.053%	1.252%	0.030%	0.401%	0.009%	0.372%
	15%	0.032%	0.586%	0.044%	0.742%	0.030%	0.468%	0.007%	0.346%
	20%	0.032%	0.465%	0.037%	0.720%	0.030%	0.401%	0.006%	0.320%
ResNet-18-I	5%	0.005%	0.071%	0.007%	0.168%	0.041%	0.115%	0.016%	0.114%
	10%	0.005%	0.070%	0.005%	0.212%	0.041%	0.096%	0.010%	0.061%
	15%	0.005%	0.054%	0.004%	0.108%	0.041%	0.083%	0.008%	0.062%
	20%	0.005%	0.045%	0.003%	0.082%	0.042%	0.089%	0.006%	0.032%
ResNet-34	5%	0.003%	0.061%	0.005%	0.140%	0.045%	0.136%	0.016%	0.087%
	10%	0.003%	0.059%	0.003%	0.084%	0.045%	0.127%	0.011%	0.097%
	15%	0.003%	0.054%	0.002%	0.089%	0.046%	0.122%	0.009%	0.080%
	20%	0.003%	0.048%	0.002%	0.063%	0.045%	0.120%	0.007%	0.061%

Also, it is observed that the MEA CVF is less than 0.065% in all experiments for both activations and filters sampling analysis. It means that the averaged VF for the logarithm-based random sampling within each channel results in a highly accurate CVF. This phenomenon is a result of the unified distribution of weights within a channel in a pre-trained DNN. The maximum observed error in neuron analysis is 0.004% to 1.443% throughout DNNs, whereas the mean error remains low, meaning that for most of the channels, obtained CVFs are highly accurate leading to an overall high accuracy for obtained CVFs.

On the other hand, *channel sampling ratio* directly affects the accuracy of LVF calculations. According to Table 4.5, the error of LVF decreases with the increase of *channel sampling ratio*. Considering both MEA and maximum error for LVF, a 10% *channel sampling ratio* can guarantee a minimal error for VF calculations. Based on the MEA of complete DeepVigor+ compared to exhaustive FI in Table 4.4, sampling DeepVigor+ with 10% *channel sampling ratio* ensures that the overall error of obtained VFs will be below 1%.

In conclusion, the proposed stratified sampling in DeepVigor+ results in highly accurate VF for channels and layers obtained from both activations and filter analysis with a *channel sampling ratio* of 10%. Sampling DeepVigor+ results in a higher error than state-of-the-art statistic FI approaches such as data-aware and data-unaware, yet it meets the requirement of average error for resilience study which is 1%.

4.2.5.3 Run-Time and Scalability Investigation To show the excellence of DeepVigor+ in terms of complexity, scalability and execution time against FI, first, we investigate the complexity of each based on the required number of simulations (i.e., forward pass executions) to obtain VF. Then we present the execution time for the complete and sampling DeepVigor+ on an *NVIDIA A100 GPU*.

Exhaustive FI is the most accurate method for determining precise VFs. In exhaustive FI, the required number of simulations equals the number of activations/weights times bit-width (i.e., 32 bits). Therefore, its complexity is proportional linearly to the size of

DNNs. Whereas the complete DeepVigor+ analysis estimates VFs with high accuracy and significantly lower complexity. Although its complexity is affected by the size of DNNs, DeepVigor+ analysis exploits various optimizations to reduce the number of simulations resulting in significantly lower complexity than exhaustive FI. This is evidenced by the results in Table 4.6 where the required number of simulations is compared between Exhaustive FI and complete DeepVigor+ analysis, for activations and filters separately.

Table 4.6: Number of simulations for exhaustive FI vs complete DeepVigor+ for activations and filters analysis.

	Activations		Filters	
DNNs	Exhaustive FI	DeepVigor+	Exhaustive FI	DeepVigor+
VGG-11	7,438,336	1,636,414	294,967,296	11,035
VGG-16	5,931,008	2,018,447	470,734,848	13,704
ResNet-18-C	21,331,968	3,387,189	357,095,424	15,906
MobileNet-V2	27,330,048	3,029,125	74,176,512	42,234
ResNet-18-I	69,844,992	20,730,314	357,341,184	23,450
ResNet-34	109,985,792	28,821,489	680,564,736	43,098

As observed, for each DNN, DeepVigor+ complete analysis requires significantly lower executions either in activations or filters analysis. Throughout the results, activations analysis by complete DeepVigor+ is 2.93 to 9.02 times faster than exhaustive FI. According to our detailed investigations, activations analysis by DeepVigor+ requires 6 forward simulations per neuron, on average, throughout the DNNs under study. In this regard, the introduced loss function to obtain the vulnerability values (*box 4* in Fig 4.9) contributes to skipping the analysis for up to 26% of neurons. Moreover, the vulnerability value can be obtained by a single forward simulation for up to 55% of neurons due to separating VVSS exploration (*boxes 7* and *8* in Fig. 4.9).

In the complete filters analysis, the DeepVigor+ fault propagation modeling implies a huge impact on the number of simulations in the orders of magnitude. DeepVigor+ can derive VF for filters $1,756$ to $34,350$ times faster than exhaustive FI. To obtain the vulnerability values for channels, up to 1% of channels are skipped by leveraging the loss function, and up to 34% of channels require one forward simulation. These results indicate that complete DeepVigor+ provides accurate VF for neurons and channels of CNNs with significantly lower complexity and shorter execution time than exhaustive FI enabled by its optimal fault propagation modeling and analysis.

On the other hand, sampling DeepVigor+ is proposed to further reduce the execution time and complexity of resilience analysis and achieve a scalable method. To show its performance against statistical FI (SFI), Table 4.7 compares the number of simulations for various state-of-the-art SFI [200] with sampling DeepVigor+. As presented, data-aware SFI leads to the least number of executions in FI-based simulation. It is observed that DeepVigor+ sampling activations analysis with 10% channel sampling ratio leads to 8.72 to 20.5 times fewer simulations compared to data-aware SFI. For the filters analysis, DeepVigor+ obtains their VF with 59.4 up to 96.2 times fewer simulations than data-aware SFI.

To obtain the Model Vulnerability Factor (MVF) both activations and filters should be analyzed separately, based on Eq. (4.21). Therefore, sampling DeepVigor+ accelerates the process from 14.9 up to 26.9 times throughout the DNNs. The scalability and speed of the method are achieved by both channel sampling ratio and logarithmic sampling within them. It can be observed that with the remarkable growth of DNNs under analysis in their number of parameters, the number of simulations in DeepVigor+ does not grow linearly.

Table 4.7: Comparison of required simulations for statistical FI [200] and sampling DeepVigor+.

	Activations Analysis				Filters Analysis			
Analysis method	Layer-wise	Data-unaware	Data-aware	Sampling DeepVigor+ 10%	Layer-wise	Data-unaware	Data-aware	Sampling DeepVigor+ 10%
VGG-11	74,863	1,562,657	71,173	**4,596**	75,351	2,141,913	66,934	**1,038**
VGG-16	117,992	1,839,889	106,513	**10,283**	123,266	3,588,834	112,151	**1,476**
ResNet-18-C	188,644	4,264,406	181,911	**15,996**	189,772	5,253,096	164,159	**1,706**
MobileNetV2	493,871	8,402,977	452,650	**22,077**	475,407	8,307,671	259,614	**4,364**
ResNet-18-I	191,205	5,407,659	189,325	**21,680**	190,896	5,358,315	167,447	**2,178**
ResNet-34	344,042	9,624,374	340,383	**39,002**	344,285	10,031,494	313,484	**4,003**

To demonstrate the execution time of DeepVigor+ analysis, we employed an *NVIDIA A100 GPU* and performed sampling DeepVigor+ for activations and filters with different channel sampling ratios. Table 4.8 and Table 4.9 present the average execution time over 50 executions of the method for complete and sampling DeepVigor+ with different channel sampling ratios, for activations and filters respectively. It is observed that VFs can be obtained in a few minutes for the experimented DNNs. Considering 10% channel sampling ratio for both activations and filters analysis, the total MVF for VGG-11, VGG-16, ResNet-18-C, MobileNet-V2, ResNet-18-I and ResNet-34 is obtained almost in 8.7 minutes, 9.6 minutes, 12.7 minutes, 43.5 minutes, 19.4 minutes and 46 minutes, respectively. It is worth mentioning that the complete DeepVigor+ analysis for the DNNs under study takes almost 22 hours for VGG-16 and 18.5 days for ResNet-34, on the same GPU. Fast execution and accurate estimation of VF obtained by sampling DeepVigor+ analysis provide a remarkable opportunity for a high-speed resilience analysis for DNNs.

Table 4.8: Average execution time over 50 repetitions on A100 GPU for DeepVigor+ activations analysis, with different channel sampling ratios.

DNN	5%	10%	15%	20%	complete analysis
VGG-11	203 sec (≈ 3.4 min)	415 sec (≈ 6.9 min)	625 sec (≈ 10.4 min)	838 sec (≈ 13.9 min)	66,083 sec (≈ 0.7 days)
VGG-16	223 sec (≈ 3.7 min)	453 sec (≈ 7.5 min)	676 sec (≈ 11 min)	915 sec (≈ 15 min)	43,326 sec (≈ 0.5 days)
ResNet-18-C	284 sec (≈ 4.7 min)	580 sec (≈ 9.5 min)	876 sec (≈ 14.5 min)	1,171 sec (≈ 19.5 min)	94,939 (≈ 1.1 days)
MobileNetV2	1,037 sec (≈ 17 min)	2,097 sec (≈ 35 min)	3,153 sec (≈ 52.5 min)	4,071 sec (≈ 68 min)	211,868 (≈ 2.4 days)
ResNet-18-I	448 sec (≈ 7.4 min)	917 sec (≈ 15 min)	1,390 sec (≈ 23 min)	1,866 sec (≈ 31 min)	634,872 (≈ 7.3 days)
ResNet-34	1,100 sec (≈ 18 min)	2,237 sec (≈ 37 min)	3398 sec (≈ 56 min)	4549 sec (≈ 75 min)	1,402,362 (≈ 16.2 days)

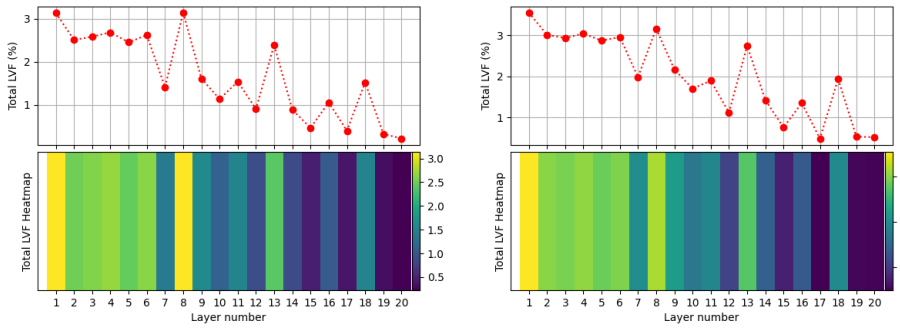
4.2.5.4 Reliability Visualization and Comparison for CNNs It has been shown that VF for DNNs' layers and the entire model can be obtained accurately in a few minutes by DeepVigor+. The obtained VF results by DeepVigor+ can be used to visualize the vulnerability of layers within a DNN and identify more vulnerable ones. Fig. 4.11 illustrates the LVF comparison for ResNet-18 trained on CIFAR-100 and ImageNet datasets, while the total LVF for each layer is obtained based on the numerator of Eq. (4.21). This visualization sketches how vulnerable each layer is to single faults compared to each other. As ob-

Table 4.9: Average execution time over 50 repetitions on A100 GPU for DeepVigor+ filters analysis, with different channel sampling ratios.

DNN	5%	10%	15%	20%	complete analysis
VGG-11	57 sec (≈ 0.95 min)	111 sec (≈ 1.8 min)	170 sec (≈ 2.8 min)	219 sec (≈ 3.6 min)	23,031 sec (≈ 0.26 days)
VGG-16	61 sec (≈ 1 min)	125 sec (≈ 2.1 min)	168 sec (≈ 3.1 min)	245 sec (≈ 4.1 min)	35,634 sec (≈ 0.4 days)
ResNet-18-C	92 sec (≈ 1.5 min)	184 sec (≈ 3 min)	275 sec (≈ 4.6 min)	367 sec (≈ 6.1 min)	60,410 sec (≈ 0.7 days)
MobileNetV2	257 sec (≈ 4.3 min)	513 sec (≈ 8.5 min)	759 sec (≈ 12.6 min)	1,025 sec (≈ 17.1 min)	39,713 sec (≈ 0.46 days)
ResNet-18-I	117 sec (≈ 1.9 min)	250 sec (≈ 4.1 min)	370 sec (≈ 6.1 min)	506 sec (≈ 8.4 min)	134,164 sec (≈ 1.5 days)
ResNet-34	259 sec (≈ 4.3 min)	524 sec (≈ 8.7 min)	795 sec (≈ 13.2 min)	1,064 sec (≈ 17.7 min)	201,305 sec (≈ 2.3 days)

served, in both ResNet18-C and ResNet18-I, the first layers are more vulnerable than the latter ones. Therefore, DeepVigor+ enables vulnerability visualization and comparison within a DNN.

Furthermore, DeepVigor+ results in total MVF based on Eq. (4.21) which is the weighted average of obtained LVFs providing a comprehensive examination of vulnerability between different DNNs. Fig. 4.12 indicates MVFs for activations and filters of DNNs, separately, as well as their total MVF. As a result, it is observed that activations are more vulnerable than weights. However, weights generally contribute more to the total MVF since their memory footprint is higher than that of activations in DNNs. Based on total MVF, VGG-16 is the least vulnerable DNN (MVF = 1.19%) and MobileNet-V2 is the most vulnerable one (MVF = 2.76%).



(a) ResNet-18-C *(b) ResNet-18-I*

Figure 4.11: LVF visualization and comparison for ResNet-18 trained on a) CIFAR-100 and b) ImageNet.

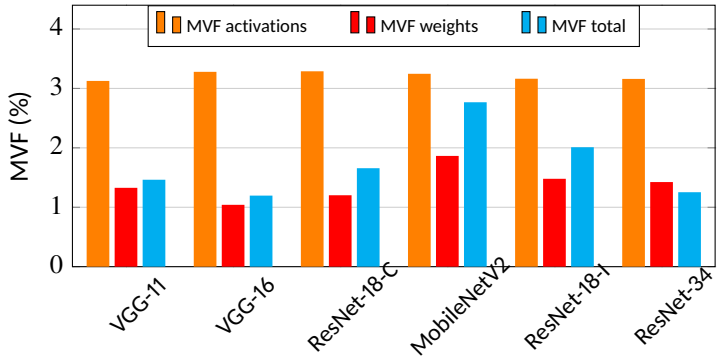


Figure 4.12: MVF comparison for CNNs based on activations, filters and the entire model derived by DeepVigor+.

4.2.5.5 Impact of Input Data on the Quality of Results To obtain the VFs in DeepVigor+, we considered one batch of 100 data. Nonetheless, to investigate the impact of data on the quality of analysis results, we repeat the experiments for different batches of data with the size of 100 input data and derive DNNs' total MVF. Fig. 4.13 illustrates the obtained total MVF for DNNs over 8 different batches of data. As observed, the variation between the total MVF for each DNN is negligible for different batches of data, demonstrating that analyzing DNNs' resilience with one batch of data provides confident results.

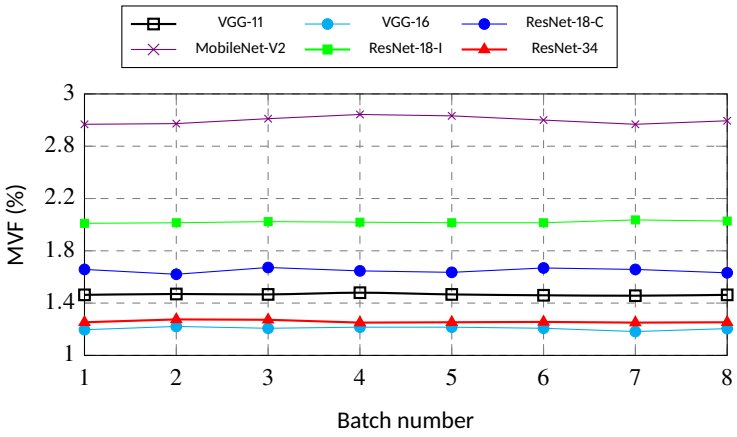


Figure 4.13: Total MVF variation over different batches of data for all DNNs.

4.2.6 Discussion

As shown, DeepVigor+ achieves a fast, scalable and accurate resilience analysis for emerging DNNs. The analysis provided by DeepVigor+ is not limited to fault resilience assessment, but it can be exploited for designing fault-tolerant and resilient DNNs. Identifying more vulnerable channels and layers can lead to cost-effective selective fault mitigation techniques as well as a comparative investigation between different architectures of DNNs. Moreover, it enables design space exploration to identify more resilient DNNs against faults.

The other output of DeepVigor+ is VVR representing the values that a fault should

affect neuron/weight to misclassify DNN's golden results. These values can be used for obtaining VFs and identifying more vulnerable components as well as for fault detection at inference and identifying bit-level resilience analysis since they are represented in the power of 2 values and can be mapped down to bits.

Yet, there are some constraints in this method to be considered and extended in future research:

- It is assumed that the parameters within the layers of DNNs under analysis are unifiedly distributed among channels and their places are not resorted after training. In such cases, a higher channel sampling ratio is needed to obtain accurate VF results.
- DeepVigor+ supports a single-bit fault model in input activations and weights of convolutional layers and obtained VF corresponds to this fault model. For multi-bit fault models, the corresponding error propagation should be applied.
- The error propagation analysis presented in this section is based on 32-bit floating point data representation. However, the same concept can be extended and applied to fixed-point and integer data representations for QNNs resilience analysis.
- DeepVigor+ analysis is hardware-agnostic. It assumes that the accelerator architecture is dataflow and each neuron in a layer utilizes individual hardware resources. If hardware resources are shared between the neurons of a layer, a hardware-aware analysis through the error propagation step should be applied.

This section addresses one of the major challenges in fault resilience analysis for DNNs in the literature. It introduces DeepVigor+, the first semi-analytical scalable alternative method to fault injection, quantifying emerging DNN's resilience accurately in a short time. DeepVigor+ is facilitated by optimal fault propagation modeling in DNNs accompanied by stratified sampling tackling the scalability problem for resilience analysis for DNNs. This open-source method unleashes a fast resilience assessment, enabling fine-grain evaluation and design space exploration for various fault-tolerant and cost-effective designs for DNNs.

The results indicate that DeepVigor+ derives vulnerability factors for layers and the entire model of DNNs with less than 1% error, with 14.9 up to 26.9 times fewer simulations than the best-known state-of-the-art statistical FI. It is shown that DNN's Model Vulnerability Factor can be obtained within minutes by analyzing their activations and weights. DeepVigor+ is presented as an open-source tool for researchers and engineers to enable them to exploit it for DNNs' fault resilience assessment and enhancement.

4.3 QDeepVigor: Applications for QNNs

DeepVigor is primarily developed for the resilience analysis of CNNs using 32-bit floating point data representation. As discussed, DeepVigor can be extended for other data types such as integers. In addition, DeepVigor's outcomes can be utilized for fault-tolerant design in DNN accelerators. Furthermore, QDeepVigor can be leveraged by FI to accelerate hardware-aware fault simulation, leading to a hybrid method for reliability assessment. In this section, we present QDeepVigor and some case studies where it is exploited for the aforementioned purposes.

4.3.1 Cross-layer reliability enhancement for QNNs accelerators

Quantized Neural Networks (QNNs) are proposed to improve the computational efficiency of DNNs by reducing the bit precision. However, their reliability is a concern, particularly in

safety-critical applications. Throughout the literature, protecting DNNs against soft errors is primarily achieved through architecture-level methods such as hardened PEs or Triple Modular Redundancy (TMR) [168]. 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.

In related works, the criticality of neurons has been identified based on their contribution scores to outputs [7, 202, 210, 211]. 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. In this section, we present QDeepVigor, an extension of DeepVigor, to identify critical neurons in QNNs. The resilience analysis enables us to design a method for correcting soft errors in the datapath of DNN accelerators.

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. This modifies the QNN in a way that a Lightweight Correction Unit (LCU) corrects the detected 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.

4.3.1.1 Accelerator Model Fig. 4.14 illustrates the accelerator model considered in this work, which is inspired by [183]. 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 the 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 the next subsections.

4.3.1.2 Identifying Critical Neurons by QDeepVigor Algorithm 4.2 presents QDeepVigor, the resilience analysis of QNNs to obtain 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* in Algorithm 4.2) by the gradients of a loss function ($\mathscr{L}$) 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* in Algorithm 4.2). Thereafter, it maps the obtained δ to a corresponding possible bitflip location in the data type (*lines 8*, *9* in Algorithm 4.2) and counts it as a vulnerable location (*lines 10*, *11* in Algorithm 4.2). In the end, regarding the count of vulnerable times for each bit, it

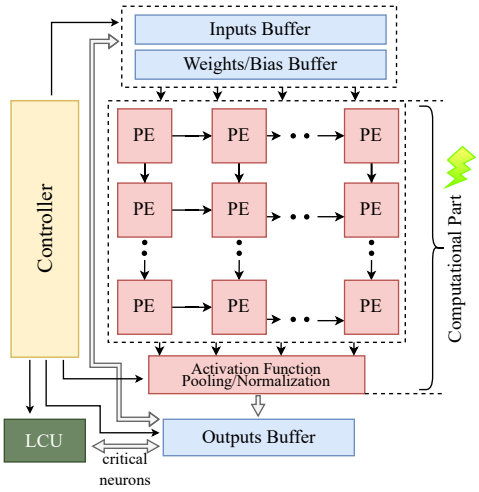


Figure 4.14: An abstract view of the accelerator and where the faults may happen.

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* in Algorithm 4.2).

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 bitflip 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.

4.3.1.3 Resilience Enhancement by LCU and Neuron Splitting 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. 4.15 depicts how a critical neuron is split into two halves. As 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 neurons 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 computational part of the accelerator. The QNN is modified in a way that the selected critical neurons from the analysis are split. The modified QNN 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. 4.14).

Algorithm 4.2 QDeepVigor: Fault Resilience Analysis for 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]; $\mathscr{E}_{c_t}$ 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: $\mathscr{L} = sigmoid(\sum_{j=0}^{N}(\mathscr{E}_{c_t} - \mathscr{E}_{c_j}))$
4: *grad* = $\nabla\mathscr{L}/out_{neuron}$
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: $bit_{upper} = \lfloor log(r_{upper}) \rfloor + 1$;
9: $bit_{lower} = \lfloor log(|r_{lower}|) \rfloor$;
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**;

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 is 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, 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. 4.16 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.

4.3.1.4 Experimental Setup The experimented QNNs in this work are fully quantized (all parameters and activation) to 8-bit signed integer using TFLite [70]. The experiments in this work have been performed on a 7-layer MLP and LeNet-5 trained on MNIST as well as an AlexNet 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 are implemented in PyTorch. The resilience analysis is conducted over the training set. The critical neurons regarding different thresh-

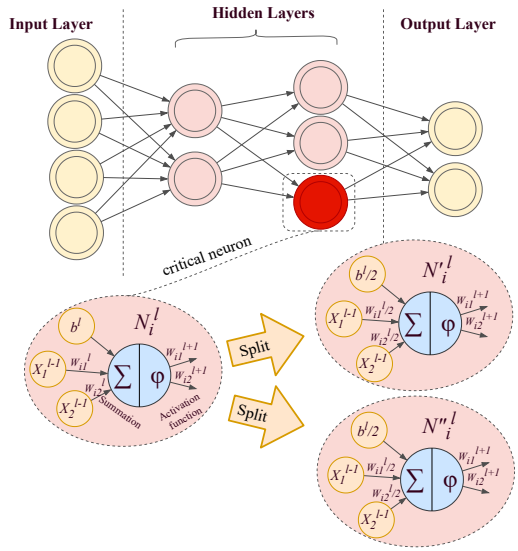


Figure 4.15: Splitting critical neurons in a QNN by halving the input parameters.

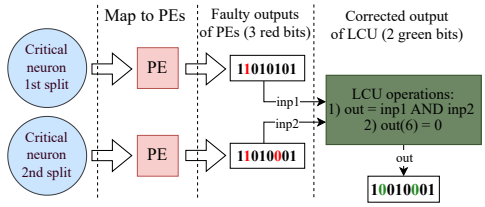


Figure 4.16: An example of how LCU corrects faulty critical neurons.

olds 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 Eq. (4.7) [144], 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:

- accuracy loss of QNNs over the fault injection,
- the portion of critical faults in a fault injection campaign. Critical faults are the ones that misclassify the network from its golden classification.

4.3.1.5 Results: 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

4.10 presents the number of critical neurons in different NVFs ranging from 0% (all neurons are critical) to 50% (no neuron is critical). According to Table 4.10, 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 smaller number of critical neurons to be protected. Table 4.10 represents the overhead of any protection mechanism over the critical neurons.

Table 4.10: 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. 4.17 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. 4.17-(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. 4.17, 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 4.10, 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 4.10, 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.

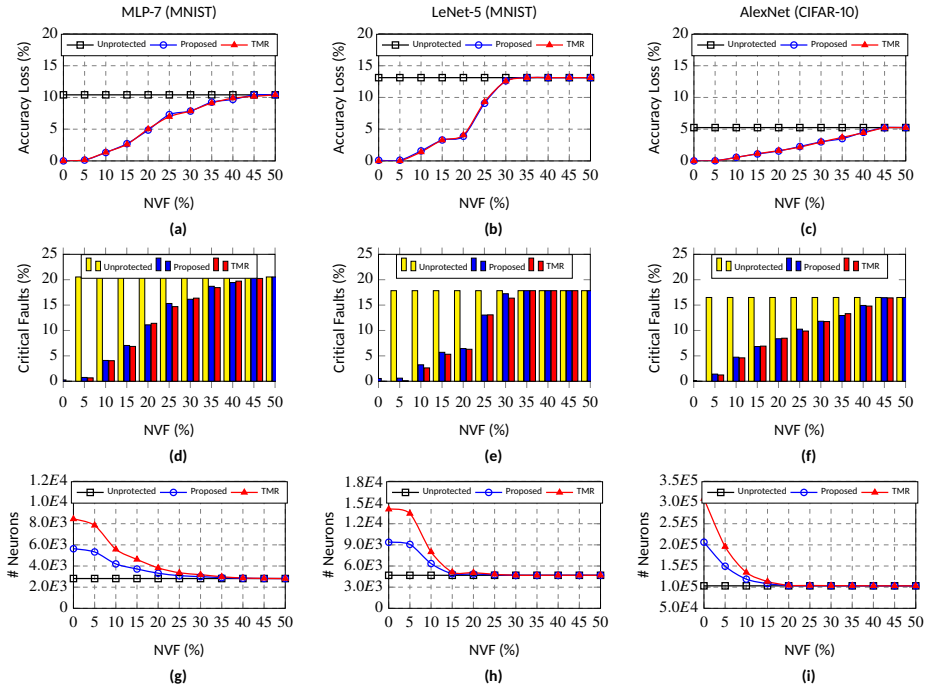


Figure 4.17: QNNs comparison in terms of accuracy loss (a-c), critical faults (d-f), and network size (g-i) under different levels of protection: unprotected, proposed protection, and TMR, considering different thresholds for NVF from 0% to 50%.

4.3.2 A Hybrid Method for QNNs' Reliability Assessment

This subsection introduces a novel hybrid reliability assessment method for QNNs which targets systolic-array-based DNN accelerators using FI and QDeepVigor. The proposed hybrid method adopts a software-based hardware-aware FI simulator for SAs, called SAFFIRA [225]; thus offering the advantage of being more accurate than a hardware-agnostic tool, yet much faster than traditional RTL-level simulations. QDeepVigor is exploited to prune the fault space to further optimize and speed up the reliability assessment process.

4.3.2.1 Hybrid Method: QDeepVigor and SAFFIRA The methodology for the proposed hybrid reliability assessment method is illustrated in Fig. 4.18. The inputs are a pre-trained QNN and a fault list which is generated based on random transient faults in the registers of the systolic array's PEs. The fault simulation by SAFFIRA is performed from the beginning of the QNN to the fault's location to obtain the erroneous output of the corresponding neuron. Then, we map the deviation at the erroneous neuron's output induced by a fault in PE's registers to the obtained vulnerability ranges for the corresponding neuron. 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.

To prune the fault space, we adopt QDeepVigor to obtain vulnerability analysis for QNNs, as described in the previous subsection. To this end, we find error values for each OFMap that misclassifies the network output. Let $\delta_k^l(X_i)$ be an added positive or negative error value to an OFMap by a fault in the k-th neuron at layer l with input data X_i. For each

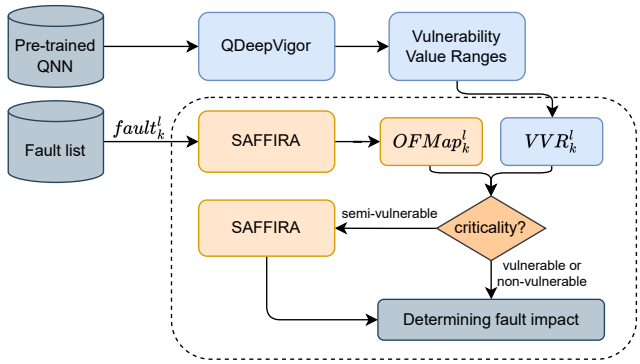


Figure 4.18: The hybrid reliability assessment method for QNNs on SAs.

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 (VVR) for each neuron, as shown in Fig. 4.19. 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.

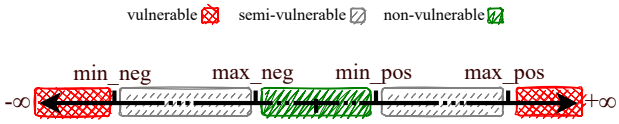


Figure 4.19: VVRs for fault space pruning

4.3.2.2 Results: Simulation Speed-up To evaluate the methodology, experiments were performed using a 16-bit quantized LeNet-5 trained on the MNIST dataset. The fault model is a random bitflip in the weight register of a random PE in SA. The target architecture is an output-stationary Systolic Array. In this experiment, faults are injected only in the first layer of the QNN. It is worth mentioning that the VVRs are obtained in less than one minute on an NVIDIA 3090 GPU. The produced fault list includes 964 random faults, each simulated with various 100 input images of the MNIST test data.

According to the obtained results, 77.48% of faults are classified as *non-vulnerable* and *vulnerable*, leading to an early stop during the fault simulation. It means that the fault simulation process is accelerated remarkably. In conclusion, the presented approach is capable of significantly reducing the FI simulation time by incorporating QDeepVigor.

4.4 Chapter Conclusions

This Chapter mainly attempted to propose novel resilience analysis methods for CNNs to address one of the most significant challenges in the literature which is scalability. Towards that end, DeepVigor is introduced as an accurate and metric-oriented hardware-agnostic method that is faster than FI. It derives vulnerability value ranges for all neurons in CNNs leading to calculating VF metrics for bits, neurons, and layers. This method addresses RQ2.1 and RQ2.2 in Chapter 1.

Based on that, DeepVigor+ is introduced to tackle the scalability of resilience analysis by exploiting an optimal fault propagation analysis. DeepVigor+ analyzes activations and parameters and acquires MVF as a total vulnerability metric for CNNs. This open-source tool can obtain a comprehensive fault resilience analysis in a few minutes with high accuracy for emerging CNNs, addressing RQ2.3 in Chapter 1. DeepVigor is also extended to support integer datatypes and its results can be used for fault enhancement as well as being integrated into hybrid fault resilience analysis approaches which addresses RQ.2.4 in Chapter 1.

5 Reliability Enhancement for CNNs

Hardware devices, including general-purpose processors (e.g., CPUs and GPUs) and specialized accelerators (e.g., FPGAs and ASICs), are widely employed to efficiently execute CNN models [52, 127]. Due to technology miniaturization, soft error rates in modern digital devices have increased in recent years, in particular in SRAM-based memories [120]. CNN accelerators store parameters (i.e., weights and biases) in memory that is prone to soft errors. Parameters in memories are not being overwritten as frequently as values in the data path, input buffers, and logic elements (e.g., buffers in PEs) [185]. Hence, bit-flips originating from soft errors in parameters are accumulative and persistent, leading to constantly producing errors throughout the inference of a CNN accelerator.

Consequently, a major concern in deploying CNNs on hardware devices is their resilience to faults in memory, particularly those affecting their parameters. Extensive studies have demonstrated that faults in CNN parameters lead to drastic accuracy drops at very low error rates [31, 164, 176, 219]. Therefore, it is crucial to enhance the fault tolerance of CNN models against faults in parameters to effectively employ them in safety-critical applications.

This Chapter attempts to tackle the aforementioned challenges concerning fault tolerance in CNNs. This Chapter addresses P3 which includes RQ3.1-3.3 and presents contributions mentioned in C3, in Chapter 1. This Chapter is based on the following publications:

I S. Mousavi, M. H. Ahmadilivani, J. Raik, M. Jenihhin, and M. Daneshtalab. ProAct: Progressive Training for Hybrid Clipped Activation Function to Enhance Resilience of DNNs. *Under review*, pages 1–12, 2024

II 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. In *The 30th IEEE International Symposium on On-Line Testing and Robust System Design (IOLTS)*, pages 1–6. Rennes, France, 2023

In the rest of the Chapter, Section 5.1 overviews the existing fault tolerance techniques for CNNs and highlights the existing gaps in the literature. Section 5.2 presents ProAct, a novel, effective, and lightweight activation restriction method that outperforms the existing ones. Section 5.3 introduces a cost-effective approach for hardening CNNs by architectural modifications. Moreover, Section 5.4 presents SentinelNN, an open-source framework that integrates DeepVigor into the proposed enhancement methods in this Chapter. Eventually, Section 5.5 concludes this Chapter.

5.1 Related Works: Fault Tolerance for CNNs

Fault-tolerant techniques to enhance the reliability of DNN accelerators against soft errors in memories can be generally carried out at the architecture and algorithm level. Architecture-level techniques are accelerator-specific and exploit hardware redundancy, laying overhead on performance, area, and power. In these techniques, computing units or memory components of the DNN accelerators are either designed to be reliable (i.e., hardened) or redundant units are included [115, 147, 152]. To mitigate the impact of faults on the deployment of CNNs and at the same time avoid the high overhead in conventional fault-tolerant techniques such as Triple Modular Redundancy (TMR), researchers proposed selective hardening approaches [7, 149, 202, 210, 211]. Here, the objective is to protect the parameters or neurons that have a larger effect on the neural network's outputs against faults and errors. Therefore, the more vulnerable neurons are identified by

resilience analysis and they are executed on hardened PEs on the target hardware.

Although these methods propose a model-level resilience analysis to identify the more vulnerable parameters/neurons, their protection techniques are restricted to FPGAs and ASICs that can be freely modified and redesigned. Whereas there exist numerous applications from high-performance to edge computing, where general-purpose computing devices such as CPUs and GPUs or hard and firm accelerator cores are deployed that do not support redesigning the hardware for fault-tolerance [9, 252].

Algorithm-level techniques, i.e., Software-Implemented Hardware Fault Tolerance (SIHFT) techniques, modify the DNN models in the software which is executable by any accelerators. Throughout the literature, several cost-effective algorithm-level fault tolerance techniques for enhancing the reliability of DNNs are presented. *Quantization* is shown to be highly effective for the resilience of CNNs [185] since it restricts the numerical range within a CNN, thus eliminating the effect of large values produced due to faults and bit-flips in a CNN. Nevertheless, apart from accuracy concerns, deploying quantized CNNs requires dedicated hardware accelerators for handling associated operations. Otherwise, they carry out the floating-point arithmetic of general-purpose computing devices [94] which leads to the reliability issues of floating-point data types, that is contradictory to the purpose of hardening by quantization. Nonetheless, the model-level fault tolerance methods are mostly orthogonal to quantization and they can be employed on top of each other to improve the resilience of DNNs.

Fault-aware training [49, 247] effectively improves the resilience of DNNs. However, it retrains the entire CNN with numerous fault injection scenarios that is not only excessively complex but also requires the possibility of having access to parameters. *Error Correction Codes (ECC)* and *Algorithm-based Fault Tolerance (ABFT)* utilize data encoding/decoding processes for real-time fault detection and correction [143, 252]. However, the practicality of these techniques in fault correction is questionable due to the overhead they introduce to memory and computations, posing a considerable challenge for CNNs that already have substantial memory and computational requirements.

Activation restriction methods [53, 93, 113] bound the activation values between layers through activation functions (i.e., ReLU) to mitigate error propagation to the outputs of CNNs. They clip the activations to 0 when their values exceed pre-identified ranges. These methods are effective in enhancing the resilience of CNNs, however, they do not provide error correction, and CNNs fail to work at high error rates due to the replacement of numerous feature maps with 0. [26] proposes a correction layer that executes each convolutional layer three times for fault detection and correction, which, however lays a prohibitive performance overhead to CNNs.

Authors in [53] present Ranger, a clipped ReLU that bounds the layer activations' output to 0 in case their value is higher than a fixed threshold. The threshold values are obtained from the maximum values at each layer seen in a validation set of the dataset. The method of clipping out-bound values to 0 in Ranger is demonstrated to be effective, however, the method does not provide optimal clipping thresholds. Hoang et al. [113] have analyzed various boundary values on the model's accuracy. Their method, named FT-ClipAct, attempts to find optimal boundary values for each layer that are not necessarily the maximum values of the layers' activations and are smaller than the maximum bounds. The authors propose a heuristic interval search algorithm based on the fault injection process to find appropriate threshold values for the ReLU activation function at each layer. FT-ClipAct incurs significant computational overhead in determining the thresholds for DNNs' layers due to the injection of faults at each step of the search algorithm. Therefore, it is unfeasible to employ it for every single neuron in a DNN.

FitAct proposes an activation function based on the *sigmoid* function that is differentiable to boundary values to optimize them with a gradient-based algorithm [93]. FitAct considers the boundary values for each neuron and smoothly maps the activation outputs to 0. Furthermore, Fitact demonstrates that for fixed-point representation, the optimal threshold values that maintain the baseline accuracy of the fault-free model tend to be smaller. While FitAct effectively enhances the resilience of DNNs, it concurrently elevates both memory overhead and the likelihood of faults occurring in the activation functions' parameters. This indicates that as the number of parameters in DNNs grows, the likelihood of faults occurring in the threshold values also increases, thereby diminishing the resilience. Furthermore, FitAct trains all the threshold parameters in the clipping activation function simultaneously, which decreases the possibility of providing the optimal threshold for each activation function.

This Chapter introduces two novel model-level hardening solutions to modify the architecture of CNNs to allow fault correction at inference inherently, outperforming the state-of-the-art methods.

5.2 ProAct: Progressive Training for Hybrid Clipped Activation Function

In this section, we attempt to address the identified issues in the existing activation restriction methods. To the best of our knowledge, for the first time, a novel activation restriction method is introduced that combines layer-wise and neuron-wise clipping incorporated with progressive training employing Knowledge Distillation (KD) [95, 112] to achieve significant resilience with negligible memory overhead in CNNs. Progressive training for hybrid clipped activation function thresholds (ProAct) empowers CNNs to mitigate error propagation to the output with a considerably lower memory overhead than the state-of-the-art methods.

5.2.1 Research Motivation

The optimal clipping threshold value for an activation function (layer-wise or neuron-wise) is the minimum possible value that maintains the accuracy of the fault-free DNN model [93, 113]. The heuristic optimization method in FT-ClipAct [113] demands extensive computation overhead as it involves conducting fault injections at every step of the search process and finding sub-optimal thresholds due to the limited number of search steps. The gradient-based optimization method in FitAct [93] improves the resilience of DNNs compared to FT-ClipAct in a less complex way. However, the obtained thresholds for neurons are not optimal since all of them are trained simultaneously in each backward pass.

To illustrate the aforementioned shortcoming, we calculate the clipping threshold values for activation functions in the AlexNet using FitAct. Afterward, we attempt to progressively reduce the clipping threshold values for the neurons layer by layer while ensuring that the model's baseline accuracy remains unaffected. We accomplish this by training the threshold parameters for each layer individually for 5 epochs, using a higher weight decay hyper-parameter. Fig. 5.1 illustrates the results for AlexNet resilience based on its accuracy under different BERs into parameters, after minimizing the neurons' thresholds in each layer progressively. It is observed that the obtained clipping thresholds by FitAct are not the optimal values and it is possible to identify more appropriate clipping threshold values to improve the resilience of DNNs. These results demonstrate the necessity for a new training mechanism to optimize clipping threshold values. In this section, we introduce the ProAct algorithm, which progressively trains threshold values layer by layer accompanied by Knowledge Distillation (KD).

Moreover, the main factor contributing to memory overhead in FitAct is the imple-

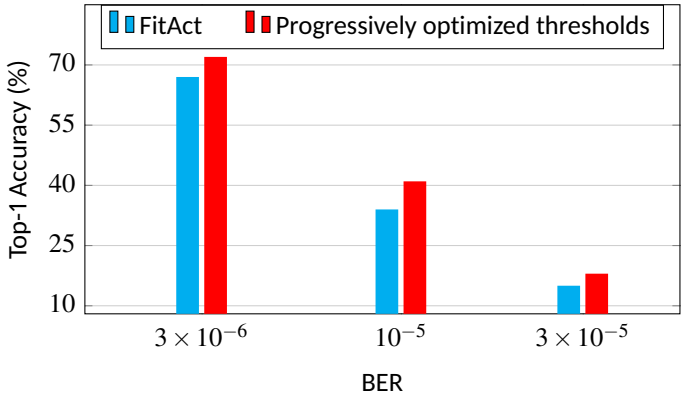


Figure 5.1: Top1-Accuracy of AlexNet under different BERs employing FitAct and progressively optimized thresholds.

mentation of clipped thresholds at the level of individual neurons. Applying an individual clipping threshold to each neuron not only enlarges memory overhead but also increases the probability of memory faults as there are more stored parameters. To comprehend the impact of neuron-wise activation restriction, we examine error propagation through *FitAct*-instrumented AlexNet by illustrating the distribution of activations of each layer without and with faults into parameters (BER = 3×10^{-5}) in Fig. 5.2. To enhance the visualization, the distribution of values is partitioned into two ranges, the left-hand side column presents the activation values between $[0, 1]$ and the right-hand side one shows them in $(1, \infty)$, respectively.

The distribution of activations in both fault-free and faulty models exhibits a similarity in the initial layers. While, by proceeding through the depth of the model, the disparity between the distribution of activations in fault-free and fault models becomes more pronounced. This phenomenon reveals that the errors are mostly amplified in the last layers of DNNs where it is crucial to harness them. This observation suggests that the output activations in the initial layers of DNNs can be restricted by layer-wise clipping thresholds and the last layer can be restrained by neuron-wise ones. As a solution, we introduce a hybrid clipped activation function that incorporates neuron-wise thresholds specifically for the last layer of DNNs and layer-wise thresholds for the rest of the layers, aiming to decrease memory overhead and enhance resilience.

5.2.2 Methodology: ProAct and HyReLU

In this subsection, we introduce ProAct, a progressive training approach for clipping activation function thresholds, implemented in a hybrid manner: neuron-wise exclusively in the last layer of DNNs and layer-wise in all other layers. Progressive training aims to minimize clipping thresholds for activation functions and enhance DNNs resilience while maintaining their baseline accuracy. Moreover, the hybrid clipped activation function mitigates memory overhead and reduces the occurrence of faults in memory locations.

The proposed hybrid clipped ReLU activation function incorporates neuron-specific thresholds for the neurons in the final layer of CNNs while employing layer-specific thresholds for the neurons in the preceding layers. In addition, to utilize the gradient-based optimization method, it is necessary to create a differentiable version of the activation function. To achieve these objectives, we introduce a hybrid clipped ReLU activation function (HyReLU), which guarantees smooth transitions around the threshold values (λ) utilized

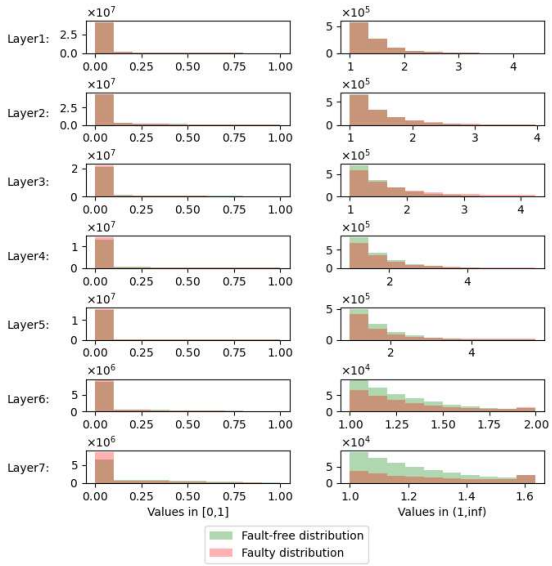


Figure 5.2: The distribution of output activation values for the AlexNet model on the CIFAR-10 dataset after applying the FitAct algorithm to find threshold parameters.

in the clipped ReLU in Eq. (5.1). In this equation, x is an input activation, λ is a trainable parameter representing the value of the clipping threshold in the respective neuron/layer and k is a hyper-parameter determining the slope for the smooth transition to 0, which is obtained through cross-validation. L indicates the last layer index.

$$HyReLU(x,\lambda,l) = \begin{cases} \max\{0, x_i[1-\sigma(k[\lambda_i - x_i])]\} \; if \; l = L \\ \max\{0, x[1-\sigma(k[\lambda - x])]\} \;\; otherwise \end{cases} \tag{5.1}$$

HyReLU is employed across all neurons of the last layer of DNNs (i.e., the layer preceding the output layer), with each having a distinct trained λ_i. For other layers, the function is applied separately, with each layer possessing its own trained λ. Consequently, the memory overhead introduced to a DNN with the HyReLU is formulated in Eq. (5.2).

$$Memory\ Overhead\ = \frac{\#Layers\ +\ \#Neurons_{last\ layer}}{\#Parameters_{DNN}} \tag{5.2}$$

To obtain the best clipping thresholds (λ) in HyReLU for each layer/neuron in a DNN, we propose ProAct, a layer-wise progressive training method exploiting knowledge distillation. Fig. 5.3 depicts an outline of the ProAct approach, where the purpose is to find an optimal λ for each HyReLU without breaching the maximum permitted accuracy drop and memory overhead.

ProAct trains the clipping threshold of each layer separately, from the last to the first layer. ProAct includes two main steps to find the threshold parameters: 1) Pre-processing, and 2) Progressive training. In the first step, we start by profiling the model on validation data to determine the initial values for the threshold parameters. Specifically, we initialize the threshold parameters with the maximum activation value observed by the corresponding layer/neuron on the validation dataset. Then, we replace all ReLU activation functions with the proposed HyReLU, using the initial threshold parameters (lines 1-8 in Algorithm 5.1). Within the progressive training step, we select layers from the last layer

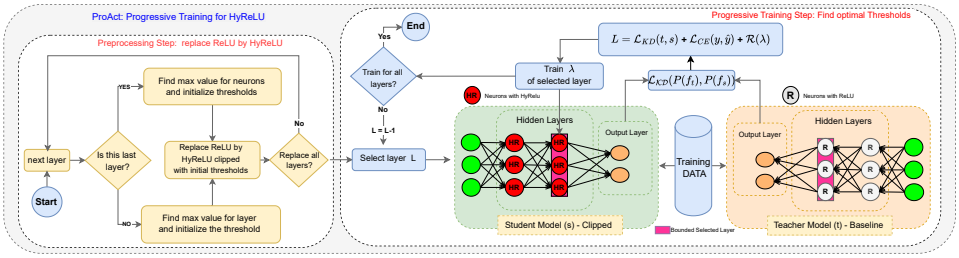


Figure 5.3: Hybrid Progressive training based on Knowledge Distillation.

to the first one and train the threshold parameters (λs) in the HyReLU of the selected layer through the KD-based training. The clipping threshold of the target layer is trained using KD and this process continues down to the first layer (lines 9-15 in Algorithm 5.1).

Algorithm 5.1 ProAct: Progressive Training for HyReLU Activation Function

Input: The unbounded teacher and bounded student models (t, s), learning rate (α), Regularization parameter (γ), number of epochs (N), BERs = $[10^{-6}, 3\times10^{-6}, 10^{-5}, 3\times10^{-5}, 10^{-4}]$;

Output: Resilience DNN;

Preprocessing Step

1: **for** $l \in [1, 2, \cdots, L]$ **do**:
2: **if** $l = L$ **then**:
3: Profile the model to find max value in each ***neuron***;
4: **else**:
5: Profile the model to find max value in each ***Layer***;
6: **end if**;
7: Initial the threshold parameters (λ) based on max values;
8: **end for**

Progressive Training Step

9: **for** $l \in [L, L-1, \cdots, 1]$ **do**:
10: **for** $i \leftarrow 1$ to N **do**:
11: Compute $\mathscr{L}$ (loss function) based on Eq. (5.4);
12: Compute $\partial\mathscr{L}/\partial\lambda$ where $\lambda : \nabla_\lambda \mathscr{L}(X, Y, \lambda)$;
13: Update λ via Adam optimizer;
14: **end for**;
15: **end for**;

The proposed training method utilizes KD in a way that the clipping thresholds in HyReLU are trained based on the supervision of the unbounded fault-free (baseline) model. Through this process, the purpose is to mimic the output values of the unbounded fault-free model in the modified model with the HyReLU activation function. The pre-trained baseline model including ReLU is used as the teacher model that includes the golden output values and the modified DNN is the student model that has the same structure as the teacher model, but ReLU is replaced by HyReLU. The loss function $\mathscr{L}_{KD}$ is computed based on the Kullback–Leibler divergence (KL) distance between the distribution of output values in the student and the teacher model as in Eq. (5.3) where $P(f(.,T))$ show the soft output logits with temperature parameter T.

Table 5.1: Baseline accuracy for each baseline CNNs.

CNNs	AlexNet	VGG-16	ReNet-50
Accuracy for CIFAR-10	81.67%	89.87%	91.11%
Accuracy for CIFAR-100	55.44%	65.45%	74.37%

$$\mathscr{L}_{KD}(X,s,t) = \mathbb{E}_{x\sim D} KL\Big(P(f_s(x,T,\lambda))||P(f_t(x),T)\Big) \tag{5.3}$$

Therefore, the whole loss function ($\mathscr{L}$) that we use to train the threshold parameters in the selected layer can be expressed as in Eq.(5.4), where the L_{ce} and $R(\lambda)$ show the cross entropy loss function and l_2-regularization. The regularization helps to constrain the magnitude of the threshold parameters, preventing them from becoming excessively large.

$$\mathscr{L}(X,Y,\lambda) = \mathscr{L}_{KD}(X,s,t) + \mathbb{E}_{x,y\sim D} L_{ce}(f_s(x,\lambda),y) + \gamma R(\lambda) \tag{5.4}$$

5.2.3 Experimental Setup

The proposed method ProAct is applied to and evaluated on three DNNs: AlexNet, VGG-16, and ResNet-50, all trained on both CIFAR-10 and CIFAR-100 datasets. Their baseline classification accuracy on the test sets is shown in Table 5.1. It is noteworthy that experiments in this work exploit 32-bit fixed point data type representation in which the Most Significant Bit (MSB) is for the sign, 15 bits are for the integer and 16 bits are for the fraction. All experiments in this section are performed on NVIDIA A4000-16GB GPU.

To demonstrate the excellence and effectiveness of ProAct with respect to the state-of-the-art, results are compared to Ranger [53], FT-ClipAct [113], and FitAct [93] methods. We implement Ranger in, both, layer-wise and neuron-wise manners and use a random small part of training data ($3,000$ out of $60,000$ in both CIFAR-10 and CIFAR-100 datasets) as the validation data to find the maximum values for the layers/neurons. FT-ClipAct is implemented layer-wise and FitAct is implemented neuron-wise, as they are presented in [113] and [93], respectively. All mentioned activation restriction methods are implemented in the PyTorch framework and their source code is published on a GitHub repository as a tool[1], to enable researchers for further achievements in this area.

To obtain a quantitative comparison with the existing works, we carry out the reliability assessment by injecting random bitflip faults into the parameters of CNNs, including weights, bias, and parameters of clipping activation functions as the fault space. Bits are randomly selected and flipped based on the 32-bit fixed-point data representation. We consider different BERs to flip multiple bits to model the accumulative effect of faults into the memory through time. The experimented BERs are $\{10^{-7}, 3\times10^{-7}, 10^{-6}, 3\times10^{-6}, 10^{-5}, 3\times10^{-5}\}$.

Fault injection experiments are repeated 500 times for each BER and average results for Top-1 accuracy are reported and compared. For the fault injection, we adopt and extend PyTorchFI [160] to consider clipping thresholds in the fault space, developed on top of PyTorch. The training iterations consist of 50 epochs for neuron-specific clipping thresholds in the final layer and 20 epochs for layer-wise HyReLUs. This ensures comparable computational overhead to FitAct, which employs 150 epochs. We initialize the learning rate at 0.01, halving it every 10 epochs, and utilize a batch size of 128.

[1] https://github.com/hamidmousavi0/reliable-relu-toolbox.git

5.2.4 Results: Overhead Reduction and Resilience Improvement

5.2.4.1 Effect of Activation Restriction Methods on DNNs' Baseline Accuracy- and Memory Footprint As mentioned, the clipping thresholds in any activation restriction method are obtained through validation data (not test data). On the other hand, the main requirement of applying them is that they are required not to drop the baseline accuracy of fault-free DNNs over unseen test data. Table 5.2 shows the impact of activation restriction methods on the baseline accuracy for each DNN after application. It is observed that:

- Ranger has the least effect on the accuracy drop compared to the other methods. Since Ranger considers the clipping threshold as the maximum value seen in validation data (either for neurons or layers), the obtained clipping thresholds are large enough not to affect the inference of the test data. As a result, Ranger reduces the baseline accuracy by less than 0.2%.

- FT-ClipAct introduces the largest accuracy drop through all methods, from 0.9% in AlexNet trained up to 4.68% in ResNet-50 both trained on CIFAR-10. Such an accuracy drop is significant for DNNs, especially in safety-critical applications, and decreases the effectiveness of the applied activation restriction method. Since the heuristic search algorithm is very complex and slow, it is exploited with a small subset of the training data (1,000 data in both CIFAR-10 and CIFAR-100). Therefore, the obtained thresholds are not optimal and influence the accuracy considerably.

- The accuracy drop induced by ProAct is less than 1% which is negligible. Moreover, ProAct reduces the baseline accuracy of DNNs less than both FT-clipAct and FitAct. This is due to the progressive training method, which ensures finding the optimal threshold for each layer separately without sacrificing accuracy.

Table 5.2: Accuracy drop of CNNs after applying different activation function restriction methods.

Activation restriction method	Ranger (layer-wise)	Ranger (neuron-wise)	FT-ClipAct (layer-wise)	FitAct (neuron-wise)	ProAct (hybrid)
AlexNet CIFAR-10	0.00%	0.00%	0.90%	0.78%	0.29%
AlexNet CIFAR-100	0.01%	0.03%	2.40%	0.64%	0.51%
VGG-16 CIFAR-10	0.00%	0.02%	1.15%	1.14%	1.00%
VGG-16 CIFAR-100	0.00%	0.07%	1.69%	0.83%	0.35%
ResNet-50 CIFAR-10	0.15%	0.19%	4.69%	0.30%	0.22%
ResNet-50 CIFAR-100	0.00%	0.08%	1.60%	0.03%	0.10%

The existing methods are either neuron-wise or layer-wise, which lay different memory overheads on CNNs. Layer-wise approaches introduce new clipping threshold parameters to DNNs proportional to the number of layers which is a negligible overhead. Whereas neuron-wise approaches lay a remarkable overhead as the number of neurons in CNNs is huge. However, the ProAct memory footprint is limited since it is a hybrid neuron-wise and layer-wise activation function.

Table 5.3 compares the induced memory overhead to baseline CNNs between neuron-wise, layer-wise, and the proposed hybrid approach activation restriction. It is observed that ProAct significantly reduces memory overhead compared to neuron-wise techniques (FitAct), ranging from 10.5 to 134.28 times while ensuring enhanced accuracy in CNNs against faults.

Table 5.3: Comparison of memory overhead for neuron-wise, layer-wise, and hybrid activation restriction methods.

Activation restriction method	neuron-wise	layer-wise	**Hybrid (ProAct)**
AlexNet CIFAR-10	0.29%	0.00003%	**0.017%**
AlexNet CIFAR-100	0.21%	0.000035%	**0.020%**
VGG-16 CIFAR-10	1.88%	0.0000884%	**0.014%**
VGG-16 CIFAR-100	0.85%	0.0000446%	**0.012%**
ResNet-50 CIFAR-10	12.23%	0.0002%	**0.134%**
ResNet-50 CIFAR-100	12.23%	0.0002%	**0.134%**

5.2.4.2 Resilience Comparison of Activation Restriction Methods Fig. 5.4 depict the Top-1 accuracy of DNNs leveraging different activation restriction methods on CIFAR-10 and CIFAR-100 under FI campaigns. The right column in both figures magnifies the results to highlight the impact of ProAct against the state-of-the-art methods, in particular FT-ClipAct and FitAct. It is observed that equipping DNNs with ProAct remarkably enhances the resilience of DNNs compared to the other state-of-the-art methods.

Regarding Fig. 5.4, at all BERs, the accuracy of DNNs with ProAct is higher than the DNNs with other activation restriction methods. As it is observed, Ranger provides the least resilient DNNs. According to the results, although FitAct provides better resilience than FT-ClipAct, it introduces a remarkable memory overhead orders of magnitude more than FT-ClipAct. Whereas ProAct achieves a higher resilience than all existing methods with negligible overhead.

Moreover, it is observed that in model-wise FI experiments, all activation restriction methods can effectively improve the resilience of DNNs compared to unprotected DNNs. However, they fail to provide highly resilient DNNs at high BERs. When faulty weights are spread throughout a DNN, several neurons in various layers are affected. Consequently, in a high BER, the values of affected neurons are restricted by their activation functions and make numerous erroneous activations propagate to the DNN output resulting in a considerable accuracy drop. However, ProAct surpasses the other activation restriction techniques in terms of providing superior accuracy for DNNs in model-wise FI.

Table 5.4 summarizes the results for accuracy drop of experimented DNNs with respect to their own baseline accuracy in Table 5.1, hardened by FT-ClipAct, FitAct and ProAct, at the BERs where the accuracy drop of ProActed CNNs for CIFAR-10 is less than 5%, and for CIFAR-100 is less than 10%. This comparison implicitly includes the accuracy drop due to activation restriction methods exhibiting the overall benefit of ProAct. According to the results, it is observed that ProAct reduces the accuracy drop of CNNs from 1.36 up to 6.4 times compared to FT-ClipAct, and from 1.07 up to 1.72 times compared to FitAct.

5.2.5 Discussion

This section introduced ProAct, a progressive training method for determining threshold values in a novel activation function i.e., HyReLU, aimed at enhancing the resilience of

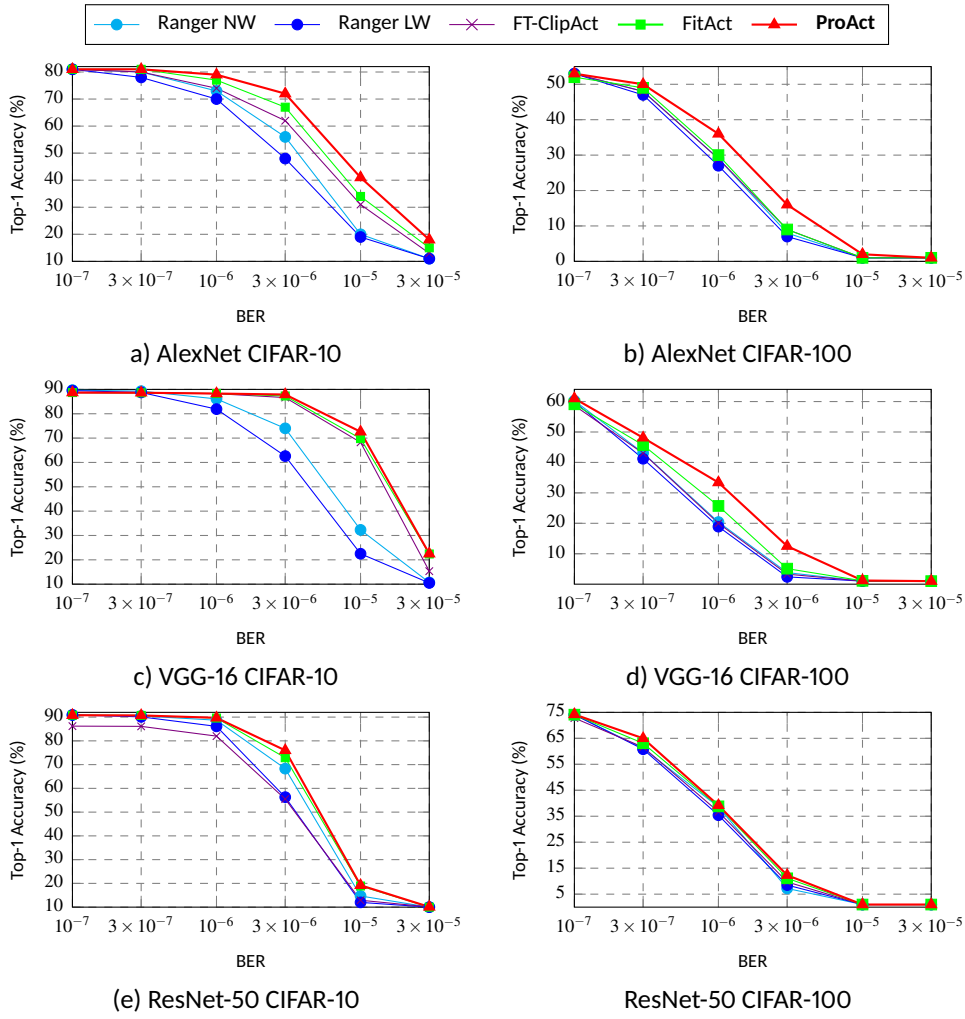


Figure 5.4: Top-1 accuracy comparison of DNNs using ProAct with Ranger neuron-wise (NW), Ranger layer-wise (LW), FT-ClipAct, and FitAct methods under FI.

DNNs. The proposed HyReLU reduces memory overhead by applying neuron-wise clipping solely in the last layer and layer-wise clipping in the preceding layers. Furthermore, we proposed a progressive training method utilizing KD to train the threshold parameters layer by layer, effectively identifying optimal threshold values for the HyReLU. Our experimental results indicate that ProAct significantly improves the resilience of DNNs, with enhancements of up to 6.4 times in high BERs. Furthermore, our approach dramatically reduces memory overhead, achieving reductions up to 134.28 times compared to a neuron-wise state-of-the-art method. Furthermore, we have published all source codes to enable researchers to present more effective approaches in this area.

5.3 Channel Duplication and Vulnerability-Aware Pruning

Activation restriction methods are effective in enhancing the resilience of CNNs, however, they do not conduct error correction, and CNNs fail to function at high BERs due to the

Table 5.4: Comparing the accuracy drop of DNNs using different activation restriction methods under fault injection.

DNNs	BER	FT-ClipAct	FitAct	**ProAct**
AlexNet CIFAR-10	1E-6	7.67%	4.34%	**2.52%**
VGG-16 CIFAR-10	3E-6	3.19%	2.61%	**1.88%**
ResNet-50 CIFAR-10	1E-6	9.09%	1.53%	**1.42%**
AlexNet CIFAR-100	3E-7	7.89%	6.31%	**5.28%**
VGG-16 CIFAR-100	1E-7	6.74%	6.34%	**4.40%**
ResNet-50 CIFAR-100	3E-7	12.75%	11.24%	**9.37%**

replacement of numerous neurons with 0. This section introduces a novel model-level hardening solution to modify the architecture of CNNs to allow fault correction at inference inherently. An efficient error correction mechanism is designed and enabled by selectively duplicated channels within the structure of CNNs. In the proposed method, the parameter vulnerability of CNNs is analyzed and the more vulnerable ones are duplicated. Thereafter, a correction layer detects and corrects the erroneous output activations based on the two duplicated values. To reduce the method's overhead, a strategy is proposed for channel pruning based on the vulnerability of parameters to effectively shrink the size of CNNs with a negligible accuracy loss.

5.3.1 CNN Model Hardening

5.3.1.1 Vulnerability Estimation Vulnerability estimation of CNN's parameters reflects how they affect classification outputs in the presence of faults. We adopt a vulnerability estimation approach introduced by [161] and adapt it to the parameters of a channel in a CNN. This approach is validated by FI in [161]. Eq. (5.5) describes the vulnerability estimation for each channel:

$$Vulnerability_{channel} = \sum_{i=1, i\neq t}^{C} \frac{\sum_{w \in channel} |\frac{\partial (Z_i - Z_t)}{\partial w}|^2}{|Z_i - Z_t|^2} \tag{5.5}$$

In Eq. (5.5), the *vulnerability of a channel* with multiple weights w in a convolutional (CONV) layer of a CNN with C number of classes is estimated for a single input data. The output logits of the network corresponding to each output class is Z_i and the top class's logit is Z_t. This equation represents the effect of each channel on the output logits as a vulnerability estimation and a higher value represents a higher vulnerability of the corresponding channel. A similar equation is applied to the weights corresponding to a neuron in Fully Connected (FC) layers.

5.3.1.2 CNN Model Hardening Method After obtaining the vulnerability of channels of a pre-trained CNN, the CNN model is hardened by performing two steps:

- Duplication of the more vulnerable channels,
- Insertion of the Error Detection and Correction (EDAC) layer after each CONV/FC.

Fig. 5.5 illustrates how the duplication of channels functions. A channel of parameters contains multiple weights for obtaining an OFMap F_k resulting from the summation of weighted inputs. In the lth CONV layer with C^l output channels, a channel is a 3-dimensional array of weights X^l, Y^l, C^l. In an FC layer, an output channel is a 1-dimensional

weight array corresponding to a neuron. Duplicating a channel of parameters generates duplicated values in F_k which provides an opportunity to detect and correct errors produced by faults in parameters. This method selects a ratio of more vulnerable channels with respect to Eq. (5.5) for duplication.

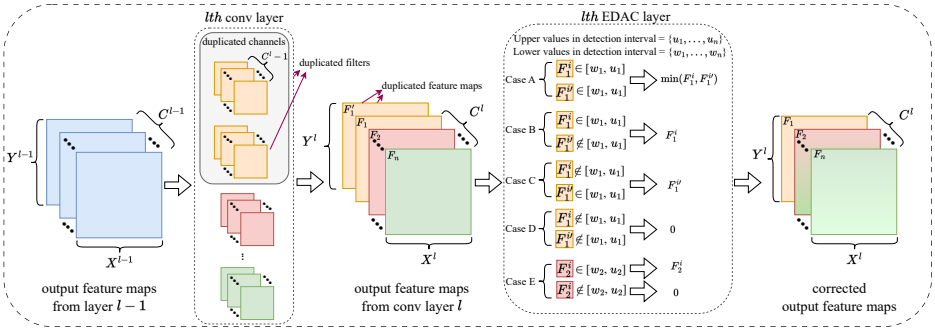


Figure 5.5: Channel duplication and EDAC layer.

After duplicating the vulnerable parameter channels, an EDAC layer is inserted into the CNN after each CONV and FC layer. The EDAC layer is meant to detect and correct errors in its incoming F_k from a CONV/FC layer within the CNN. One of the major challenges with 32-bit floating point data representation in general-purpose devices such as CPU and GPU is that faults may lead to overflows in CNNs producing Not-a-Number (NaN) values and corrupting the outputs. To address this issue, one of the primary operations in the EDAC layer is to replace any produced NaN value with 0 in the feature maps F_k.

Fig. 5.5 illustrates how the EDAC layer operates. The EDAC layer exploits a detection interval containing the minimum and maximum values in the channels of F_k that are the lower values $\{w_1, w_2, ..., w_n\}$ and the upper values $\{u_1, u_2, ..., u_n\}$, respectively. Detection intervals are obtained by profiling the CNN on the training dataset. It is assumed that the data distribution of training is representative enough to provide generic and valid detection intervals for the unseen data during the inference [99].

The EDAC layer is aware of duplicated and non-duplicated channels. In the duplicated channels, an error is detected and corrected in two cases:

- Both duplicated values of an OFMap in the corresponding channels are in the detection interval but are not equal. In this case, the minimum value between them is selected as the correct output F_k^i (case A in Fig. 5.5). The reason behind this correction is that CNNs are more resilient to small numbers [113].

- A value in a channel exceeds the detection interval, thus, the duplicated value that is in the detection interval is the correct value for the output F_k^i (case B and C in Fig. 5.5). If both duplicated values are not in the detection interval, the output F_k^i is set to 0 (case D in Fig. 5.5).

In the non-duplicated channels, faults are detected and corrected based on the detection intervals. If any value in the channel exceeds the corresponding detection interval output F_k^i sets to 0 (case E in Fig. 5.5). The rationale behind zeroing is that it eliminates the propagation of erroneous values within a DNN. Note, that the detection and correction are repeated for each element of the two-dimensional array of the OFMap F_k. Fig. 5.6

depicts an example of the operation of the EDAC layer. The red values in this figure are erroneous and green values are fault-free.

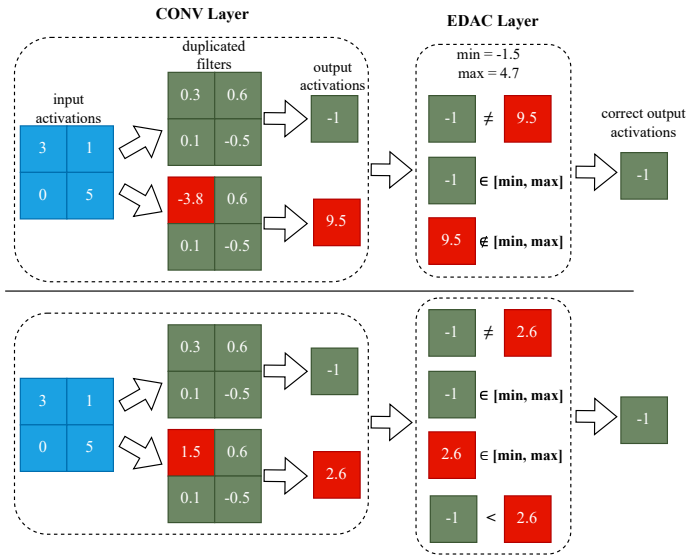


Figure 5.6: An example for the operation of EDAC layer operation.

To prevent faults from any immediate misclassification at the last layer, all output channels of the last layer in CNNs (i.e., neurons in the last FC layer) are duplicated and protected by an EDAC layer. It is worth mentioning that EDAC is implemented in a highly parallel way in Pytorch so that it can operate detection and correction on all duplicated and non-duplicated channels in parallel. Moreover, the hardened CNNs have the same accuracy as the baseline ones.

5.3.2 Experimental Setup

The parameters of a pre-trained CNN could be faulty at inference time due to several reasons, including soft errors, temperature or voltage variation, process variation, aging, etc. To examine the resilience of CNNs, we model faults in the parameters by flipping their bits considering different Bit Error Rates (BERs). To this end, any layer in the CNN's parameters, including convolutional, Fully Connected (FC), batch normalization, and EDAC layers is subject to a fault injection campaign. We have developed the fault injection on top of Pytorch, and the data representation is IEEE-754 32-bit floating point. The number of bitflips in a layer is equal to $BER \times \#parameters \times 32$ in that layer. The fault injection simulations are performed on an NVIDIA 3090 GPU and any fault injection experiment is repeated 1000 times and the average accuracy drop is reported as the resilience metric. The experimented BERs are 10^{-8}, 5×10^{-8}, 10^{-7}, 5×10^{-7}, 10^{-6}, 5×10^{-6}, 10^{-5}, 5×10^{-5}, and 10^{-4}.

The experiments in this work are performed on three deep CNNs: AlexNet and VGG-11 trained on CIFAR-10 and VGG-16 trained on CIFAR-100. Their baseline accuracy as well as the number of parameters and MAC operations are reported in Table 5.5. The performance in terms of execution time of the CNNs over their test set is examined on an

NVIDIA 3090 GPU coupled with an *AMD Threadripper 3960X* 24-core processor. Note that the accuracy of unprotected CNNs decreases drastically even at relatively low BERs. The unprotected AlexNet drops 26% at $BER = 5 \times 10^{-7}$ and the accuracy of unprotected VGG-11 and VGG16 drops 24.07% and 31.17% at $BER = 5 \times 10^{-8}$, respectively.

Table 5.5: The baseline CNNs leveraged in this section.

CNN	Dataset	Base accuracy	#parameters	#MACs	Performance (sec)
AlexNet	Cifar-10	73.15%	21,623,562	42,316,288	0.591
VGG-11	Cifar-10	92.85%	9,228,362	153,293,824	0.655
VGG-16	Cifar-100	73.20%	34,015,396	332,756,992	0.782

5.3.3 Results

5.3.3.1 Hardening by Channel Duplication vs. Triplication First, we demonstrate how EDAC performs if the detection intervals are not exploited for non-duplicated channels and compare it with a triplication-based correction performed by a voter. The voter takes three replicated OFMaps in the corresponding channel and outputs the most repeated value. In the case where all three OFMaps are different (if at least two replicated filters are faulty), the voter outputs the minimum value.

Fig. 5.7 presents the results for accuracy drop and memory overhead of *duplication + EDAC* vs. *triplication + voter* for AlexNet at $BER = 10^{-4}$ over different channel hardening ratios. A similar trend is observed for VGG-11 and VGG-16. The highlights that can be observed from the Figure are:

- *Duplication + EDAC* achieves a similar resilience to that of *triplication + voter* in terms of accuracy drop, with twice less memory overhead.
- The memory overhead is proportional to the channel duplication and triplication ratio. The memory overhead of the EDAC layer is negligible compared to the total memory and computational requirements of CNNs.
- A high resilience is achieved only at full channel hardening. At lower hardening ratios, although the more vulnerable channels are protected, the unprotected channels incur a high accuracy drop in CNNs due to the high BER.

As observed, we need to apply a full channel duplication + EDAC to protect CNNs which leads to a significant overhead in the hardened CNNs compared to the unprotected ones. The hardened CNNs have double memory and computational requirements (100% overhead) and the execution time increases up to 1.83 times. To tackle this issue, we exploit the detection intervals in the non-duplicated channels to protect less vulnerable channels which leads to lower hardening ratios. It is presented in the next subsection.

5.3.3.2 Hardening by Selective Channels Duplication and EDAC Layer This subsection presents the results for the selective channel duplication with EDAC layers. In a pre-trained DNN, a ratio of the more vulnerable channels are duplicated and both duplicated and non-duplicated channels exploit detection intervals to be hardened at the EDAC layer. Since the hardening method is at the model level, the performance in terms of execution time is influenced. We present the performance overhead on *NVIDIA 3090 GPU*.

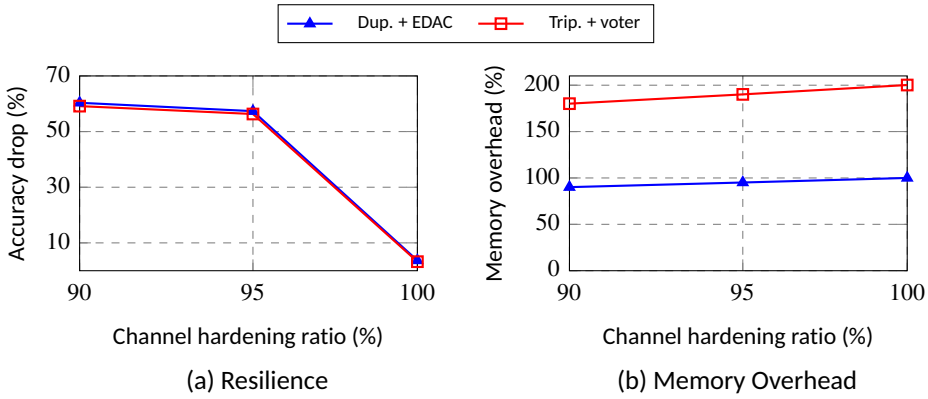


Figure 5.7: Resilience (a) and memory overhead (b) for AlexNet hardened by duplication + EDAC vs. triplication + voter at BER=10^{-4}, without applying detection intervals to Non-duplicated channels.

Fig. 5.8 demonstrates the resilience and performance overhead for all the experimented CNNs at the highest BERs where the accuracy drop is not yet significant (lower than 5%). As observed, exploiting detection intervals in unprotected channels has a remarkable effect on reducing the hardening ratio to achieve high resilience. It can be observed that by merely using detection intervals without channel duplication (*hardening ratio = 0%*), the accuracy drops at high BERs are lower than 4% with up to 11.4% performance overhead compared to the unprotected baseline CNNs. Nonetheless, increasing the channel hardening ratio improves the resilience without a significant performance overhead. With a 15% channel hardening ratio the accuracy drop improves 17%, 38%, and 24% for AlexNet, VGG-11, and VGG-16 respectively, achieved by 6.7% to 12% longer execution time, compared to 0% hardening ratio.

As noted, EDAC layers exploiting detection intervals for all channels can significantly reduce the overhead of the hardened CNNs compared to the full duplication. However, a tangible overhead is laid upon CNNs due to hardening. The overheads are caused by both channel duplication and EDAC layer operations. To tackle this issue, we deploy a pruning method to reduce the size of baseline CNNs by removing the least vulnerable channels and applying EDAC to the most vulnerable ones, so the total overhead can be further reduced. This method is presented in the next section.

5.3.4 Overhead Reduction by Pruning based on Parameters' Vulnerability

As observed, although the introduced hardening technique exhibits a high resilience to CNNs, it lays a considerable overhead to them. To address this issue, we apply an effective structured channel pruning to CNNs to shrink their baseline size and then apply the hardening mechanism.

5.3.4.1 Vulnerability-Aware Pruning Structured pruning is a well-known method for CNN models to reduce their size leading to optimizing their performance and resource utilization. In this method, a metric for the significance of the effect of parameters on the output accuracy is considered and the least important weights are removed from the CNN with a negligible accuracy loss.

Conventionally, the significance of the weights effect is examined by L1-norm which is shown to be effective [106]. In this work, we exploit Eq. (5.5) as the importance metric for

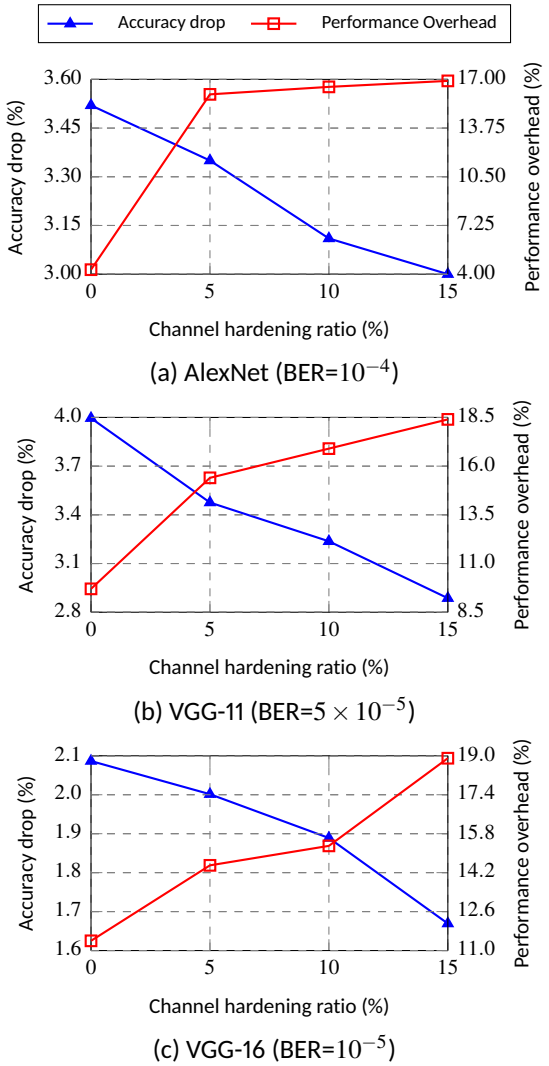


Figure 5.8: Accuracy drop and performance overhead comparison for hardened CNNs over different channel pruning ratios at the BERs where accuracy drop is below 5%.

channel parameters and remove a ratio of the least vulnerable channels from CONV and FC layers in CNNs. To avoid losing too much accuracy, we perform lightweight training on the pruned CNNs with 10 epochs using SGD with a learning rate of 0.001 on the training dataset. Fig. 5.9 shows that our vulnerability-aware pruning method is more effective than L1-norm pruning in terms of removing the channels of CNNs while the accuracy is still close to that of the baseline CNN.

To obtain the highest possible pruning ratios for each CNN, we perform an extensive exploration over different pruning ratios of CONV and FC layers to minimize the number of parameters and MAC operations maintaining the test accuracy within 1% of its unprotected baseline. Table 5.6 shows the selected pruning ratios for the experimented CNNs and their improved memory and computational requirements compared to the baseline ones. As it is observed, the pruned CNNs achieve from 1.18 to 6.19 times fewer parame-

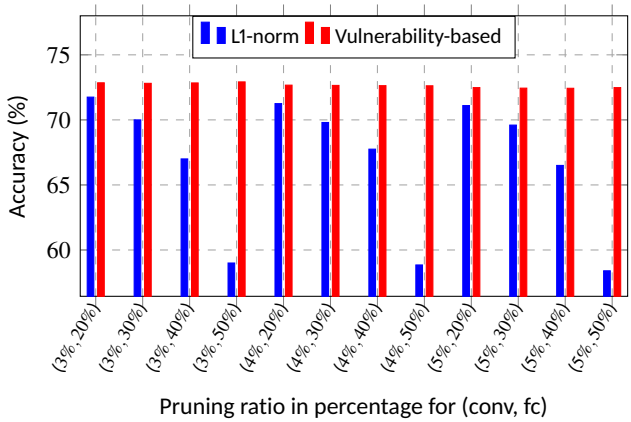


Figure 5.9: Comparison of L1-norm pruning and vulnerability-based pruning in AlexNet.

ters, 1.03 to 2.06 times fewer MAC operations, and 1% to 11.1% less execution time than the baseline ones.

Table 5.6: Pruned CNNs specifications.

CNNs	Conv. prun. ratio	FC prun. ratio	Pruned CNN Accuracy	Norm. #params to baseline	Norm. #MACs to baseline	Norm. perf. to baseline
AlexNet	5%	80%	72.38%	0.1615	0.4851	0.888
VGG-11	4%	35%	91.96%	0.847	0.9059	0.987
VGG-16	1%	15%	72.4%	0.826	0.9665	0.998

5.3.4.2 Resilience and Overhead of the Hardened Pruned CNNs By shrinking the baseline CNNs using pruning, the overhead of hardened CNNs is reduced compared to the baseline ones. To that end, the pruned CNNs' channel vulnerability is obtained, the more vulnerable channels are duplicated, and EDAC layers are implanted into the model with the corresponding detection intervals. Fig. 5.10 illustrates how resilience is improved in the hardened pruned CNNs against hardened baseline ones over different BERs, with 15% channel hardening ratio. It is observed that the proposed pruning not only reduces the overhead of hardened CNNs but also improves their resilience.

Fig. 5.11 compares the performance overhead in terms of the execution time of different hardened CNNs on *NVIDIA 3090 GPU*. As observed, the overhead of *triplication + voter* is significantly higher than the other methods. On the other hand, hardened pruned CNNs have the best performance among the hardened CNNs. Throughout the results, the performance of 15% hardened pruned Alexnet, VGG-11, and VGG-16 is improved by 24%, 1%, and 4.7%, respectively, compared to the 15% hardened ones without pruning. Noteworthy, the hardened pruned AlexNet has 6.06% less execution time than its unprotected baseline. The selective hardened pruned AlexNet, VGG-11, and VGG-16 require 81.40%, 2.67%, and 3.98% less memory, respectively, than their unprotected baseline to store their parameters.

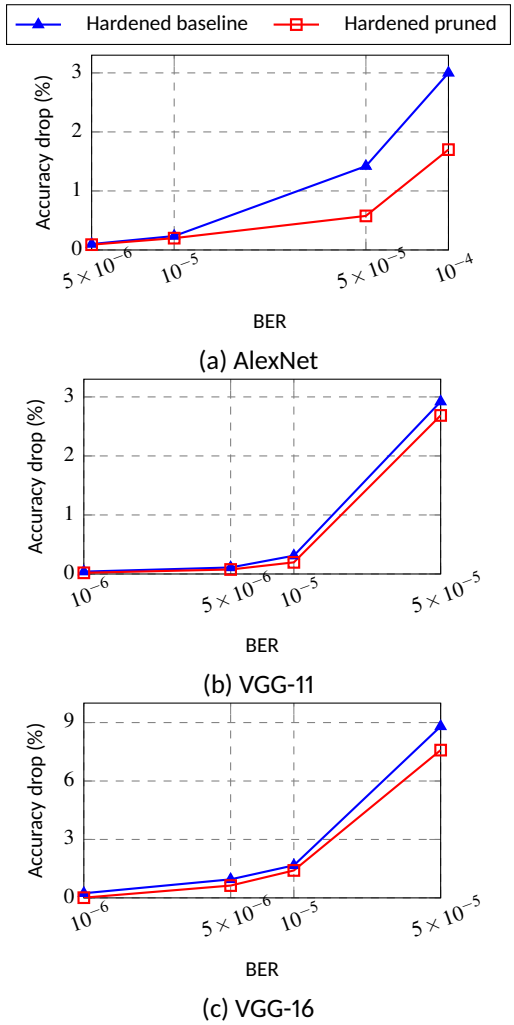


Figure 5.10: Resilience comparison in terms of accuracy drop of hardened baseline and hardened pruned CNNs at different BERs with 15% channel hardening ratio.

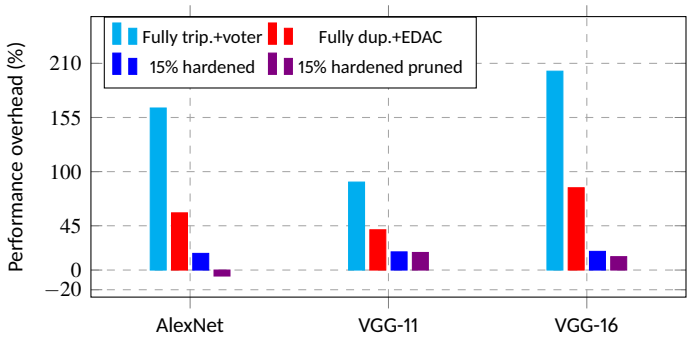


Figure 5.11: Performance overhead comparison for hardened CNNs.

5.3.5 Discussion

This section presents a model-level hardening method for CNNs by selective channel duplication and EDAC layers. The proposed method enables CNNs to detect and correct faults inherently, at inference time. The hardened CNNs perform reliably at orders of magnitude higher error rates than unprotected CNNs with merely a 15% hardening ratio, yet incurring 12% performance overhead. To further minimize the incurred overhead by the hardening method, for the first time, a vulnerability-based pruning that improves resilience is presented. As a result, the hardened pruned CNNs achieve up to 24% higher performance than the un-pruned hardened CNNs.

5.4 SentinelNN: Model-Level CNN Hardening Framework

Based on the findings in the current and previous Chapters, an open-source framework is developed for model-level CNN hardening, named SentinelNN. This framework employs DeepVigor+ for a fast and accurate resilience analysis to identify the more vulnerable channels and conducts selective channel duplication and correction. Furthermore, it employs activation restriction (Ranger) through ReLU to constrain the generation of large values generated as a result of hardware faults.

Fig. 5.12 illustrates the workflow of this open-source framework. As depicted, the framework receives a pre-trained CNN model and the dataset and performs DeepVigor+ to derive vulnerability factors for all channels in convolutional layers. Based on the obtained vulnerability, the less vulnerable channels in each layer are structurally pruned based on the user-specified *channel pruning ratio*. Afterward, the pruned CNN is analyzed by DeepVigor+ to provide the channels' vulnerability of the pruned CNN. Thereafter, based on a user-specified *channel hardening ratio*, the higher-vulnerable channels are duplicated, and a correction function similar to the proposed method in Fig. 5.5 is applied to them. It is noteworthy that SentinelNN does not incorporate detection intervals, and instead, employs a layer-wise Ranger for ReLU throughout the CNN.

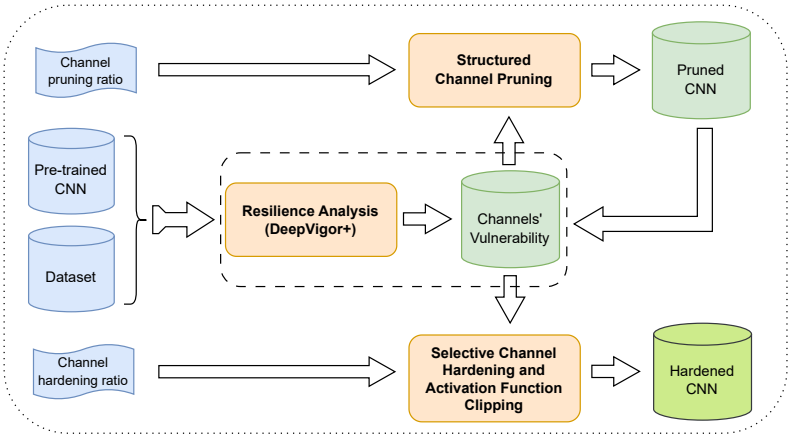


Figure 5.12: SentinelNN framework for CNN models hardening.

5.4.1 Experimental Setup

SentinelNN is fully implemented using Python and the Pytorch library. Its source code is published in https://github.com/mhahmadilivany/SentinelNN as a convenient

and scalable tool for researchers and engineers to adopt and build upon. The user can separately specify the pre-trained model and the corresponding ratio for channel pruning and hardening. This framework supports various architectures of CNNs (e.g., VGG and ResNet) and includes multiple resilience analysis methods including DeepVigor+, Vulnerability Gain [161] i.e., Eq. (5.5), and LRP for weights [7]. These methods can be employed for pruning and hardening CNNs to explore their efficacy.

To briefly present the outputs and effectiveness of the SentinelNN framework, we perform some experiments on two CNNs on CIFA-100: VGG-19 on CIFAR-100 with an accuracy of 73.87%, and ResNet-20 on ImageNet with an accuracy of 68.83%. The CNNs are pruned and hardened with different ratios and their overhead to the number of parameters and MAC operations are reported. Moreover, to show the effectiveness of the hardening method, we perform FI into their weights with various BERs and measure their accuracy loss. We experiment CNNs with a batch size of 256 images, pruning ratio of 5% and 10%, and hardening ratio of 10% and 20%. DeepVigor+ is conducted for resilience analysis. FI campaigns are repeated $1,000$ times and the average accuracy drop is reported.

5.4.2 Experimental Results

As mentioned, SentinelNN employs pruning to reduce the size of DNNs and conduct hardening for selective channel duplication, producing fault-tolerant CNN models. Table 5.7 presents the results for pruning and hardening CNNs using DeepVigor+ for weight channel vulnerability analysis. These results demonstrate that we can obtain hardened CNNs with the same or smaller size than the baseline ones, considering the number of parameters and MAC operations.

Table 5.7: Comparing the pruned CNNs specifications.

CNNs	Channel Pruning Ratio	Pruned CNN Accuracy	Channel hardening ratio	Normalized #params to baseline	Normalized #MACs to baseline
VGG-19	5%	70%	10%	0.994	0.996
			20%	1.082	1.086
	10%	69.24%	10%	0.894	0.894
			20%	0.815	0.815
ResNet-20	5%	64.88%	10%	1.007	1.033
			20%	1.096	1.136
	10%	63.5%	10%	0.892	0.906
			20%	0.977	1.003

Fig. 5.13 presents the resilience of hardened CNN under the FI campaign with different BERs. It can be observed that the fault resilience of hardened CNNs is remarkably higher than that of baseline CNNs, with the same or smaller size. The accuracy drop for the unprotected baseline VGG-19 is 70% at $BER = 5 \times 10^{-7}$ and for ReNset-20 is 50% at $BER = 10^{-5}$. Whereas the hardened CNNs produced by SentinelNN have a remarkably lower accuracy drop at high BERs, according to Fig. 5.13. Furthermore, a higher hardening ratio at a same pruning ratio leads to a more fault-tolerant CNN. SentinelNN allows the user to control the size of CNNs based on the application requirement concerning accuracy and performance while achieving high fault tolerance.

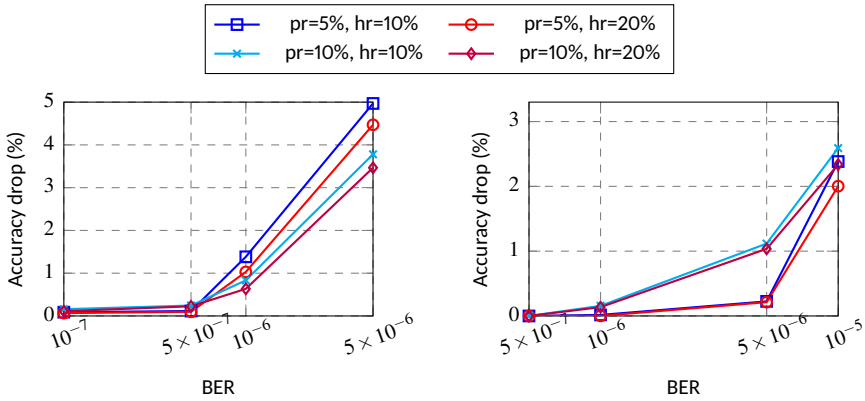


Figure 5.13: Resilience comparison in terms of accuracy drop for pruned and hardened CNNs at different BERs (pr = pruning ratio, hr = hardening ratio).

5.5 Chapter Conclusions

This Chapter addresses the identified gaps in the literature concerning model-level techniques to enhance the resilience of CNNs based on SIHFT-based methods. The proposed approaches enable reliable deployment of CNNs on any accelerator, ranging from general-purpose computing devices such as GPUs and CPUs to ASICs and pre-designed IPs, addressing RQ3.1 in Chapter 1. The proposed method attempted to outperform the state-of-the-art methods in terms of both fault resilience and computational/memory overhead of emerging CNNs and accelerators, addressing RQ3.2 and RQ3.3 in Chapter 1

We proposed HyReLU, a novel hybrid layer-wise and neuron-wise clipping activation function that is trained progressively by ProAct to identify the optimal values for clipping thresholds. It is shown that this method is more effective in enhancing the fault resilience of CNNs than the best state-of-the-art method while reducing its memory overhead remarkably. Furthermore, we introduced a novel and effective approach for hardening CNNs against faults and enable to correct errors inherently. On the other hand, vulnerability-aware pruning is proposed to enable the management of overheads vs. fault resilience of CNNs. Eventually, this Chapter introduces the open-source SentinelNN framework that integrates vulnerability-aware pruning using DeepVigor+ and hardening (channel duplication and range restriction) to produce highly fault-resilient CNNs. The proposed approaches in this Chapter consider the trade-off between performance and reliability and attempt to provide more resilience for CNNs with lower costs than the state-of-the-art.

6 Reliability Assessment and Enhancement for LSTMs

Given the safety-critical nature of medical and healthcare domains, the hardware reliability of DL devices is of paramount importance. Healthcare applications exploit Deep Neural Networks (DNNs) extensively for various tasks such as diagnosis, treatment, and prediction of diseases and anomalies [109, 165] because of their outstanding strength in processing time-series data [151, 223].

As mentioned in Chapter 3, nearly all existing works study the reliability of Convolutional Neural Networks (CNNs) for image classification and object detection tasks. Although LSTMs are widely deployed in safety-critical applications, including healthcare, their reliability against faults and fault tolerance methods are not extensively explored.

In this Chapter, we conduct resilience assessment and enhancement for LSTMs in different medical applications: gait analysis and disease prediction. This Chapter attempts to address P4 which includes RQs 4.1 and 4.2 and presents contributions mentioned in C4, in Chapter 1. This Chapter is based on the following publications:

I M. H. Ahmadilivani, J. Raik, M. Daneshtalab, and A. Kuusik. Analysis and Improvement of Resilience for Long Short-Term Memory Neural Networks. In *IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, pages 1–4. Juan-Les-Pennes, France, 2023.

II B. Parchekani, S. Nazari, M. H. Ahmadilivani, A. Azarpeyvand, J. Raik, T. Ghasempouri, and M. Daneshtalab. Zero-Memory-Overhead Clipping-Based Fault Tolerance for LSTM Deep Neural Networks. In *IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, pages 1–4. Oxforshire, United Kingdom, 2024.

In the rest of the Chapter, Section 6.1 investigates the resilience of multiple LSTM-based neural networks for gait analysis and applies a lightweight method to enhance their fault resilience. Section 6.2 studies the fault resilience of convolutional LSTM DNNs and applies some zero-overhead techniques to them to enhance their reliability, and Section 6.3 concludes the Chapter.

6.1 LSTM-based NN for Gait Analysis

Gait analysis as an essential diagnostic tool in various medical fields, including orthopedics, neurology, wellness assessment, and rehabilitation, benefits greatly from the utilization of LSTMs for processing time-series data typically collected by wearable Micromechanical Motion Sensors (MEMS) [60, 109, 180, 221].

In this work, we investigate the resilience of different LSTM-based NNs to detect gait abnormalities in time-series gait data and study their resilience using Fault Injection (FI) into the weights. Moreover, by analyzing the distribution of weight values and assessing the resilience of the NNs, we propose a method to enhance their resilience.

6.1.1 Proposed Method: Resilience Assessment and Enhancement

6.1.1.1 LSTMs Under Study Three LSTMs are developed based on the architecture illustrated in Fig. 6.1 for gaits abnormality detection. Each LSTM has two output classes representing normal and abnormal steps. The examined LSTMs are called ANN-*n* where *n* represents the number of cells in the LSTM layer and also the number of neurons in the first FC layer. The designed LSTMs and their achieved results are fully presented and discussed in *paper XVI*, which is out of the scope of this thesis.

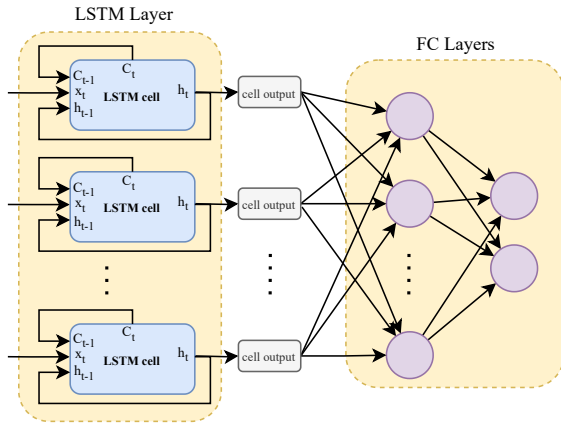


Figure 6.1: LSTM-based ANNs adopted in this work for gait abnormality detection.

6.1.1.2 Resilience Assessment by Fault Injection. Resilience assessment is performed by FI into different sets of weights. Faults are assumed to occur in memory that impact the weight parameters of ANNs. To represent this phenomenon, we consider different Bit Error Rates (BERs) to model accumulated bitflips throughout the parameters. Notably, faults in bias are not considered as their effect was negligible in our preliminary experiments. In our FI experiments, we flip random bits across the weights considering different BERs. We perform a random FI into the weights at each BER before a full inference. Then we obtain the outputs of the network for further resilience analysis and repeat the experiments several times in order to reach high-confidence results.

To perform both coarse-grain and fine-grain resilience analyses over the weights, we inject faults into randomly selected weights considering different scenarios, as follows:

- **Model-wise FI:** Injecting faults into randomly selected weights across the ANN,
- **LSTM FI:** Injecting faults only into randomly selected weights of the LSTM layer,
- **FC FI:** Injecting faults only into randomly selected weights of FC layers,
- **Recursive weights (W) FI:** Injecting faults only into randomly selected weights of the LSTM layer that involve the recursive inputs of cells (vector W in Eq. (2.11)),
- **Input weights (U) FI:** Injecting faults only into randomly selected weights of LSTM layer that involve the time-series inputs (vector U in Eq. (2.11)).

For each FI experiment, the faulty outputs of the last layer in ANNs are stored and compared with the ones in a fault-free execution. We present different evaluation metrics for resilience analysis in this work as follows:

- **Accuracy loss:** The difference between the accuracy of fault-free and faulty execution of the ANNs,
- **F1-score Loss:** The difference between the F1-score of fault-free and faulty execution of the ANNs.

We repeat FI experiments with different sets of random weights several times to achieve high-confidence results. Therefore, the average accuracy and F1-score over all the FI experiments are considered. Furthermore, we analyze the effect of faults on the output values to classify them. In each FI experiment, the outputs are classified into one of the following classes:

- **Masked:** The outputs in faulty and fault-free execution are the same,
- **Non-critical Silent Data Corruption (SDC):** The output values are different in faulty and fault-free execution but the classification result is the same,
- **Critical SDC:** The output values and classification are different in faulty and fault-free execution,
- **Detected Unrecoverable Error (DUE):** ANN produces 'NaN' values in the output considered as *system exception*.

6.1.1.3 Resilience Enhancement: Weights Online Checking and Correction To improve the resilience of LSTMs, first, we profile the weight values in the ANNs to observe the pattern of their distribution. This would lead us to the existing value ranges in the weights and find the most frequent values. Fig. 6.2 depicts the value distribution of weights across each network separately. It is observed that in all ANNs the range of weights is limited to some exclusive values. In addition, the values of most of the weights are close to 0.

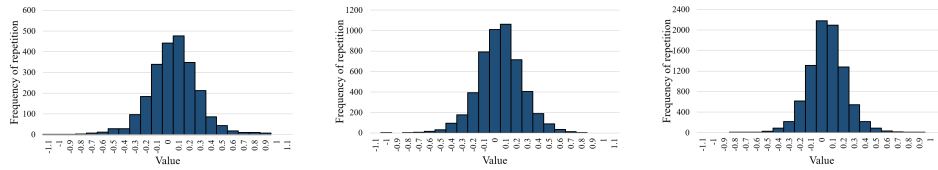


Figure 6.2: The distribution of values of weights for a) ANN-20, b) ANN-30, c) ANN-40, respectively.

For resilience improvement, two modes are considered; 1) offline mode, 2) deployment. In the offline mode, the minimum and maximum values for an ANN are obtained by profiling its weights. Thereafter, in the deployment, to detect and correct the faulty weights before an inference, we perform these operations iteratively:

- Comparing the value of all weights with the obtained range from the profiling,
- Replacing the exceeded weights with 0.

Using these operations allows us to find and remove the faulty weights that are larger than the expected values in the weights at run-time. The application specifies the frequency of iterating over the correction procedure.

6.1.2 Experimental Setup

The LSTMs that are examined in this section, detect abnormal gaits in a time-series dataset obtained from clinical experiments on 15 patients including their normal and abnormal steps. Table 6.1 presents the accuracy and F1-score as well as the number of parameters in LSTM-based ANNs. *It is strictly determined by the application that the accuracy of an*

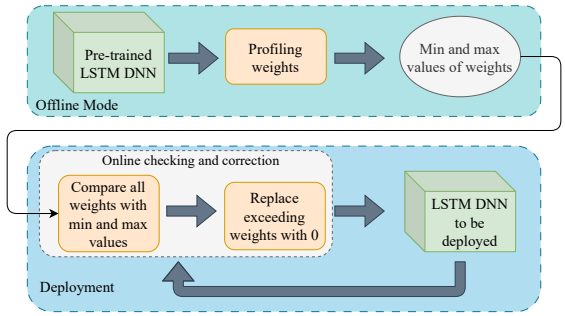


Figure 6.3: The proposed protection mechanism for LSTMs.

ANN should be higher than 80% *and its F1-score should be more than* 70%. Also, ANNs are employed by a general-purpose computer that supports 32-bit floating-point IEEE-754 data representation.

Table 6.1: Accuracy, F1-score, and the number of different weight sets of the LSTM-based ANNs.

	Accuracy	F1-score	#weights in U	#weights in W	#weights in FC
ANN-20	93.35%	80.33%	320	1,600	440
ANN-30	94.73%	85.39%	480	3,600	960
ANN-40	95.79%	88.57%	640	6,400	1,680

As mentioned, various FI scenarios are carried out over different sets of weights in the ANNs. The number of injected faults is determined by the Bit Error Rates (BERs) between 0.000015 to 0.006 to observe the full possible spectrum of the pieces of evidence. To achieve high-confidence results through random FI in terms of statistics, we repeat each FI experiment 2,000 times and report the average accuracy and F1-score. For the fault classification results, we save the outputs of each FI campaign and label them as *Masked*, *Non-critical SDC*, *Critical SDC* and *DUE*, and in the end, the rate of each fault class over the 2,000 experiments is reported. All experiments are implemented in PyTorch and are run on an *A100 NVIDIA GPU* along with an *AMD EPYC 64-core CPU*.

6.1.3 Experimental Results

6.1.3.1 Model-wise Resilience Analysis. The results of average accuracy loss and f1-score loss through a model-wise FI are reported in Fig. 6.4. It shows how the accuracy and F1-score of ANNs decrease together with the increase in the error rate. Regarding the mentioned accuracy constraints by the application (accuracy should be more than 80% and F1-score more than 70%), all networks fail to function at the BERs higher than 1.5×10^{-4}.

Assuming that only 0.015% of the weights across the model are faulty the average accuracy of ANN-20, ANN-30, and ANN-40 decreases by 4.29%, 7.26%, and 11.86%, respectively. Moreover, the F1-score of the ANNs at the mentioned BER is reduced by 7.12%, 13.11%, and 19.02%, respectively. These findings indicate a general trend: larger ANNs exhibit lower resilience compared to the smaller ones. This observation can be attributed to the fact that larger neural networks possess a greater number of parameters, making them more susceptible to the corrupting effects of faults.

In addition, faults are classified according to their effect on the outputs of ANNs in Fig. 6.5. It is observed that at low BERs, most of the faults are either masked or non-critical.

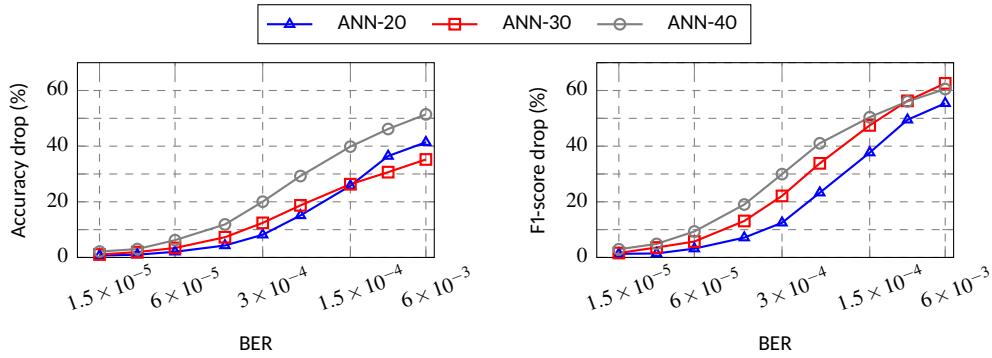


Figure 6.4: Resilience analysis of LSTM-based ANNs under model-wise FI.

However, by increasing the frequency of faults, networks are no more resilient. Especially when BER is 1.5×10^{-5} or more, where nearly all the faults are propagated to the output as SDC. At $BER = 1.5 \times 10^{-5}$, critical SDC for ANN-20, ANN-30, and ANN-40 is 5.44%, 8.79%, and 13.69% and the rest of the faults are mostly non-critical SDC (less than 0.1% are masked). It can be observed that DUE appears only when BER is very high.

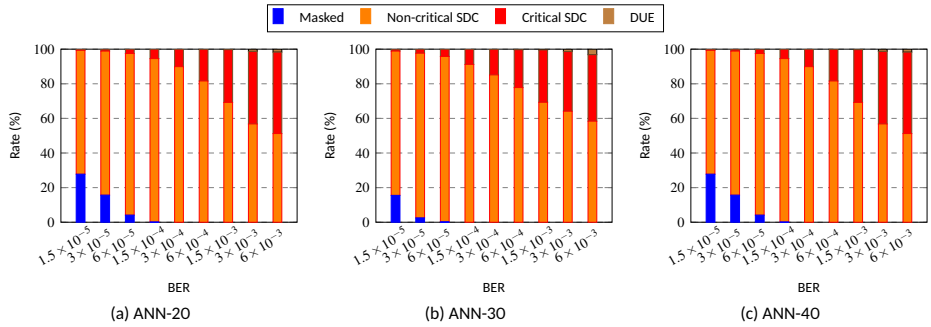


Figure 6.5: Fault classification in model-wise FI for LSTM-based ANNs.

6.1.3.2 Layer-wise Resilience Analysis. Fig. 6.6 presents the accuracy drop of ANNs when weights in the layers are faulty separately. Based on the results, it is observed that for BERs below 1.5×10^{-4}, the resilience of both, LSTM and FC layers are approximately similar, i.e., the variation in accuracy drop resulting from FI into these layers separately is minimal, with a difference of less than 0.5%. On the other hand, when BER is 1.5×10^{-4} and higher, in nearly all cases, the accuracy drop of ANNs in LSTM FI is remarkably higher. Therefore, the main observation in these results is that in all ANNs, the weights of the LSTM layer are more vulnerable than the ones in the FC layers, especially at high BERs.

Fault classification is also conducted for layer-wise FI. Analyzing its results provides the following observations:

- FC layers outperform the LSTM layer in fault-masking, also evidenced by the accuracy loss results. This observation can be attributed to the dense connections and the fault-masking capability of the ReLU activation function in the FC layers.

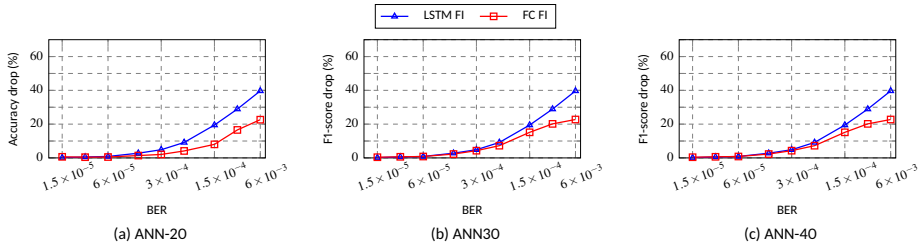


Figure 6.6: Comparison of ANNs' accuracy drop in layer-wise FI (LSTM vs FC).

- Faulty weights in an LSTM layer lead to more critical cases in all ANNs. Given the LSTM layer's role in memorizing input patterns, faults in its parameters can significantly influence ANNs resulting in increased misclassifications.
- In LSTM FI, the DUE rate is consistently 0, whereas in FC FI, DUE occurs at high BERs. This is caused by sigmoid and tanh activation functions within LSTM cells, which effectively eliminate large values caused by faulty weights. In contrast, FC layers employ the ReLU activation function, allowing the propagation of large positive values, leading to DUE occurrences.

6.1.3.3 Inter-LSTM Resilience Analysis: FI into *U* vs *W* As observed, LSTM layers are less resilient than FC layers. Accordingly, we further analyze the resilience of weights inside LSTM layers. As mentioned, two scenarios of Recursive weights (W) FI and Input weights (U) FI are performed separately. Fig. 6.7 depicts the results of this study. It is observed that in all ANNs, *W* weights are remarkably more vulnerable. At $BER = 1.5 \times 10^{-4}$, the accuracy drop resulting from FI experiments into *W* parameters is 2.41, 3.25, and 4.55 times worse than that of FI into *U*, for ANN-20, ANN-30, and ANN-40, respectively. The reason behind this observation is that *W* parameters involve memorizing the incoming time-series data through the recursive inputs of the LSTM layer. Therefore, faults in these parameters persist in influencing the recursive inputs through time and corrupt the memorizing ability of the network more than the other weights.

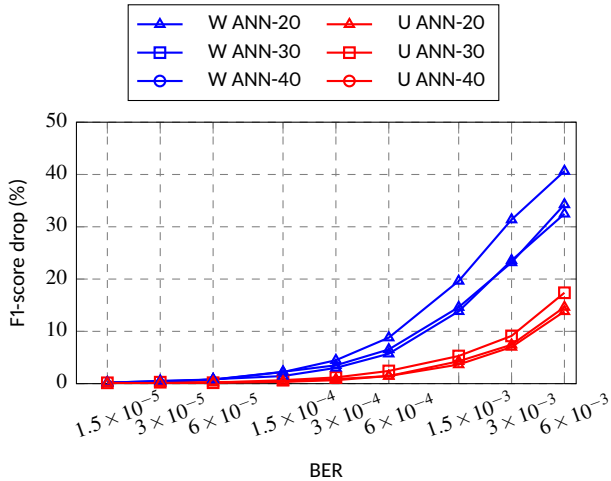
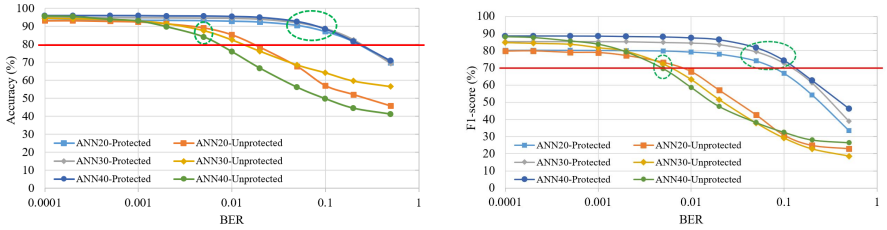


Figure 6.7: Inter-LSTM FI and accuracy comparison for LSTM-based ANNs.

6.1.3.4 Resilience Improvement of LSTM-based ANNs It is observed that ANNs are highly vulnerable to faults and adopted ANNs are no longer applicable when $BER > 1.5 \times 10^{-4}$. To improve the resilience of ANNs, any weight throughout an ANN that exceeds the expected values is replaced by 0, based on the offline mode profiling. To show the efficacy of the proposed method, we have applied it to the ANNs and performed a model-wise FI to obtain the results of accuracy loss, F1-score loss, and fault classification. Fig. 6.8 shows how the proposed method improves the accuracy and F1-score of ANNs in different BERs. Considering the required accuracy and F1-score of ANNs in the application, protected ANN-20 can operate when $BER <= 1.5 \times 10^{-3}$ (10 times higher than the unprotected one). In addition, the protected ANN-30 and ANN-40 can meet the requirements up to $BER = 3 \times 10^{-3}$ meaning that they can tolerate up to 20x higher BERs.



(a) Accuracy (b) F1-score

Figure 6.8: Accuracy and F1-score improvement of ANNs.

Moreover, we perform the fault classification in the FI experiments for protected ANNs. It is worth highlighting that the critical SDC rate and DUE rate have been significantly reduced in comparison with the unprotected ANNs. As a result, the critical SDC rate is reduced 7 times at $BER = 1.5 \times 10^{-3}$ for ANN-20 as well as 3.96 and 5.26 at $BER = 3 \times 10^{-3}$ for ANN-30 and ANN-40 respectively. Our protection mechanism results in 0.65 and 1.72 times less DUE at $BER = 1.5 \times 10^{-3}$ for ANN-20 and $BER = 3 \times 10^{-3}$ for ANN-30 respectively. In addition, it has removed any DUE in ANN-40 throughout FI experiments.

6.1.4 Discussion

This section presents a pioneering study that highlights the criticality of conducting reliability assessments for LSTM-based ANNs. Extensive experiments using fault injection have been performed to thoroughly examine the effect of faults on various sets of weights in ANNs. Notably, our findings demonstrate that recursive weights within LSTM cells are particularly vulnerable parameters in LSTM-based ANNs. Furthermore, a lightweight yet effective resilience improvement technique has been proposed which involves replacing faulty weights with 0s. Remarkably, the implementation of this technique results in ANNs experiencing 7 times fewer critical faults while successfully operating in environments up to 20 times harsher than unprotected networks.

6.2 Convolutional LSTM DNN for Disease Prediction

This section investigates the resilience of more complex LSTM-based DNNs containing convolutional layers.

6.2.1 Proposed Method: Resilience Assessment and Enhancement

6.2.1.1 LSTMs Under Study StageNet [87] is an LSTM-based DNN designed for disease prediction in healthcare. It predicts the stage of a patient's disease according to the characteristics of the tests performed by the patient through time. Fig. 6.9 illustrates the overall structure of StageNet. It is composed of an LSTM layer for characterizing the disease stage over time, a convolutional (CONV) module, and a Fully Connected (FC) layer to output the predicted disease condition. The CONV module contains one CONV layer and two FC layers, and their outputs are multiplied point-wise.

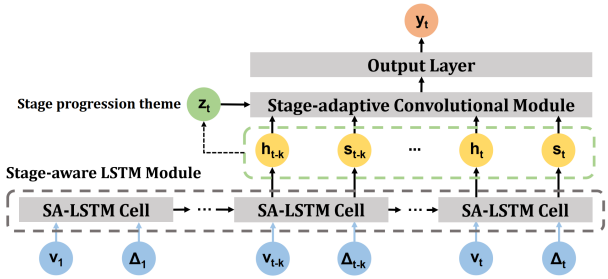


Figure 6.9: Overal structure of StageNet [87].

StageNet receives time-series data as inputs according to the patient visits $(v_1, v_2, \cdots, v_t)$ which contain numerical clinical features at different times $(\Delta_1, \Delta_2, \cdots, \Delta_t)$. They pass through the LSTM layer with multiple cells inferring the variation of a patient's health stage considering their current status. The produced results from time-series data by the LSTM layer are forwarded to the CONV module for learning patterns of the disease stages. Afterward, a classification is performed for disease stage and risk prediction. In the variations of StageNet, the number of LSTM cells can be customized, and also the presence of the CONV module is arbitrary. In this work, we consider four variations of StageNet for resilience analysis.

To measure the performance of StageNet, the following metrics are evaluated:

- **Accuracy:** This metric represents the percentage of correct predictions of StageNet compared to the expected outputs.
- **AUROC:** This metric refers to the area under the receiver operating characteristic curve illustrating the trade-off between true positive rate and false positive rate. It shows how a classifier can discriminate the positive and negative classes and it is extensively used when a dataset is imbalanced.

Since the distribution of data in the dataset between different disease stages is imbalanced, AUROC is a more suitable metric to show the performance of StageNet. An imbalanced data record is common in medical data since unstable cases happen less frequently than stable patient conditions.

6.2.1.2 Resilience Assessment by Fault Injection As mentioned, DNNs' weights are stored in memory, which is susceptible to soft errors. To assess the potential impact of these threats on the performance of LSTM DNNs, random bits determined by predefined Bit Error Rates (BER) are flipped in the weights of the DNN before inference. This process is repeated several times and average results over the repetition are reported.

To analyze the resilience of DNNs, random bitflips are applied throughout the DNN's weights to assess the overall network's behavior in the presence of faults. To quantify the resilience analysis, once the average performance metrics (Accuracy and AUROC) for a DNN under test are obtained, their difference with the fault-free metrics is considered as accuracy drop and AUROC drop. Furthermore, the output effect of faults is categorized into four classes, similar to the previous section: 1) Masked, 2) Non-critical SDC, 3) Critical SDC, and 4) DUE.

Moreover, to identify the critical bits in DNNs, we perform a bit-wise FI experiment throughout the DNNs. In such an experiment, one bit is considered as the target and it is flipped in all parameters and the inference is performed and the performance metrics are measured. The bit that has the highest impact on the performance metrics is identified as the most critical bit.

6.2.1.3 Resilience Enhancement: Weights and Activations Clipping To enhance the resilience of the LSTM DNNs, first, we profile the bit values in weights and all activation values with validation input data. Observing the bit patterns of weights leads us to the first protection mechanism for memory errors. On the other hand, faulty weights produce erroneous activation values. Therefore, observing the range of values in the activations through a forward pass of the DNN leads us to a second fault tolerance technique for LSTMs. Consequently, we propose two model-level fault tolerance techniques with zero memory overhead: 1) Weights Bit Clipping (WBC), and 2) Activations Value Clipping (AVC).

In the *WBC* method, all weights of fault-free DNN models are profiled and their bit patterns are analyzed. As a result, a consistent bit pattern is revealed in the DNNs under study. Moreover, using FI, the most critical bit is identified. Therefore, the method suggests clipping the most critical bits to a certain value throughout the DNN, before an inference.

In the *AVC* method, first, the input values to each activation function of the LSTM cells as well as the CONV and FC layers in DNNs are profiled and their maximum and minimum values are obtained, during a fault-free forward pass with validation data. The obtained values are then utilized for detecting faults. When an input to corresponding activation functions exceeds the determined value range, a fault is detected; so the the corresponding value is clamped to the minimum or maximum threshold value. It is worth mentioning that since LSTM cells possess sigmoid and Tanh activation functions, their outputs are already limited to a certain range (i.e., $[0, 1]$ and $[-1, 1]$ respectively). Therefore, AVC is applied to the inputs of activation functions throughout the LSTM-based DNNs, whether they are sigmoid, tanh, or ReLU.

6.2.2 Experimental Setup

The resilience of four variations of StageNet is experimented against faults in parameters. Two variations represent the full StageNet model which includes the CONV module, with different numbers of LSTM cells (384 and 72), and the two variations that exclude the CONV module, containing only the LSTM layer, also with 384 and 72 LSTM cells.

The test data is sourced from the Medical Information Mart for Intensive Care (MIMIC-III) data set, which includes 17 physiological variables recorded at each visit. This data is transformed into a 76-dimensional vector comprising numerical and one-hot encoded categorical clinical features for 33,678 unique patients.

Baseline metrics, including accuracy and AUROC in fault-free executions, are summarized in Table 6.2, alongside the number of parameters for each model. All models were executed on a general-purpose processor supporting 32-bit floating-point IEEE-754 data

representation.

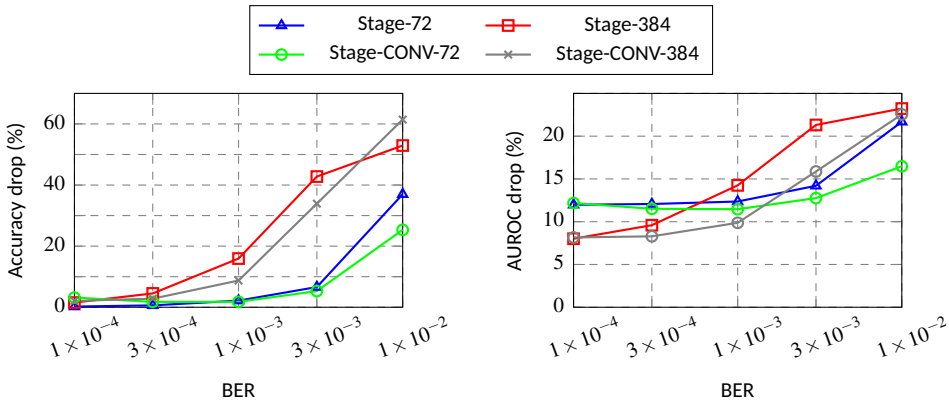
Table 6.2: Accuracy, AUROC, and the number of weights in variations of StageNet.

	Accuracy	AUROC	#weights in LSTM	#weights in CONV	#weights in FC
Stage-CONV-384	94.94%	79.21%	738,618	442,368	24,960
Stage-CONV-72	83.75%	76.75%	48,764	51,840	1,800
Stage-384	90.28%	79.29%	738,618	0	384
Stage-72	77.28%	76.97%	48,764	0	72

We conduct FI across all weights in the DNNs under study. The number of injected faults is determined using BER ranging from 0.0001 to 0.01, covering a comprehensive range of potential errors. FI is repeated 1,000 times to ensure an acceptable confidence level. For each iteration, a drop in accuracy and AUROC is obtained, and faults are classified. Eventually, the average results over all iterations are reported in the paper. All experiments are implemented and performed using PyTorch and executed on an *Intel Core™ i7-9700 CPU*.

6.2.3 Experimental Results

6.2.3.1 Resilience Analysis Results As mentioned, weight parameters across all DNN models are the fault space for random bit flips determined by different BERs. As illustrated in Fig. 6.10, the performance metrics for all DNNs significantly drop under FI campaigns. In this figure, as the BER increases, larger models (i.e., Stage-CONV-384 and Stage-384) demonstrate more accuracy drop than the smaller ones, at the same BERs. However, the AUROC drop metric appears differently. Stage-CONV-72 and Stage-72 are remarkably sensitive to faults in terms of their AUROC at the lowest BER. At higher BERs, the AUROC drop is higher for larger DNNs. As AUROC expresses the discrimination of classification over different thresholds, this observation shows that smaller DNNs are remarkably sensitive to faults to distinguish the stage of patients' disease correctly.



(a) Accuracy drop (b) AUROC drop

Figure 6.10: Resilience analysis of various structures of StageNet under model-wise FI.

According to the results, the AUROC metric for all DNNs falls below 60% when $BER =$

0.01. At such high BERs, although larger DNNs are shown to be more error-prone, none of them functions reliably in this particular application. Noteworthy that when AUROC is close to 50% DNNs perform random classification. On the other hand, it is observed that DNNs possessing the CONV module are generally more resilient than the ones without the CONV module. It shows that the CONV module increases the capability of fault masking in LSTM-based DNNs and improves their inherent resilience.

Fig. 6.11 presents the fault classification for DNNs under FI. In all experiments, the fault effects with *masked* and *non-critical SDCs* decrease as the BER increases, followed by more *critical SDCs* and *DUEs*. This figure also evidences that the DNNs with the CONV module are more resilient to faults than the ones without it. Obtained results indicate that when $BER = 0.01$, total critical SDC and DUE rate for StageNet-CONV-384, StageNet-CONV-72, StageNet-384 and StageNet-72 is 90.05%, 54.97%, 82.02%, and 77.33%, respectively.

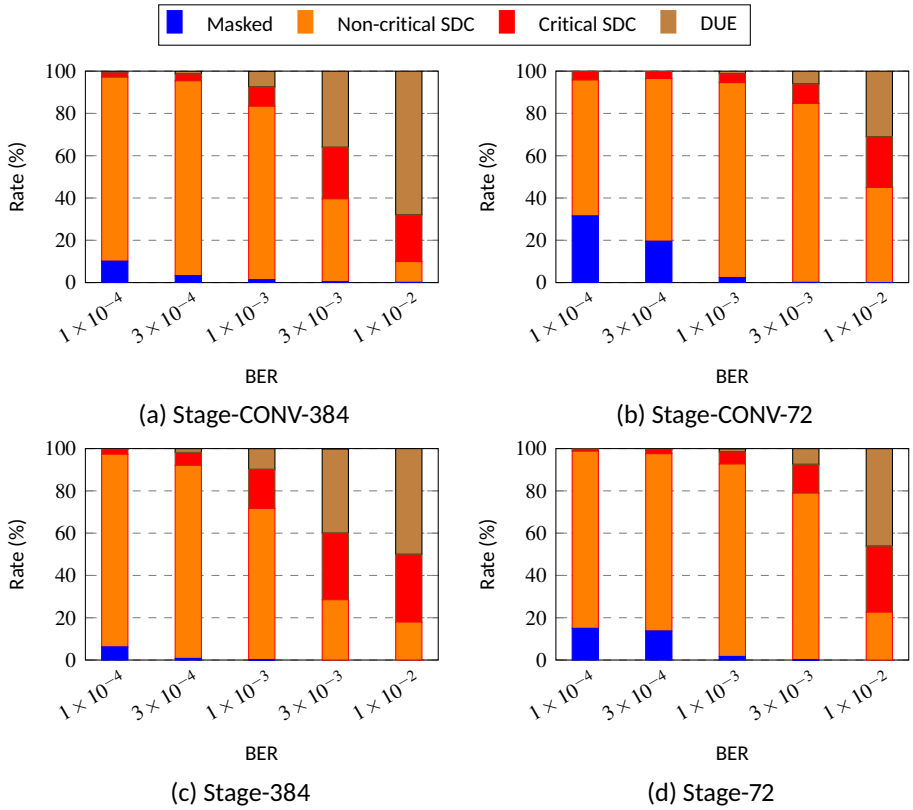
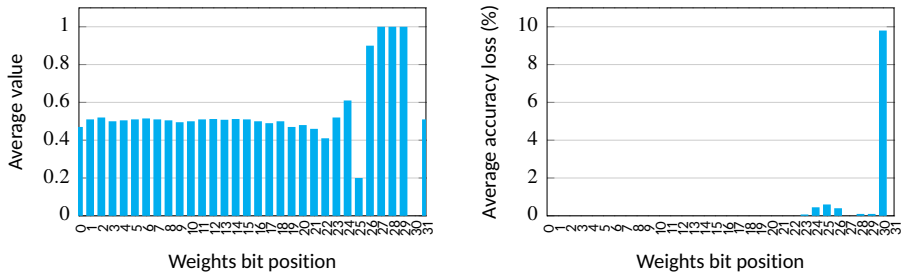


Figure 6.11: Fault classification in model-wise FI for different structures of StageNet.

Towards mitigating the effect of faults, we perform a bit-level analysis on weights. It is observed in Fig. 6.12-a that bits 0 (i.e., LSB) to 26 in 32-bit floating point data representation almost have a unified distribution between 0 and 1. while bits number 27, 28, and 29 are always 1 and bit number 30 is always 0. Therefore, the bits with constant values throughout the weights can be protected. However, to decrease the fault tolerance overhead, we further analyze the resilience of DNNs against each bit position. As depicted in Fig. 6.12-b, the accuracy drop for bit 30 in all DNNs is significantly higher than the other bits. Consequently, bit 30 is identified as the most critical bit possessing a constant value

of 0. Hence, this bit can be always set to 0, which ensures that the most critical bit is consistently protected.



(a) Average bit values (b) Average criticality

Figure 6.12: Bit-level analysis of weights based on their values and criticality.

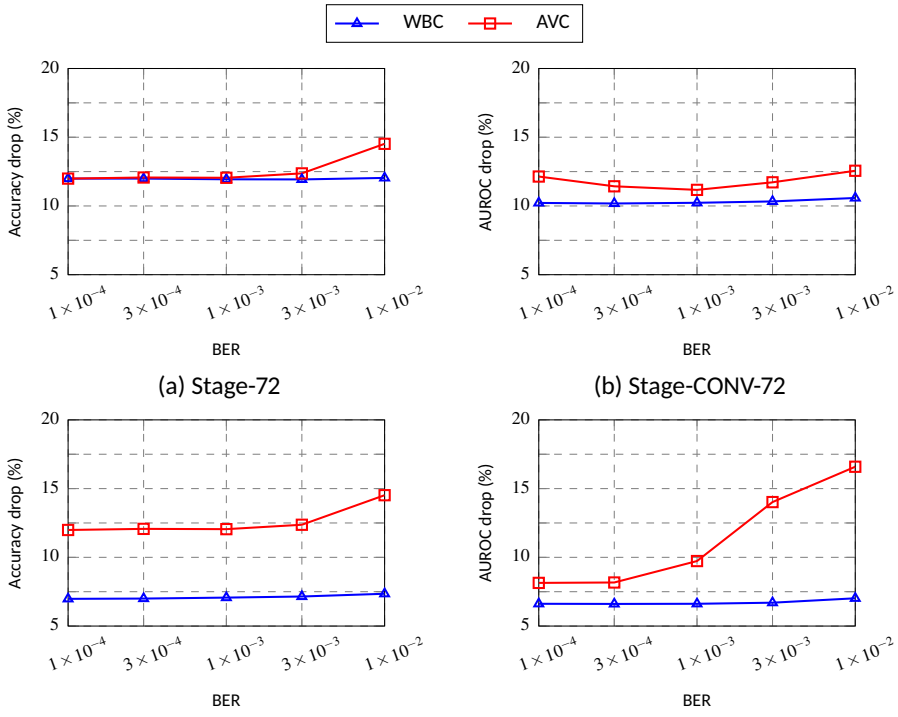
6.2.3.2 Resilience Improvement Results. All variations of StageNet are hardened by both WBC and AVC methods and are experimented with FI campaigns. It is observed that WBC lays a significant improvement on the reliability of LSTM-based DNNs. The accuracy drop of WBC-protected DNNs is close to zero through all BERs. Furthermore, the AUROC drop for them is remarkably improved for all DNNs, e.g., it is reduced by up to 3.2x when $BER = 0.01$. According to the fault classification results of WBC-based StageNet DNNs, the critical SDC and DUE rate is remarkably reduced. This protection mechanism is capable of effectively removing all DUE impacts on the outputs; since WBC prevents producing any large value by clipping the MSB in weights. As a result, the total critical SDC and DUE rate across DNNs is reduced by up to 278.6 times when $BER = 0.01$.

In AVC-protected DNNs, the accuracy drop and AUROC drop are reduced by up to 15.54 and 1.5 times among the DNNs, respectively, when $BER = 0.01$. The fault classification results through the FI campaigns on protected DNNs by AVC indicate that this method is also capable of removing all DUE effects. As a result, the total DUE and critical SDC rate for StageNet-CONV-384, StageNet-CONV-72, StageNet-384 and StageNet-72 is 13.88%, 5.18%, 36.47%, and 16.86% resulting in up to 10.6 times reduction across DNNs when $BER = 0.01$.

Fig. 6.13 compares the proposed zero-memory overhead fault-tolerant techniques for LSTM-based DNNs. As observed, WBC generally demonstrates a more consistent protective effect across different DNNs. It reduces the AUROC drop and the incidence of critical faults across DNNs more effectively than AVC. WBC achieves up to 2.36 times less AUROC drop than AVC throughout the DNNs when $BER = 0.01$. Nonetheless, each proposed fault-tolerance technique is applicable in different design scenarios. WBC applies directly to the memory and can be conducted to the bit values of the stored data before an inference. On the other hand, AVC is applied during the inference and prevents errors produced by faulty weights during the inference.

6.2.4 Discussion

In this section, the reliability of various LSTM-based DNNs (variants of StageNet) in healthcare is studied and two zero-memory overhead fault-tolerance techniques is proposed for them. Using FI, the fault resilience of different structures is analyzed. Results indicate that LSTM-based DNNs possessing convolutional layers demonstrate more resilience than the



(a) Stage-72 (b) Stage-CONV-72

(c) Stage-384 (d) Stage-CONV-384

Figure 6.13: Comparing WBC and AVC methods based on AUROC drop under FI.

ones without convolutional layers. Furthermore, we performed bit-level analysis resulting in the identification of the most critical bits.

Moreover, two zero-overhead protection techniques to improve their fault tolerance are proposed: Weights Bit Clipping (WBC) and Activations Value Clipping (AVC). It is shown that WBC can reduce the AUROC drop by up to 3.2 times and DUE and critical faults by up to 278.6 times compared to the unprotected DNNs at a high BER. Also, AVC reduces the AUROC drop by 1.5 and DUE and critical SDCs by 10.6, under the same conditions. The results demonstrate that the WBC method is more effective than AVC in mitigating the effect of faults occurring in the parameters of LSTM-based DNNs.

6.3 Chapter Conclusions

This section investigates and mitigates the fault resilience of various LSTM-based DNNs for healthcare applications for the first time. FI experiments extensively showed that LSTMs are highly susceptible to hardware faults, addressing RQ4.1 in Chapter 1. The purpose of LSTM layers is to retain memory through time, thus, faults can corrupt the memory and result in failure. In this section, multiple methods to mitigate the fault effects on LSTMs are introduced. It is shown that LSTMs can be significantly hardened against faults with minimal overhead, , addressing RQ4.2 in Chapter 1.

7 Conclusions and Future Directions

This thesis focuses on addressing the major challenges against hardware reliability assessment and enhancement for DNNs. Chapter 3 addresses the ambiguity in the literature pertaining to the reliability assessment methods for DNNs by conducting a systematic literature review. It includes 139 papers published between 2017-2022, categorizes the existing methods, investigates them in detail, and identifies the gaps in the literature. This Chapter identifies three major methods for reliability assessment of DNNs: Fault Injection, Analytical, and Hybrid. It is outlined that FI is mostly used for the reliability assessment of DNNs, however, it is non-scalable for emerging DNNs. On the other hand, analytical and hybrid methods are potential scalable alternatives for FI, yet they are not accurate and metric-oriented.

Accordingly, Chapter 4 attempts to introduce a new paradigm for fault resilience assessment of DNNs. It introduces DeepVigor, the first semi-analytical and metric-oriented method, which derives vulnerability factors for various components of DNNs through vulnerability value ranges for neurons. DeepVigor demonstrates 99.9% to 100% accuracy in identifying critical and non-critical faults for DNNs compared to FI, while 3.5 times faster. DeepVigor provides Vulnerability Factors (VFs) for bits, neurons, and layers in DNNs as a reliability metric and enables various fault mitigation techniques for them.

On top of that, DeepVigor+ is developed as an open-source tool to tackle the scalability problem in fault resilience assessment for DNNs, by reducing the analysis time from weeks to minutes. It exploits an optimal fault resilience analysis within neurons and obtains VFs for DNNs' layers and models. The results indicate that DeepVigor+ derives VFs with less than 1% error, with 14.9 up to 26.9 times fewer simulations than the best-known state-of-the-art statistical FI. This open-source method unleashes a fast resilience assessment, enabling fine-grain evaluation and design space exploration for various fault-tolerant and cost-effective designs for DNNs.

Moreover, this Chapter presents QDeepVigor, which is an extension of DeepVigor for QNNs. It is shown that DeepVigor and its extensions are as accurate as FI, yet faster and scalable. QDeepVigor is exploited for a cross-layer reliability enhancement in QNN accelerators. In this method, the critical neurons identified by QDeepVigor are split at the model level and corrected by a Lightweight Correction Unit in the accelerator. This method provides a similar fault resilience for accelerators to TMR-based methods, with twice reduced memory overhead. Furthermore, QDeepVigor is employed to conduct a hybrid resilience assessment method for QNN accelerators. QDeepVigor prunes the critical faults in QNN, resulting in a remarkable acceleration in the fault simulation process. According to the obtained results, 77.48% of faults are classified as non-vulnerable and vulnerable by QDeepVigor, leading to an early stop during the fault simulation.

Chapter 5 and Chapter 6 present multiple methods to achieve a reliable DNN deployment in safety-critical applications. Chapter 5 introduces two novel methods to achieve model-level fault tolerance for COTS DNN accelerators with minimal overhead. ProAct proposes a KD-based progressive training for determining optimal threshold values in a novel activation function, i.e., HyReLU. The proposed HyReLU reduces memory overhead by applying neuron-wise clipping solely in the last layer and layer-wise clipping in the preceding layers. Results indicate that ProAct significantly improves the resilience of DNNs by up to 6.4 times in high BERs. Furthermore, it dramatically reduces memory overhead, achieving reductions up to 134.28 times compared to a neuron-wise state-of-the-art method. This source code of this method is published to enable researchers to present more effective approaches in this area.

Moreover, a model-level hardening method for CNNs is proposed with selective chan-

nel duplication and EDAC layers, enabling them to correct errors inherently, at inference time. The hardened CNNs perform reliably at orders of magnitude higher error rates than unprotected CNNs with merely a 15% hardening ratio, yet incurring 12% performance overhead. To minimize the incurred overhead, for the first time, a vulnerability-based pruning that improves resilience is presented. As a result, the hardened pruned CNNs achieve up to 24% higher performance than the un-pruned hardened CNNs. Eventually, Chapter 5 introduces the open-source SentinelNN framework that integrates vulnerability-aware pruning using DeepVigor+ and hardening (channel duplication and range restriction). This framework allows controllability to produce cost-effective fault-tolerant CNNs deployed on DNN HW accelerators.

Chapter 6 attempts to enhance the resilience of LSTM-based DNNs in medical applications. The fault resilience of various structures of LSTM-based DNNs is investigated in this Chapter, using FI. It is shown that LSTM layers are more vulnerable than CONV and FC layers. Throughout this Chapter, multiple methods to mitigate the fault effects on LSTMs are investigated, including online monitoring and correction to weights, and clipping-based methods to weights and activations. It is shown that LSTMs can be significantly hardened against faults with minimal overhead.

The attempts in this thesis have built a foundation for further research in the field of reliable deployment of emerging ML systems. The presented scalable fault resilience analysis can be applied to other applications such as object detection. Furthermore, it can be extended for emerging ML models such as Transformers. On the other hand, the proposed SIHFT-based approaches can be applied not only to emerging ML models but also to hardware accelerators for their reliable and efficient deployment. We presented multiple open-source tools that enable researchers and engineers in this field to extend the existing methods. The application of proposed methods is not limited to hardware faults and they can be also employed to enhance the robustness of DNNs against malicious attacks in weights or inputs.

The achievements in this thesis can be a foundation to further research and developments, such as:

- DeepVigor+ can be a basis to develop scalable hardware-aware fault resilience methods for emerging DL systems. Safety certification of DL systems compliant with industrial standards can be accomplished by designing cross-layer techniques for assessing hardware reliability.
- DeepVigor+ can be used in design space exploration for producing reliable DNNs, employing Neural Architecture Search (NAS).
- The proposed techniques for enhancing the reliability of DNNs can be integrated with DHA design to increase the efficiency of a fault-tolerant deployment. Moreover, the effectiveness of model-level techniques under fault simulation/emulation on DHAs can be studied.
- All proposed techniques in this thesis can be extended for emerging DL models such as LLMs and ViTs, to enable their fault resilience assessment and enhancement.

List of Figures

List of Tables

References

[1] "N2D2 CAD framework for DNNs". https://github.com/cea-list/N2D2. [Online].

[2] H.R.6216 - national artificial intelligence initiative act of 2020. https://www.congress.gov/bill/116th-congress/house-bill/6216, 2020. [Online].

[3] Artificial intelligence (AI) in healthcare market size, share & industry analysis, by platform, by application, by end-user, and regional forecasts, 2024-2032. https://www.fortunebusinessinsights.com/industry-reports/artificial-intelligence-in-healthcare-market-100534, 2024. [Online].

[4] Automotive artificial intelligence (AI) market size, share, and trends 2024 to 2034. https://www.precedenceresearch.com/automotive-artificial-intelligence-market, 2024. [Online].

[5] IBM global ai adoption index – enterprise report. https://newsroom.ibm.com/2024-01-10-Data-Suggests-Growth-in-Enterprise-Adoption-of-AI-is-Due-to-Widespread-Deployment-by-Early-Adopters, 2024. [Online].

[6] Regulation (EU) 2024/1689 of the european parliament and of the council. https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:32024R1689, 2024. [Online].

[7] 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*, 378(2164):20190164, 2020.

[8] G. Abich, R. Garibotti, J. Gava, R. Reis, and L. Ost. Impact of thread parallelism on the soft error reliability of convolution neural networks. In *2022 IEEE 13th Latin America Symposium on Circuits and System (LASCAS)*, pages 1–4. IEEE, 2022.

[9] G. Abich, R. Garibotti, R. Reis, and L. Ost. The impact of soft errors in memory units of edge devices executing convolutional neural networks. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 69(3):679–683, 2022.

[10] G. Abich, J. Gava, R. Garibotti, R. Reis, and L. Ost. Applying lightweight soft error mitigation techniques to embedded mixed precision deep neural networks. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 68(11):4772–4782, 2021.

[11] 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)*, pages 1–4. IEEE, 2020.

[12] G. Abich, R. Reis, and L. Ost. The impact of precision bitwidth on the soft error reliability of the mobilenet network. In *2021 IEEE 12th Latin America Symposium on Circuits and System (LASCAS)*, pages 1–4. IEEE, 2021.

[13] K. Adam, I. I. Mohamed, and Y. Ibrahim. A selective mitigation technique of soft errors for dnn models used in healthcare applications: Densenet201 case study. *IEEE Access*, 9:65803–65823, 2021.

[14] K. Adam, I. I. Mohd, and Y. Ibrahim. Analyzing the instructions vulnerability of dense convolutional network on gpus. *International Journal of Electrical and Computer Engineering (IJECE)*, 11(5):4481–4488, 2021.

[15] K. Adam, I. I. Mohd, and Y. M. Younis. The impact of the soft errors in convolutional neural network on gpus: Alexnet as case study. *Procedia Computer Science*, 182:89–94, 2021.

[16] 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)*, pages 286–296. IEEE, 2022.

[17] D. Agiakatsikas, N. Foutris, A. Sari, V. Vlagkoulis, I. Souvatzoglou, M. Psarakis, M. Luján, M. Kastriotou, and C. Cazzaniga. Evaluation of the xilinx deep learning processing unit under neutron irradiation. In *2021 21th European Conference on Radiation and Its Effects on Components and Systems (RADECS)*, pages 1–4. IEEE, 2021.

[18] M. H. Ahmadilivani, M. Barbareschi, S. Barone, A. Bosio, M. Daneshtalab, S. Della Torca, G. Gavarini, M. Jenihhin, J. Raik, A. Ruospo, E. Sanchez, and M. Taheri. Special Session: Approximation and Fault Resiliency of DNN Accelerators. In *IEEE 41st VLSI Test Symposium (VTS)*, pages 1–10. San Diego, United States of America, 2023.

[19] 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, J. E. Rodriguez Condia, M. Sonza Reorda, M. Taheri, and M. Traiola. Special Session: Reliability Assessment Recipes for DNN Accelerators. In *IEEE 42nd VLSI Test Symposium (VTS)*, pages 1–11. Tempe, United States of America, 2024.

[20] 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. In *The 30th IEEE International Symposium on On-Line Testing and Robust System Design (IOLTS)*, pages 1–6. Rennes, France, 2023.

[21] M. H. Ahmadilivani, J. Raik, M. Daneshtalab, and M. Jenihhin. Deepvigor+: A Scalable, Accurate and Automated Framework for Resilience Analysis of Deep Neural Networks. *Under review*, pages 1–14, 2024.

[22] M. H. Ahmadilivani, J. Raik, M. Daneshtalab, and A. Kuusik. Analysis and Improvement of Resilience for Long Short-Term Memory Neural Networks. In *IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, pages 1–4. Juan-Les-Pennes, France, 2023.

[23] M. H. Ahmadilivani, M. Taheri, J. Raik, M. Daneshtalab, and M. Jenihhin. DeepVigor: VulnerabIlity Value RanGes and FactORs for DNNs' Reliability Assessment. In *IEEE European Test Symposium (ETS)*, pages 1–6. Venice, Italy, 2023.

[24] M. H. Ahmadilivani, M. Taheri, J. Raik, M. Daneshtalab, and M. Jenihhin. Enhancing Fault Resilience of QNNs by Selective Neuron Splitting. In *IEEE 5th International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, pages 1–5. Hangzhou, China, 2023.

[25] 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*, 56(6):1–36, 2024.

[26] M. S. Ali, T. B. Iqbal, K.-H. Lee, A. Muqeet, S. Lee, L. Kim, and S.-H. Bae. Erdnn: Error-resilient deep neural networks with a new error correction layer and piecewise rectified linear unit. *IEEE Access*, 8:158702–158711, 2020.

[27] C. Amarnath, M. Mejri, K. Ma, and A. Chatterjee. Soft error resilient deep learning systems using neuron gradient statistics. In *2022 IEEE 28th International Symposium on On-Line Testing and Robust System Design (IOLTS)*, pages 1–7. IEEE, 2022.

[28] C. Amarnath, M. Mejri, K. Ma, and A. Chatterjee. Error resilience in deep neural networks using neuron gradient statistics. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2023.

[29] 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)*, pages 190–196. IEEE, 2018.

[30] A. P. Arechiga and A. J. Michaels. The robustness of modern deep learning architectures against single event upset errors. In *2018 IEEE High Performance extreme Computing Conference (HPEC)*, pages 1–6. IEEE, 2018.

[31] 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)*, pages 1–10. IEEE, 2018.

[32] V. Bandeira, F. Rosa, R. Reis, and L. Ost. Non-intrusive fault injection techniques for efficient soft error vulnerability analysis. In *2019 IFIP/IEEE 27th International Conference on Very Large Scale Integration (VLSI-SoC)*, pages 123–128. IEEE, 2019.

[33] 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*, 67(7):1560–1565, 2020.

[34] F. Benevenuti, M. Gonçalves, E. C. F. P. Junior, R. G. Vaz, O. L. Gonçalez, J. R. Azambuja, and F. L. Kastensmidt. Neutron-induced faults on cnn for aerial image classification on sram-based fpga using softcore gpu and hls. In *2021 21th European Conference on Radiation and Its Effects on Components and Systems (RADECS)*, pages 1–4. IEEE, 2021.

[35] F. Benevenuti, F. Libano, V. Pouget, F. L. Kastensmidt, and P. Rech. Comparative analysis of inference errors in a neural network implemented in sram-based fpga induced by neutron irradiation and fault injection methods. In *2018 31st Symposium on Integrated Circuits and Systems Design (SBCCI)*, pages 1–6. IEEE, 2018.

[36] A. Benso and S. DiCarlo. The art of fault injection. *Journal of Control Engineering and Applied Informatics*, 13(4):9–18, 2011.

[37] A. Birolini. *Reliability engineering*. Springer, 2007.

[38] C. Bolchini, L. Cassano, A. Miele, and A. Nazzari. Selective hardening of cnns based on layer vulnerability estimation. In *2022 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, pages 1–6. IEEE, 2022.

[39] C. Bolchini, L. Cassano, A. Miele, and A. Toschi. Fast and accurate error simulation for cnns against soft errors. *IEEE Transactions on Computers*, 2022.

[40] 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)*, pages 1–6. IEEE, 2019.

[41] 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)*, pages 1–10. IEEE, 2021.

[42] 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)*, pages 1–6. IEEE, 2021.

[43] S. Burel, A. Evans, and L. Anghel. Zero-overhead protection for cnn weights. In *2021 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, pages 1–6. IEEE, 2021.

[44] 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*, 22(2):120–128, 2022.

[45] S. BurelT, A. EvansT, and L. Anghel. Improving dnn fault tolerance in semantic segmentation applications. In *2022 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, pages 1–6. IEEE, 2022.

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

[47] R. Cantoro, N. I. Deligiannis, M. S. Reorda, M. Traiola, and E. Valea. Evaluating data encryption effects on the resilience of an artificial neural network. In *2020 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, pages 1–4. IEEE, 2020.

[48] A. Canziani, A. Paszke, and E. Culurciello. An analysis of deep neural network models for practical applications. *arXiv preprint arXiv:1605.07678*, 2016.

[49] N. Cavagnero, F. Dos Santos, M. Ciccone, G. Averta, T. Tommasi, and P. Rech. Transient-fault-aware design and training to enhance dnns reliability with zero-overhead. In *2022 IEEE 28th International Symposium on On-Line Testing and Robust System Design (IOLTS)*, pages 1–7. IEEE, 2022.

[50] A. Chan, A. Gujarati, K. Pattabiraman, and S. Gopalakrishnan. The fault in our data stars: studying mitigation techniques against faulty training data in machine learning applications. In *2022 52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 163–171. IEEE, 2022.

[51] K.-C. J. Chen, M. Ebrahimi, T.-Y. Wang, Y.-C. Yang, and Y.-H. Liao. A noc-based simulator for design and evaluation of deep neural networks. *Microprocessors and Microsystems*, 77:103145, 2020.

[52] Y.-H. Chen, J. Emer, and V. Sze. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. *ACM SIGARCH Computer Architecture News*, 44(3):367–379, 2016.

[53] Z. Chen, G. Li, and K. Pattabiraman. A low-cost fault corrector for deep neural networks through range restriction. In *2021 51st Annual IEEE/IFIP DSN*, pages 1–13. IEEE, 2021.

[54] 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*, pages 1–23, 2019.

[55] Z. Chen, N. Narayanan, B. Fang, G. Li, K. Pattabiraman, and N. DeBardeleben. Tensorfi: A flexible fault injection framework for tensorflow applications. In *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*, pages 426–435. IEEE, 2020.

[56] 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 *IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*. Tanger, Morocco, 2024.

[57] 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)*, pages 90–99. IEEE, 2020.

[58] W. Choi, D. Shin, J. Park, and S. Ghosh. Sensitivity based error resilient techniques for energy efficient deep neural network accelerators. In *Proceedings of the 56th Annual Design Automation Conference 2019*, pages 1–6, 2019.

[59] A. Cicchetti, F. Ciccozzi, and A. Pierantonio. Multi-view approaches for software and system modelling: a systematic literature review. *Software and Systems Modeling*, 18(6):3207–3233, 2019.

[60] R. M. Coelho, J. Gouveia, M. A. Botto, H. I. Krebs, and J. Martins. Real-time walking gait terrain classification from foot-mounted inertial measurement unit using convolutional long short-term memory neural network. *Expert Systems with Applications*, 203:117306, 2022.

[61] 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)*, pages 1–7. IEEE, 2021.

[62] J. E. R. Condia, B. Du, M. S. Reorda, and L. Sterpone. Flexgripplus: An improved gpgpu model to support reliability analysis. *Microelectronics Reliability*, 109:113660, 2020.

[63] 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)*, pages 278–287. IEEE, 2022.

[64] P. Corneliou, P. Nikolaou, M. K. Michael, and T. Theocharides. Fine-grained vulnerability analysis of resource constrained neural inference accelerators. In *2021 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, pages 1–6. IEEE, 2021.

[65] N. Corporation. "NVDLA Open Source Project". http://nvdla.org. [Online].

[66] N. Corporation. "NVIDIA TensorRT". https://developer.nvidia.com/tensorrt. [Online].

[67] 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.

[68] I. Dadras, M. H. Ahmadilivani, S. Banerji, J. Raik, and A. Abloo. An Efficient Analog Convolutional Neural Network Hardware Accelerator Enabled by a Novel Memoryless Architecture for Insect-Sized Robots. In *11th International Conference on Modern Circuits and Systems Technologies (MOCAST)*, pages 1–6. Bremen, Germany, 2022.

[69] I. Dadras, S. Seydi, M. H. Ahmadilivani, J. Raik, and M. E. Salehi. Fully-Fusible Convolutional Neural Networks for End-to-End Fused Architecture with FPGA Implementation. In *30th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, pages 1–5. Istanbul, Turkey, 2023.

[70] 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. *Machine Learning and Systems*, 3:800–811, 2021.

[71] C. De Sio, S. Azimi, and L. Sterpone. An emulation platform for evaluating the reliability of deep neural networks. In *2020 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, pages 1–4. IEEE, 2020.

[72] C. De Sio, S. Azimi, and L. Sterpone. Firenn: Neural networks reliability evaluation on hybrid platforms. *IEEE Transactions on Emerging Topics in Computing*, 10(2):549–563, 2022.

[73] N. I. Deligiannis, R. Cantoro, M. S. Reorda, M. Traiola, and E. Valea. Towards the integration of reliability and security mechanisms to enhance the fault resilience of neural networks. *IEEE Access*, 9:155998–156012, 2021.

[74] 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*, pages 248–255. Ieee, 2009.

[75] L. Deng, G. Li, S. Han, L. Shi, and Y. Xie. Model compression and hardware acceleration for neural networks: A comprehensive survey. *Proceedings of the IEEE*, 108(4):485–532, 2020.

[76] M. Dhouibi, A. K. Ben Salem, A. Saidi, and S. Ben Saoud. Accelerating deep neural networks implementation: A survey. *IET Computers & Digital Techniques*, 15(2):79–96, 2021.

[77] F. F. dos Santos, L. Draghetti, L. Weigel, L. Carro, P. Navaux, and P. Rech. Evaluation and mitigation of soft-errors in neural network-based object detection in three gpu architectures. In *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, pages 169–176. IEEE, 2017.

[78] F. F. Dos Santos, A. Kritikakou, J. E. R. Condia, J. D. Guerrero Balaguera, M. S. Reorda, O. Sentieys, and P. Rech. Characterizing a neutron-induced fault model for deep neural networks. *IEEE Transactions on Nuclear Science*, 2022.

[79] F. F. dos Santos, P. Navaux, L. Carro, and P. Rech. Impact of reduced precision in the reliability of deep neural networks for object detection. In *2019 IEEE European Test Symposium (ETS)*, pages 1–6. IEEE, 2019.

[80] 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*, 68(2):663–677, 2018.

[81] B. Du, S. Azimi, C. De Sio, L. Bozzoli, and L. Sterpone. On the reliability of convolutional neural network implementation on sram-based fpga. In *2019 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, pages 1–6. IEEE, 2019.

[82] J. Elliott, F. Mueller, F. Stoyanov, and C. Webster. Quantifying the impact of single bit flips on floating point arithmetic. Technical report, North Carolina State University. Dept. of Computer Science, 2013.

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

[84] 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)*, pages 1–7. IEEE, 2020.

[85] G. Gambardella, N. J. Fraser, U. Zahid, G. Furano, and M. Blott. Accelerated radiation test on quantized neural networks trained with fault aware training. In *2022 IEEE Aerospace Conference (AERO)*, pages 1–7. IEEE, 2022.

[86] G. Gambardella, J. Kappauf, M. Blott, C. Doehring, M. Kumm, P. Zipf, and K. Vissers. Efficient error-tolerant quantized neural network accelerators. In *2019 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, pages 1–6. IEEE, 2019.

[87] J. Gao, C. Xiao, Y. Wang, W. Tang, L. M. Glass, and J. Sun. Stagenet: Stage-aware neural networks for health risk prediction. In *Proceedings of The Web Conference 2020*, pages 530–540, 2020.

[88] Z. Gao, X. Wei, H. Zhang, W. Li, G. Ge, Y. Wang, and P. Reviriego. Reliability evaluation of pruned neural networks against errors on parameters. In *2020 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, pages 1–6. IEEE, 2020.

[89] Z. Gao, Y. Yao, X. Wei, T. Yan, S. Zeng, G. Ge, Y. Wang, A. Ullah, and P. Reviriego. Reliability evaluation of fpga based pruned neural networks. *Microelectronics Reliability*, 130:114498, 2022.

[90] T. Garrett and A. D. George. Improving dependability of onboard deep learning with resilient tensorflow. In *2021 IEEE Space Computing Conference (SCC)*, pages 134–142. IEEE, 2021.

[91] J. Gava, G. Dorneles, R. Reis, R. Garibotti, and L. Ost. Soft error assessment of cnn inference models running on a risc-v processor. In *2022 29th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, pages 1–4. IEEE, 2022.

[92] G. Gavarini, D. Stucchi, A. Ruospo, G. Boracchi, and E. Sanchez. Open-set recognition: an inexpensive strategy to increase dnn reliability. In *2022 IEEE 28th International Symposium on On-Line Testing and Robust System Design (IOLTS)*, pages 1–7. IEEE, 2022.

[93] B. Ghavami, M. Sadati, Z. Fang, and L. Shannon. Fitact: Error resilient deep neural networks via fine-grained post-trainable activation functions. In *2022 DATE*, pages 1239–1244. IEEE, 2022.

[94] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer. A survey of quantization methods for efficient neural network inference. In *Low-Power Computer Vision*, pages 291–326. Chapman and Hall/CRC, 2022.

[95] M. Goldblum, L. Fowl, S. Feizi, and T. Goldstein. Adversarially robust distillation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 3996–4003, 2020.

[96] 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)*, pages 311–316. IEEE, 2021.

[97] B. F. Goldstein, S. Srinivasan, D. Das, K. Banerjee, L. Santiago, V. C. Ferreira, A. S. Nery, S. Kundu, and F. M. França. Reliability evaluation of compressed deep learning models. In *2020 IEEE 11th Latin American Symposium on Circuits & Systems (LASCAS)*, pages 1–5. IEEE, 2020.

[98] O. Goloubeva, M. Rebaudengo, M. S. Reorda, and M. Violante. *Software-implemented hardware fault tolerance*. Springer Science & Business Media, 2006.

[99] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[100] H. Guan, L. Ning, Z. Lin, X. Shen, H. Zhou, and S.-H. Lim. In-place zero-space memory protection for cnn. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, pages 5734–5743, 2019.

[101] J. D. Guerrero Balaguera, J. E. R. Condia, and M. S. Reorda. Neural network's reliability to permanent faults: Analyzing the impact of performance optimizations in gpus. In *2022 29th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, pages 1–4. IEEE, 2022.

[102] J. D. Guerrero Balaguera, L. Galasso, R. L. Sierra, and M. S. Reorda. Reliability assessment of neural networks in gpus: A framework for permanent faults injections. In *2022 IEEE 31st International Symposium on Industrial Electronics (ISIE)*, pages 959–962. IEEE, 2022.

[103] J. D. Guerrero Balaguera, L. Galasso, R. L. Sierra, E. Sanchez, and M. S. Reorda. Evaluating the impact of permanent faults in a gpu running a deep neural network. In *2022 IEEE International Test Conference in Asia (ITC-Asia)*, pages 96–101. IEEE, 2022.

[104] 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)*, pages 1–6. IEEE, 2022.

[105] 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)*, 12(1):1–26, 2019.

[106] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[107] S. K. S. Hari, M. Sullivan, T. Tsai, and S. W. Keckler. Making convolutions resilient via algorithm-based error detection techniques. *IEEE Transactions on Dependable and Secure Computing*, 2021.

[108] S. K. S. Hari, T. Tsai, M. Stephenson, S. W. Keckler, and J. Emer. Sassifi: An architecture-level fault injection tool for gpu application resilience evaluation. In *2017 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 249–258. IEEE, 2017.

[109] E. J. Harris, I.-H. Khoo, and E. Demircan. A survey of human gait-based artificial intelligence applications. *Frontiers in Robotics and AI*, 8:749274, 2022.

[110] 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*, pages 770–778, 2016.

[111] Y. He, P. Balaprakash, and Y. Li. Fidelity: Efficient resilience analysis framework for deep learning accelerators. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 270–281. IEEE, 2020.

[112] G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2(7), 2015.

[113] 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 DATE*, pages 1241–1246. IEEE, 2020.

[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)*, pages 434–441. IEEE, 2021.

[115] Y. Hong, J. Lian, L. Xu, J. Min, Y. Wang, L. J. Freeman, and X. Deng. Statistical perspectives on reliability of artificial intelligence systems. *Quality Engineering*, pages 1–23, 2022.

[116] A. Hosseinkhani and B. Ghavami. Improving soft error reliability of fpga-based deep neural networks with reduced approximate tmr. In *2021 11th International Conference on Computer Engineering and Knowledge (ICCKE)*, pages 459–464. IEEE, 2021.

[117] 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*, 23(1):57–77, 2019.

[118] Z. Huang, W. Shao, X. Wang, L. Lin, and P. Luo. Rethinking the pruning criteria for convolutional neural network. *Advances in Neural Information Processing Systems*, 34:16305–16318, 2021.

[119] 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*, 18(1):6869–6898, 2017.

[120] E. Ibe, H. Taniguchi, Y. Yahagi, K.-i. Shimbo, and T. Toba. Impact of scaling on neutron-induced soft error in srams from a 250 nm to a 22 nm design rule. *IEEE Transactions on Electron Devices*, 57(7):1527–1538, 2010.

[121] Y. Ibrahim, H. Wang, and K. Adam. Analyzing the reliability of convolutional neural networks on gpus: Googlenet as a case study. In *2020 International Conference on Computing and Information Technology (ICCIT-1441)*, pages 1–6. IEEE, 2020.

[122] 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*, 8:19490–19503, 2020.

[123] 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*, 115:113969, 2020.

[124] Y. Ibrahin, J. Liu, X. Yang, H. Sha, and H. Wang. Analyzing the impact of soft errors in deep neural networks on gpus from instruction level. *WSEAS Transaction on Systems and Control*, 15:699–708, 2020.

[125] M. Jang and J. Hong. Mate: Memory-and retraining-free error correction for convolutional neural network weights. *Journal of information and communication convergence engineering*, 19(1):22–28, 2021.

[126] M. Jasemi, S. Hessabi, and N. Bagherzadeh. Enhancing reliability of emerging memory technology for machine learning accelerators. *IEEE Transactions on Emerging Topics in Computing*, 9(4):2234–2240, 2020.

[127] 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*, pages 1–12, 2017.

[128] R. L. R. 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*, 69(3):567–575, 2022.

[129] 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)*, pages 99–104. IEEE, 2020.

[130] 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)*, pages 1139–1144. IEEE, 2020.

[131] 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)*, pages 1–6. IEEE, 2020.

[132] J.-S. Kim and J.-S. Yang. Dris-3: Deep neural network reliability improvement scheme in 3d die-stacked memory based on fault analysis. In *2019 56th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2019.

[133] M. C. Kim and C. S. Smidts. Three suggestions on the definition of terms for the safety and reliability analysis of digital systems. *Reliability Engineering & System Safety*, 135:81–91, 2015.

[134] J. C. Knight. Safety critical systems: challenges and directions. In *Proceedings of the 24th international conference on software engineering*, pages 547–550, 2002.

[135] I. Koren and C. M. Krishna. *Fault-tolerant systems*. Morgan Kaufmann, 2020.

[136] A. Krizhevsky, v. Nair, and G. Hinton. "The CIFAR-10 Dataset". https://www.cs.toronto.edu/~kriz/cifar.html. [Online].

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

[138] S. Laskar, M. H. Rahman, and G. Li. Tensorfi+: A scalable fault injection framework for modern deep learning neural networks. In *2022 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, pages 246–251. IEEE, 2022.

[139] S. Laskar, M. H. Rahman, B. Zhang, and G. Li. Characterizing deep learning neural network failures between algorithmic inaccuracy and transient hardware faults. In *2022 IEEE 27th Pacific Rim International Symposium on Dependable Computing (PRDC)*, pages 54–67. IEEE, 2022.

[140] M. Lavallée, P.-N. Robillard, and R. Mirsalari. Performing systematic literature reviews with novices: An iterative approach. *IEEE Transactions on Education*, 57(3):175–181, 2013.

[141] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[142] S. Lee, I. Choi, and J.-S. Yang. Bipolar vector classifier for fault-tolerant deep neural networks. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*, pages 673–678, 2022.

[143] S.-S. Lee and J.-S. Yang. Value-aware parity insertion ecc for fault-tolerant deep neural network. In *2022 DATE*, pages 724–729. IEEE, 2022.

[144] 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*, pages 502–506. IEEE, 2009.

[145] 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*, pages 1–12, 2017.

[146] G. Li, K. Pattabiraman, and N. DeBardeleben. Tensorfi: A configurable fault injector for tensorflow applications. In *2018 IEEE International symposium on software reliability engineering workshops (ISSREW)*, pages 313–320. IEEE, 2018.

[147] 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 AICAS*, pages 1–5. IEEE, 2020.

[148] 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*, 68(5):865–872, 2021.

[149] 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*, 66(1):216–222, 2018.

[150] 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*, 67(7):1478–1484, 2020.

[151] B. Lim and S. Zohren. Time-series forecasting with deep learning: a survey. *Philosophical Transactions of the Royal Society A*, 379(2194):202–209, 2021.

[152] C. Liu, C. Chu, D. Xu, Y. Wang, Q. Wang, H. Li, X. Li, and K.-T. Cheng. Hyca: A hybrid computing architecture for fault-tolerant deep learning. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 41(10):3400–3413, 2021.

[153] Z. Liu, Z. Deng, and X. Yang. Using checksum to improve the reliability of embedded convolutional neural networks. *Microelectronics Reliability*, 136:114666, 2022.

[154] Z. Liu, Y. Liu, Z. Chen, G. Guo, and H. Wang. Analyzing and increasing soft error resilience of deep neural networks on arm processors. *Microelectronics Reliability*, 124:114331, 2021.

[155] Z. Liu and X. Yang. An efficient structure to improve the reliability of deep neural networks on arms. *Microelectronics Reliability*, 136:114729, 2022.

[156] A. Lotfi, S. Hukerikar, K. Balasubramanian, P. Racunas, N. Saxena, R. Bramley, and Y. Huang. Resiliency of automotive object detection networks on gpu architectures. In *2019 IEEE International Test Conference (ITC)*, pages 1–9. IEEE, 2019.

[157] L. M. Luza, A. Ruospo, A. Bosio, E. Sanchez, and L. Dilillo. A model-based framework to assess the reliability of safety-critical applications. In *2021 24th International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS)*, pages 41–44. IEEE, 2021.

[158] 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)*, pages 1–6. IEEE, 2020.

[159] 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*, 4(3):161–175, 2018.

[160] 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)*, pages 25–31. IEEE, 2020.

[161] 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.

[162] A. Mahmoud, S. K. S. Hari, C. W. Fletcher, S. V. Adve, C. Sakr, N. R. Shanbhag, P. Molchanov, M. B. Sullivan, T. Tsai, and S. W. Keckler. Optimizing selective protection for cnn resilience. In *ISSRE*, pages 127–138, 2021.

[163] 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 20nm kintex ultrascale™ fpga. *IEEE Transactions on Nuclear Science*, 2022.

[164] E. Malekzadeh, N. Rohbani, Z. Lu, and M. Ebrahimi. The impact of faults on dnns: A case study. In *2021 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, pages 1–6. IEEE, 2021.

[165] R. Manne and S. C. Kantheti. Application of artificial intelligence in healthcare: chances and challenges. *Current Journal of Applied Science and Technology*, 40(6):78–89, 2021.

[166] L. Matanaluza, A. Ruospo, D. Soderstrom, C. Cazzaniga, M. Kastriotou, E. Sanchez, A. Bosio, and L. Dilillo. Emulating the effects of radiation-induced soft-errors for the reliability assessment of neural networks. *IEEE Transactions on Emerging Topics in Computing*, 2021.

[167] J. Mendez, K. Bierzynski, M. P. Cuéllar, and D. P. Morales. Edge intelligence: concepts, architectures, applications, and future directions. *ACM Transactions on Embedded Computing Systems (TECS)*, 21(5):1–41, 2022.

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

[169] T. Mohaidat and K. Khalil. A survey on neural network hardware accelerators. *IEEE Transactions on Artificial Intelligence*, 2024.

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

[171] S. Mousavi, M. H. Ahmadilivani, J. Raik, M. Jenihhin, and M. Daneshtalab. ProAct: Progressive Training for Hybrid Clipped Activation Function to Enhance Resilience of DNNs. *Under review*, pages 1–12, 2024.

[172] R. I. Mukhamediev, Y. Popova, Y. Kuchin, E. Zaitseva, A. Kalimoldayev, A. Symagulov, V. Levashenko, F. Abdoldina, V. Gopejenko, K. Yakunin, et al. Review of artificial intelligence and machine learning technologies: classification, restrictions, opportunities and challenges. *Mathematics*, 10(15):2552, 2022.

[173] S. Mukherjee. *Architecture design for soft errors*. Morgan Kaufmann, 2011.

[174] 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.

[175] M. A. Neggaz, I. Alouani, P. R. Lorenzo, and S. Niar. A reliability study on cnns for critical embedded systems. In *2018 IEEE 36th International Conference on Computer Design (ICCD)*, pages 476–479. IEEE, 2018.

[176] M. A. Neggaz, I. Alouani, S. Niar, and F. Kurdahi. Are cnns reliable enough for critical applications? an exploratory study. *IEEE Design & Test*, 37(2):76–83, 2019.

[177] 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)*, pages 1075–1080. IEEE, 2021.

[178] M. Nicolaidis. *Soft errors in modern electronic systems*, volume 41. Springer Science & Business Media, 2010.

[179] D. Oliveira, L. Pilla, N. DeBardeleben, S. Blanchard, H. Quinn, I. Koren, P. Navaux, and P. Rech. Experimental and analytical study of xeon phi reliability. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–12, 2017.

[180] F. J. Ordóñez and D. Roggen. Deep convolutional and lstm recurrent neural networks for multimodal wearable activity recognition. *Sensors*, 16(1):115, 2016.

[181] E. Ozen and A. Orailoglu. Sanity-check: Boosting the reliability of safety-critical deep neural network applications. In *2019 IEEE 28th Asian Test Symposium (ATS)*, pages 7–75. IEEE, 2019.

[182] 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*, 39(11):3250–3262, 2020.

[183] E. Ozen and A. Orailoglu. Just say zero: containing critical bit-error propagation in deep neural networks with anomalous feature suppression. In *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pages 1–9. IEEE, 2020.

[184] E. Ozen and A. Orailoglu. Low-cost error detection in deep neural network accelerators with linear algorithmic checksums. *Journal of Electronic Testing*, 36(6):703–718, 2020.

[185] E. Ozen and A. Orailoglu. Snr: S queezing n umerical r ange defuses bit error vulnerability surface in deep neural networks. *ACM Transactions on Embedded Computing Systems (TECS)*, 20(5s):1–25, 2021.

[186] B. Parchekani, S. Nazari, M. H. Ahmadilivani, A. Azarpeyvand, J. Raik, T. Ghasempouri, and M. Daneshtalab. Zero-Memory-Overhead Clipping-Based Fault Tolerance for LSTM Deep Neural Networks. In *IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, pages 1–4. Oxforshire, United Kingdom, 2024.

[187] L. Ping, J. Tan, and K. Yan. Sern: Modeling and analyzing the soft error reliability of convolutional neural networks. In *Proceedings of the 2020 on Great Lakes Symposium on VLSI*, pages 445–450, 2020.

[188] J. Ponader, K. Thomas, S. Kundu, and Y. Solihin. Milr: Mathematically induced layer recovery for plaintext space error correction of cnns. In *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 75–87. IEEE, 2021.

[189] A. S. Prasad, L. Benini, and F. Conti. Specialization meets flexibility: A heterogeneous architecture for high-efficiency, high-flexibility ar/vr processing. In *2023 60th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2023.

[190] M. Rausand. *Reliability of safety-critical systems: theory and applications*. John Wiley & Sons, 2014.

[191] B. Reagen, U. Gupta, L. Pentecost, P. Whatmough, S. K. Lee, N. Mulholland, D. Brooks, and G.-Y. Wei. Ares: A framework for quantifying the resilience of deep neural networks. In *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2018.

[192] R. L. Rech and P. Rech. Impact of layers selective approximation on cnns reliability and performance. In *2020 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, pages 1–4. IEEE, 2020.

[193] R. L. Rech and P. Rech. Reliability of google's tensor processing units for embedded applications. In *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 376–381. IEEE, 2022.

[194] 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*, pages 779–788, 2016.

[195] E. Rojas, D. Pérez, J. C. Calhoun, L. B. Gomez, T. Jones, and E. Meneses. Understanding soft error sensitivity of deep learning models and frameworks through checkpoint alteration. In *2021 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 492–503. IEEE, 2021.

[196] J. Rostovski, M. H. Ahmadilivani, A. Krivošei, A. Kuusik, and M. M. Alam. Real-Time Gait Anomaly Detection Using 1D-CNN and LSTM. In *Nordic Conference on Digital Health and Wireless Solutions*, pages 260–278. Oulu, Finland, Springer, 2024.

[197] 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)*, pages 1–6. IEEE, 2020.

[198] A. Ruospo, A. Bosio, A. Ianne, and E. Sanchez. Evaluating convolutional neural networks reliability depending on their data representation. In *2020 23rd Euromicro Conference on Digital System Design (DSD)*, pages 672–679. IEEE, 2020.

[199] A. Ruospo, G. Gavarini, I. Bragaglia, M. Traiola, A. Bosio, and E. Sanchez. Selective hardening of critical neurons in deep neural networks. In *2022 25th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS)*, pages 136–141. IEEE, 2022.

[200] A. Ruospo, G. Gavarini, C. De 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)*, pages 1–6. IEEE, 2023.

[201] 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)*, pages 1–5. IEEE, 2021.

[202] A. Ruospo and E. Sanchez. On the reliability assessment of artificial neural networks running on ai-oriented mpsocs. *Applied Sciences*, 11(14):6455, 2021.

[203] A. Ruospo, E. Sanchez, L. M. Luza, L. Dilillo, M. Traiola, and A. Bosio. A survey on deep learning resilience assessment methodologies. *Computer*, 56(2):57–66, 2023.

[204] A. Ruospo, E. Sanchez, M. Traiola, I. O'connor, and A. Bosio. Investigating data representation for efficient and reliable convolutional neural networks. *Microprocessors and Microsystems*, 86:104318, 2021.

[205] M. Sabbagh, C. Gongye, Y. Fei, and Y. Wang. Evaluating fault resiliency of compressed deep neural networks. In *2019 IEEE International Conference on Embedded Software and Systems (ICESS)*, pages 1–7. IEEE, 2019.

[206] M. Sabih, F. Hannig, and J. Teich. Fault-tolerant low-precision dnns using explainable AI. In *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, pages 166–174. IEEE, 2021.

[207] B. Salami, O. S. Unsal, and A. C. Kestelman. On the resilience of rtl nn accelerators: Fault characterization and mitigation. In *2018 30th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, pages 322–329. IEEE, 2018.

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

[209] R. Sanchez-Iborra and A. F. Skarmeta. Tinyml-enabled frugal smart objects: Challenges and opportunities. *IEEE Circuits and Systems Magazine*, 20(3):4–18, 2020.

[210] 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)*, pages 979–984. IEEE, 2018.

[211] C. Schorn, A. Guntoro, and G. Ascheid. An efficient bit-flip resilience optimization method for deep neural networks. In *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1507–1512. IEEE, 2019.

[212] N. Seifert, X. Zhu, and L. W. Massengill. Impact of scaling on soft-error rates in commercial microprocessors. *IEEE Transactions on Nuclear Science*, 49(6):3100–3106, 2002.

[213] 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*, 37(2):30–57, 2020.

[214] M. L. Shooman. *Reliability of computer systems and networks: fault tolerance, analysis, and design*. John Wiley & Sons, 2003.

[215] 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)*, pages 343–348. IEEE, 2021.

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

[217] R. Singh, N. S. Mangat, R. Singh, and N. S. Mangat. Stratified sampling. *Elements of survey sampling*, pages 102–144, 1996.

[218] I. Souvatzoglou, A. Papadimitriou, A. Sari, V. Vlagkoulis, and M. Psarakis. Analyzing the single event upset vulnerability of binarized neural networks on sram fpgas. In *2021 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, pages 1–6. IEEE, 2021.

[219] T. Spyrou, S. A. El-Sayed, E. Afacan, L. A. Camuñas-Mesa, B. Linares-Barranco, and H.-G. Stratigopoulos. Reliability analysis of a spiking neural network hardware accelerator. In *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 370–375. IEEE, 2022.

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

[221] Y. Su and C.-C. J. Kuo. On extended long short-term memory and dependent bidirectional recurrent neural network. *Neurocomputing*, 356:151–161, 2019.

[222] 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)*, pages 275–279. IEEE, 2021.

[223] 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*, 105(12):2295–2329, 2017.

[224] M. Taheri, M. H. Ahmadilivani, M. Jenihhin, M. Daneshtalab, and J. Raik. Appraiser: DNN Fault Resilience Analysis Employing Approximation Errors. In *IEEE 26th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS)*, pages 124–127. Tallinn, Estonia, 2023.

[225] 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)*, pages 19–24. IEEE, 2024.

[226] 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 *IEEE 24th International Symposium on Quality Electronic Design (ISQED)*, pages 1–8. San Francisco, United States of America, 2023.

[227] 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*, 77:1897–1938, 2021.

[228] T. N. Theis and H.-S. P. Wong. The end of moore's law: A new beginning for information technology. *Computing in science & engineering*, 19(2):41–50, 2017.

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

[230] 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)*, pages 284–291. IEEE, 2021.

[231] 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)*, pages 272–277. IEEE, 2021.

[232] 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*, pages 65–74, 2017.

[233] G. Van Houdt, C. Mosquera, and G. Nápoles. A review on the long short-term memory model. *Artificial Intelligence Review*, 53(8):5929–5955, 2020.

[234] A. Veronesi, F. Dall'Occo, D. Bertozzi, M. Favalli, and M. Krstic. Exploring software models for the resilience analysis of deep learning accelerators: the nvdla case study. In *2022 25th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS)*, pages 142–147. IEEE, 2022.

[235] M. Vijay and R. Mittal. Algorithm-based fault tolerance: a review. *Microprocessors and Microsystems*, 21(3):151–161, 1997.

[236] H.-B. Wang, Y.-S. Wang, J.-H. Xiao, S.-L. Wang, and T.-J. Liang. Impact of single-event upsets on convolutional neural networks in xilinx zynq fpgas. *IEEE Transactions on Nuclear Science*, 68(4):394–401, 2021.

[237] J. Wang, J. Zhu, X. Fu, D. Zang, K. Li, and W. Zhang. Enhancing neural network reliability: Insights from hardware/software collaboration with neuron vulnerability quantization. *IEEE Transactions on Computers*, 2024.

[238] O. Weng, A. Meza, Q. Bock, B. Hawks, J. Campos, N. Tran, J. M. Duarte, and R. Kastner. Fkeras: A sensitivity analysis tool for edge neural networks. *Journal on Autonomous Transportation Systems*, 2024.

[239] XILINX. "SoCs with Hardware and Software Programmability". https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html. [Online].

[240] D. Xu, C. Chu, Q. Wang, C. Liu, Y. Wang, L. Zhang, H. Liang, and K.-T. Cheng. A hybrid computing architecture for fault-tolerant deep learning accelerators. In *2020 IEEE 38th International Conference on Computer Design (ICCD)*, pages 478–485. IEEE, 2020.

[241] D. Xu, Z. Zhu, C. Liu, Y. Wang, H. Li, L. Zhang, and K.-T. Cheng. Persistent fault analysis of neural networks on fpga-based acceleration system. In *2020 IEEE 31st International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, pages 85–92. IEEE, 2020.

[242] D. Xu, Z. Zhu, C. Liu, Y. Wang, S. Zhao, L. Zhang, H. Liang, H. Li, and K.-T. Cheng. Reliability evaluation and analysis of fpga-based neural network acceleration system. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 29(3):472–484, 2021.

[243] 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)*, pages 163–168. IEEE, 2020.

[244] C. J. B. Yann, Y. LeCun, and C. Cortes. "The MNIST DATABASE of Handwritten Digits". http://yann.lecun.com/exdb/mnist/. [Online].

[245] Y. Yu, X. Si, C. Hu, and J. Zhang. A review of recurrent neural networks: Lstm cells and network architectures. *Neural computation*, 31(7):1235–1270, 2019.

[246] L. Yuan, Y. Chen, T. Wang, W. Yu, Y. Shi, Z.-H. Jiang, F. E. Tay, J. Feng, and S. Yan. Tokens-to-token vit: Training vision transformers from scratch on imagenet. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 558–567, 2021.

[247] 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 ITC*, pages 1–10. IEEE, 2020.

[248] J. Zhan, R. Sun, W. Jiang, Y. Jiang, X. Yin, and C. Zhuo. Improving fault tolerance for reliable dnn using boundary-aware activation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 41(10):3414–3425, 2021.

[249] J. J. Zhang, K. Basu, and S. Garg. Fault-tolerant systolic array based accelerators for deep neural network execution. *IEEE Design & Test*, 36(5):44–53, 2019.

[250] 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)*, pages 1–6. IEEE, 2018.

[251] Y. Zhang, H. Itsuji, T. Uezono, T. Toba, and M. Hashimoto. Estimating vulnerability of all model parameters in dnn with a small number of fault injections. In *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 60–63. IEEE, 2022.

[252] 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*, 32(7):1677–1689, 2020.

[253] 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)*, pages 545–552. IEEE, 2022.

Acknowledgements

In May 2020, when I received an email from Prof. Jaan Raik informing me of my acceptance for this position, I had no idea how intriguing and challenging the next four years would be. Now, as I reach the end of this journey, I am proud to reflect on the invaluable memories and transformative experiences it has brought me; ones I could not have gained anywhere else. This period not only allowed me to delve deeply into scientific research but also broadened my horizons by enabling me to visit many beautiful places and connect with inspiring people. I am deeply grateful to everyone who accompanied and supported me throughout this path.

First and foremost, I owe my deepest gratitude to my beloved wife, Mina, my ultimate source of inspiration. It was she who encouraged me to accept this position, and her unwavering support has been the driving force behind my ability to embark on and complete this journey. Leaving behind the life we knew and stepping into an uncertain future was a monumental change for both of us, and I will forever admire her resilience and strength in navigating these challenges alongside me. I am also profoundly thankful to my parents for their lifelong encouragement and support, which enabled me to pursue my academic aspirations. They instilled in me the values of hard work, honesty, and trustworthiness, qualities that have been instrumental in my journey.

I extend my heartfelt gratitude to Prof. Jaan Raik, Prof. Masoud Danestalab, and Prof. Maksim Jenihhin, whose guidance and mentorship have been invaluable. I was incredibly fortunate to work with such a distinguished and competent team of professors during my PhD studies, from whom I learned not only about science and research but also about life and academic growth. Under Prof. Raik's supervision, I benefited immensely from his extensive knowledge, insightful advice, and unwavering support. His positive outlook, humility, and encouragement to learn from setbacks have left a lasting impression on me, and I am deeply grateful for the nurturing research environment he cultivated. Prof. Daneshtalab taught me the importance of embracing challenges, exploring new topics outside my expertise, and broadening my perspective for future career opportunities. From Prof. Jenihhin, I learned the value of persistence, meticulous organization, and pragmatism in conducting high-quality research.

Throughout my studies, I had the privilege of collaborating with numerous talented PhD students, scientists, and engineers, all of whom contributed to the impact and scope of my research. I am especially thankful to Hamidreza Mousavi, Mahdi Taheri, Marten Roots, Ali Emre Karatopuk, Dr. Iman Dadras, Jakob Rostovski, Natalia Cherezova, Luca Di Mauro (ARM), Dr. Marco Restifo (ARM), Ahmad Mirsalari, Dr. Mohammad Riazati, Salvatore Pappalardo, Sakineh Seyedi, Samira Nazari, Prof. Alberto Bosio, Dr. Alar Kuusik, Dr. Levent Aksoy, Prof. Ali Azarpeyvand, Prof. Mostafa E. Salehi, Prof. Tara Ghasempouri, Dr. Marko Koort, Dr. Mohammad Eslami, Yuto Kobayashi, and Akiha Kusumoto.

I sincerely appreciate Prof. Luciano Ost and Prof. Yanjing Li, for their time and effort in reviewing this thesis. Their insightful feedback and invaluable comments have greatly contributed to improving its quality. Finally, I would like to express my gratitude to the staff of the Computer Systems Department at TalTech for providing a supportive work environment and access to the necessary resources for conducting high-quality research.

This dissertation is financially 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"), and the Estonian Research Council vai "Sustainable Artificial Internet of Things (SAIoT)"-("TEM-TA138").

Abstract

Assessment and Enhancement of Hardware Reliability for Deep Neural Networks

Artificial Intelligence and Machine Learning (ML) have revolutionized the conventional computing paradigm. They are extensively applied to various use cases due to their strength in solving complex problems. Safety-critical applications such as autonomous vehicles and healthcare employ ML to achieve more efficient outcomes. However, the safety of ML systems is a huge concern. Hardware reliability is one of the prominent aspects of safety, where hardware faults can lead to system failures and result in catastrophes.

Due to the variety of Deep Neural Networks (DNNs) and accelerators, there exists a wide range of research papers with distinct methods evaluating the reliability of DNNs and their accelerator. The extent of this domain and the variety of solutions have created an ambiguous research area, restraining researchers from precisely identifying and comprehending the gaps in the literature. To bridge this gap, this thesis conducts the first Systematic Literature Review focused exclusively on all methods of reliability assessment of DNNs. This study establishes a comprehensive picture of this topic by thoroughly categorizing and analyzing the existing methods and identifying the gaps in the literature.

By overviewing the literature, it can be observed that Fault Injection (FI) is the major method employed for hardware reliability assessment for DNNs. Nonetheless, since emerging DNNs are gigantic and DNN accelerators are complex, FI's non-scalability leads to an obstacle to reliability assessment. This thesis introduces the first semi-analytical, metric-oriented, and accurate method for fault resilience assessment of DNNs, called DeepVigor. The extension of this method (DeepVigor+) tackles the scalability of FI, providing highly accurate vulnerability factors for layers and models within a few minutes rather than days, by an optimal fault propagation analysis for weights and activations.

DNNs are inherently resilient to faults and they can mask a huge amount of faults, yet their accuracy may be considerably compromised by certain critical faults. Architecture-level fault-tolerant techniques are accelerator-specific and exploit hardware redundancy with performance and memory overhead and they do not apply to general-purpose processors and pre-designed IPs. Whereas algorithm-level fault-tolerant techniques modify the DNN models in software that any accelerator executes. The existing algorithm-level techniques are too complex, induce considerable overhead, and provide low fault resilience compared to the induced overhead.

This thesis proposes model-level fault tolerance methods for Convolutional Neural Networks (CNNs) as Software-Implemented Hardware Fault Tolerance. In this regard, a novel low-cost activation restriction method, called ProAct, is introduced by conducting progressive training based on Knowledge Distillation to achieve significant resilience in DNNs with minimal memory overhead. Moreover, an innovative model-level hardening method for CNNs is proposed conducting selective channel duplication and error detection and correction layers while minimizing the induced overhead leveraging vulnerability-aware pruning. Furthermore, the open-source SentinelNN framework is presented, employing DeepVigor+ vulnerability assessment for applying selective channel duplication to CNNs accompanied with range restriction.

Moreover, this thesis investigates the fault resilience assessment and enhancement of LSTM-based CNNs for the first time. The fault resilience of various structures of LSTM-based DNNs is investigated and multiple methods to mitigate the fault effects on LSTMs are proposed. It is shown that LSTM layers are highly vulnerable to faults, yet they can be significantly hardened with minimal overhead.

Kokkuvõte

Riistvara töökindluse hindamine ja täiustamine süvanärvivõrkude jaoks

Tehisintellekt ja masinõpe (ML) on toonud revolutsiooni tavapärasesse andmetöötluse paradigmasse. Neid kasutatakse laialdaselt erinevates valdkondades tänu nende tugevusele keeruliste probleemide lahendamisel. ML-i kasutatakse ohutuskriitilistes rakendustes tõhusamate tulemuste saavutamiseks, näiteks autonoomsetes sõidukites ja tervisehoius. Seetõttu on ML-süsteemide ohutus ülimalt oluline. Üks oluline töökindluse aspekt on riistvara töökindlus kuna riistvarariketest tekkinud süsteemitõrked võivad põhjustada katastroofe.

Süvanärvivõrkude (DNN) ja kiirendite külluse tõttu leidub DNN-ide ja kiirendite töökindluse hindamise kohta mitmeid teadustöid. Selle valdkonna ulatus ja lahenduste mitmekesisus on loonud laialivalguva uurimisvaldkonna, mis teeb teadlastel olemasoleva kirjanduse puuduste hindamist raskemaks. Antud uuring koostab esimese süstemaatilise kirjandusülevaate, mis käsitleb kõiki olemasolevaid DNN-ide töökindluse hindamise meetodeid. Et luua terviklik pilt antud valdkonnast, liigitatakse olemasolevad meetodid, analüüsitakse neid ja tuuakse välja olemasoleva kirjanduse puudused.

Kirjanduse ülevaatamisest selgub, et kõige levinum riistvara töökindluse hindamise meetod on vigade sisestamine (VS). Modernsete DNN-ide ja kiirendite keerukuse tõttu on VS halb skaleeruvus töökindluse hindamisel tõsiseks katsumuseks. Antud töös tutvustatakse esimest pool-analüütilist, tunnustele keskenduvat ja täpset DNN-ide hindamise meetodit, nimega DeepVigor. Selle meetodi laiendus (DeepVigor+) lahendab VS skaleeruvuse probleemi ja väljastab suure täpsusega süsteemi haavatavuse tegureid. Erinevalt VS-st, võtab protsess aega päevade asemel tunde, kasutades optimeeritud kaalude ja aktiveerimisvigade propageerimise analüüsi.

DNN-id on olemuslikult rikete suhtes töökindlad ja suudavad suure osa nendest ära siluda, kuid teatud kriitiliste rikete tulemusel võib langeda oluliselt nende täpsus. Arhitektuuritasemel töötavad tõrkekindluse tehnikad sõltuvad kiirendi tüübist ja rakendavad riistvara liiasust, millega kaasneb suurem jõudluse ja mälu koormus. Lisaks ei rakendu need hästi üldotstarbelistele protsessoritele ega juba disainitud moodulitele. Algoritmitasemel töötavad meetodid muudavad aga DNN mudeleid neid jooksutavas tarkvaras. Olemasolevad algoritmilised tehnikad on liialt keerukad, märkimisväärse lisakuluga ja nendest tulenev tõrkekindlus on võrdlemisi väike.

Käesolevas lõputöös pakutakse välja CNN-ide jaoks välja kaks tarkvaralise lahendusega mudelitasemel tõrketaluvuse tõstmise meetodit. Esimene neist on lihtne aktiveerimise piiramise meetod, ProAct. See võetakse kasutusele teadmiste destilleerimisel põhineva progresseeruva õpetamise läbiviimisel, et saavutada DNN-ides märkimisväärne vastupidavus minimaalse mälumahuga. Lisaks sellele töötasin välja uendusliku mudelitasemel CNN-ide kaitsmise meetodi, mis töötab kanalite selektiivse dubleerimise ning vigade tuvastamise ja korrigeerimise kihtide abil. See minimeerib samal ajal tekitatud üldkulu, kasutades haavatavuste teadlikku kärpimist. Peale selle esitatakse vabavaraline raamistik SentinelNN, mis rakendab DeepVigor+ haavatavuse hindamise lähenemist valikuliseks kanali dubleerimiseks CNN võrkudes koostöös väärtuste piirkonna piiramise meetodiga.

Lisaks uurib käesolev töö esmakordselt LSTM-põhiste CNN-ide riketele vastupidavuse hindamist ja täiustamist. Uuritakse LSTM-põhiste DNN-ide erinevate struktuuride tõrkekindlust ja pakutakse välja erinevaid meetodeid LSTM-ide tõrkemõjude leevendamiseks. Näidatakse, et LSTM kihid on vigade suhtes väga tundlikud, kuid neid saab minimaalse üldkuluga oluliselt karastada.

Appendix 1

I

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*, 56(6):1–36, 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

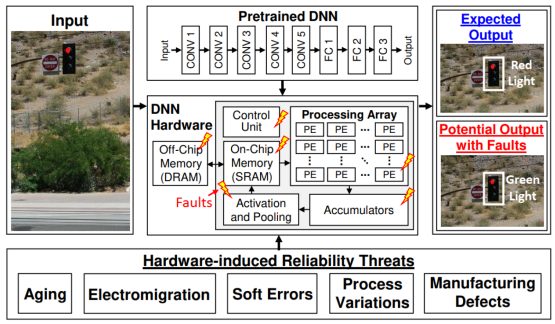


Fig. 1. Hardware-induced reliability threats in an example DHA and their possible impact on the output [1].

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

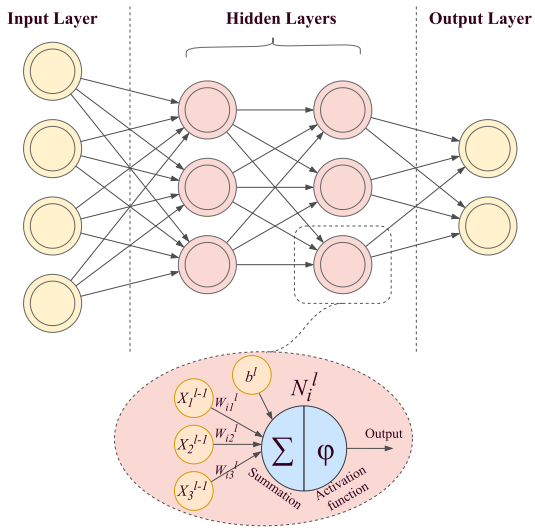


Fig. 2. Abstract view of a simple neural network with the detail of a neuron.

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].

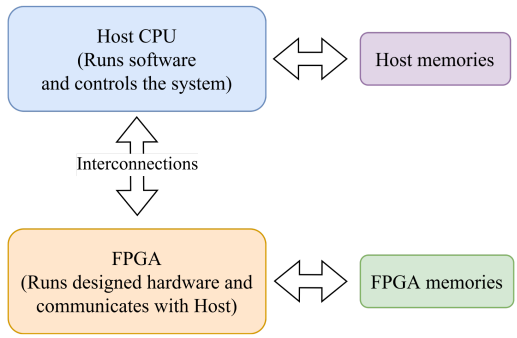


Fig. 3. Typical structure of an FPGA-based DNN accelerator [38].

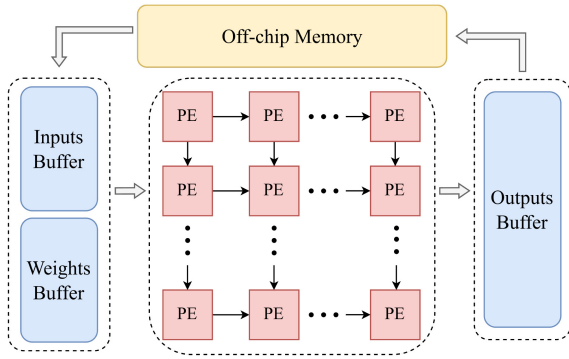


Fig. 4. An example of spatial architecture for ASIC-based DNN accelerators [42].

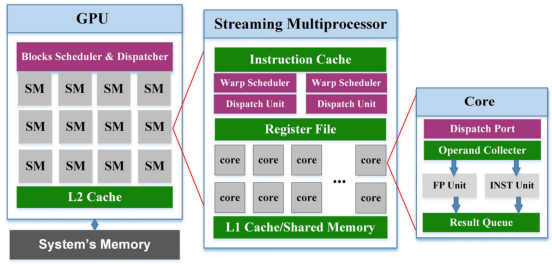


Fig. 5. General architecture of CUDA-based GPUs [44].

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

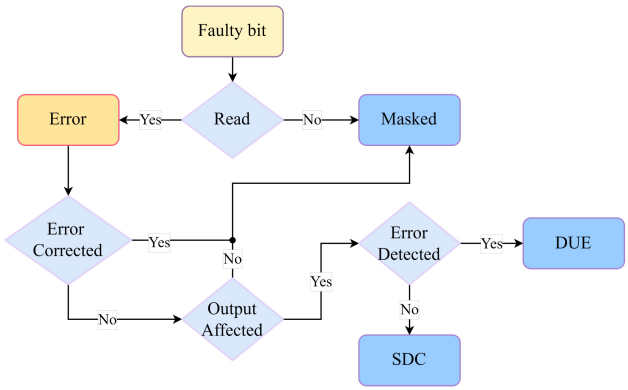


Fig. 6. The adopted fault classification based on the output point of view, as in Reference [51].

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.

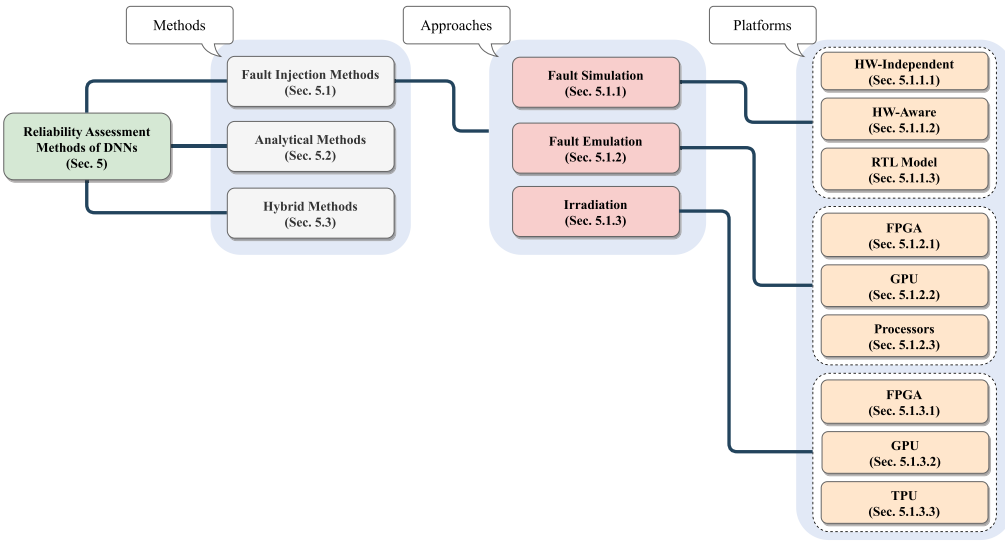


Fig. 7. Top-level overview of the reliability assessment methods in this work.

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

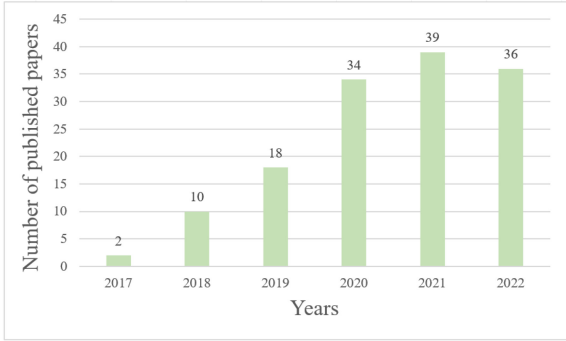


Fig. 8. Number of included papers over years.

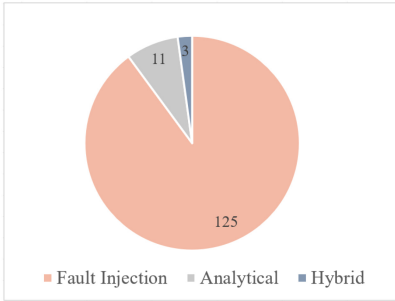


Fig. 9. Proportion of each method in the reliability assessment of DNNs among included works.

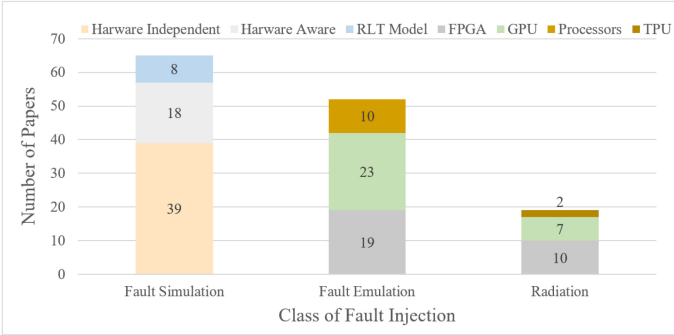


Fig. 10. Distribution of included papers over different FI approaches and platforms.

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].

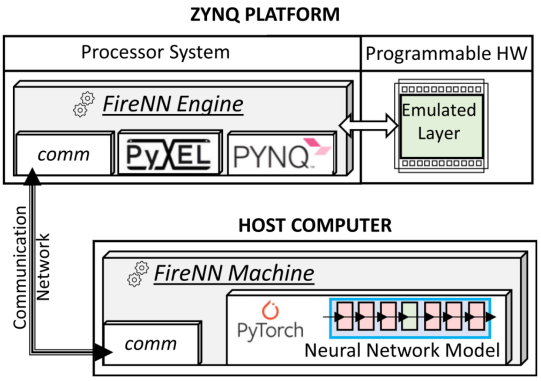


Fig. 11. An overview of the architecture of the FireNN platform [97, 98].

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

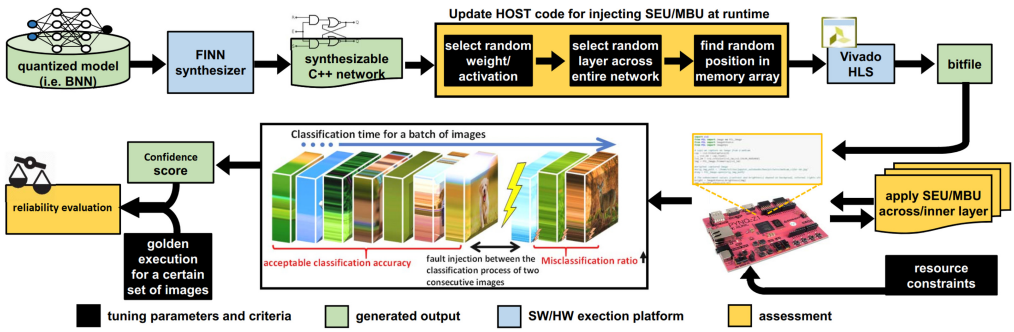


Fig. 12. Fiji-FIN framework for fault injection into FPGAs [102].

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]).

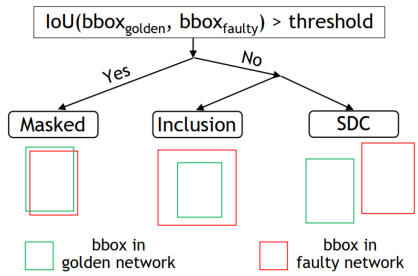


Fig. 13. Fault classification in the object detection task based on bounding boxes [137].

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

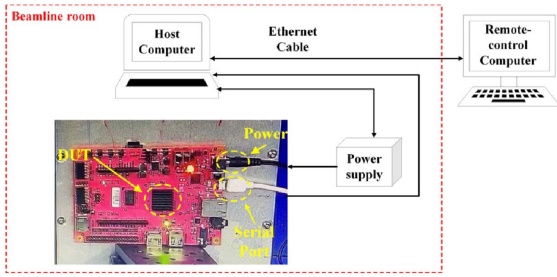


Fig. 14. Block diagram of the setup of beam experiment in Reference [108].

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

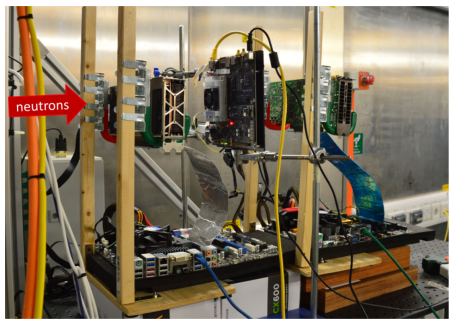


Fig. 15. Setup of neutron irradiation to GPU [10, 121, 136].

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 2

II

M. H. Ahmadilivani, M. Taheri, J. Raik, M. Daneshtalab, and M. Jenihhin. DeepVigor: VulnerabIlity Value RanGes and FactORs for DNNs' Reliability Assessment. In *IEEE European Test Symposium (ETS)*, pages 1–6. Venice, Italy, 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].

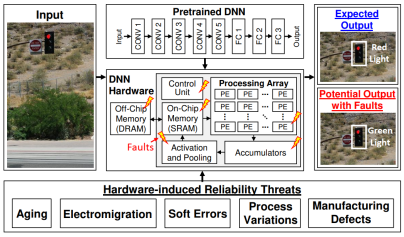


Fig. 1: Hardware reliability threats in DNN accelerators and their impact on the output [1].

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.

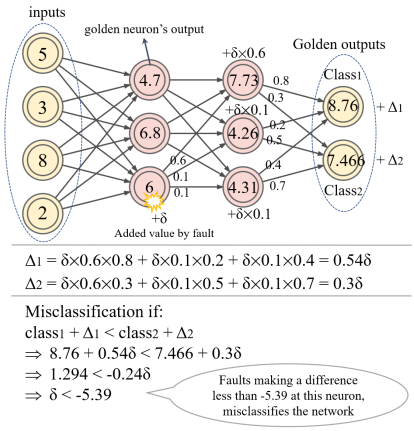


Fig. 2: An example of fault propagation analysis model and finding the vulnerability value ranges for a neuron with a given input.

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:

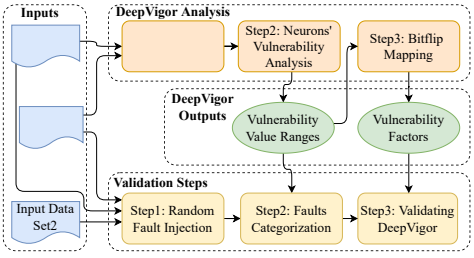


Fig. 3: Steps of the DeepVigor method for DNNs' reliability assessment and its validation.

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 $1 - threshold$.

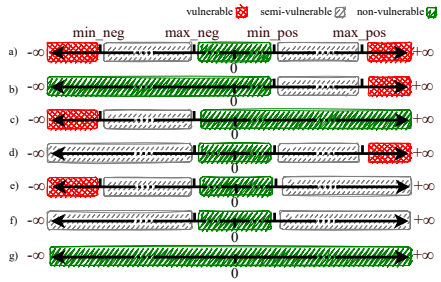


Fig. 4: Different possible cases of vulnerability ranges for each class in a neuron.

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.

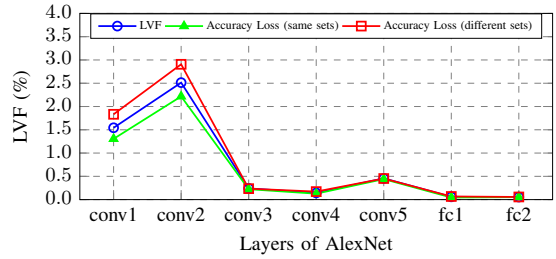


Fig. 5: Correlation between LVF and accuracy loss.

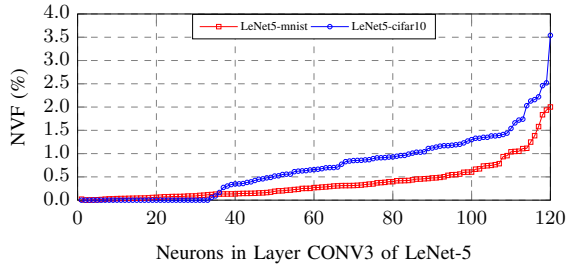


Fig. 6: NVF of neurons in CONV3 for LeNet5-mnist and LeNet-cifar10.

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 3

III

M. H. Ahmadilivani, J. Raik, M. Daneshtalab, and M. Jenihhin. Deepvigor+: A Scalable, Accurate and Automated Framework for Resilience Analysis of Deep Neural Networks. *Under review*, pages 1–14, 2024

DeepVigor+: Scalable and Accurate Semi-Analytical Fault Resilience Analysis for Deep Neural Networks

Mohammad Hasan Ahmadilivani[1], Jaan Raik[1], Masoud Daneshtalab[1,2], Maksim Jenihhin[1]
[1]Tallinn University of Technology, Tallinn, Estonia
[2]Mälardalen University, Västerås, Sweden

Growing exploitation of Machine Learning (ML) in safety-critical applications necessitates rigorous safety analysis. Hardware reliability assessment is a major concern with respect to measuring the level of safety. Quantifying the reliability of emerging ML models, including Deep Neural Networks (DNNs), is highly complex due to their enormous size in terms of the number of parameters and computations. Conventionally, Fault Injection (FI) is applied to perform a reliability measurement. However, performing FI on modern-day DNNs is prohibitively time-consuming if an acceptable confidence level is to be achieved. In order to speed up FI for large DNNs, statistical FI has been proposed. However, the run-time for the large DNN models is still considerably long.

In this work, we introduce DeepVigor+, a scalable, fast and accurate semi-analytical method as an efficient alternative for reliability measurement in DNNs. DeepVigor+ implements a fault propagation analysis model and attempts to acquire Vulnerability Factors (VFs) as reliability metrics in an optimal way. The results indicate that DeepVigor+ obtains VFs for DNN models with an error less than 1% and 14.9 up to 26.9 times fewer simulations than the best-known state-of-the-art statistical FI enabling an accurate reliability analysis for emerging DNNs within a few minutes.

I. Introduction

In recent years, the rapid advancement of Artificial Intelligence (AI), in particular of Machine Learning (ML), has provided a unique opportunity for automation leading to AI exploitation in safety-critical applications [1], [2]. Deep Neural Networks (DNNs) are dominantly employed in ML for complex tasks such as image classification and object detection that have applications in the safety-critical domain [3], [4]. According to recent regulations such as the European Union's AI ACT [5] and the US National AI Initiative Act [6], AI systems that may cause adverse impacts on people's safety are considered high-risk and must be rigorously assessed before deployment. This reflects the importance of early-stage safety risk assessment for AI deployment.

One of the major concerns in safety-critical applications is the hardware reliability of systems, which are threatened by hardware faults and errors [7]. With the technology miniaturization, the rate at which hardware faults occur is continuously rising in logic and memory circuits since the sensitivity of transistors increases against soft errors, temperature variations, etc. [8], [9], [10]. As DNNs are penetrating such applications e.g., perception in automotive [11], their hardware reliability assessment is necessary during the design and development phase, not only for risk certification but also for eliminating reliability weaknesses as early as possible.

On the other hand, the size of DNNs, in particular Convolutional Neural Networks (CNNs), in terms of memory footprint and number of operations is rapidly growing [12], [13]. Fig. 1 presents the growing model size in terms of the number of MAC operations and parameters for emerging DNNs and Vision Transformers (ViT), leading to their higher accuracy [14]. This trend necessitates leveraging more complex and performant computing systems such as GPUs, TPUs, etc. [15]. The fact that these complex systems are prone to hardware faults [16], [17], [18] leads to a necessary and complex task of their hardware reliability assessment prior to deployment in safety-critical applications.

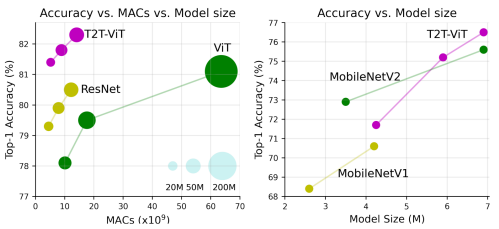


Fig. 1: Growing size of emerging DNN models regarding computation and memory requirement [14].

Hardware faults result in the alteration of values at the application level and might lead to erroneous outputs in a digital system. Hardware faults could lead to catastrophic consequences when exploiting hardware DNN accelerators in safety-critical applications. As an example, a hardware fault may result in misclassifying a red traffic light at an intersection in an autonomous vehicle and jeopardize the life of its passengers, as shown in Fig. 2, generated by AI.

Fig. 2: An example of hardware faults effect on an autonomous vehicle, generated by https://getimg.ai.

In this regard, several research works have been carried out to assess and enhance DNNs' hardware reliability against hardware faults listed in the related surveys [19], [16], [20], [21], [22]. Hardware reliability of DNN accelerators is the likelihood of a DNN accelerator performing as it is trained during the deployment [23]. The source of reliability threats is hardware-originated (soft errors, electromagnetic, temperature variation, aging, etc.) influencing a DNN's functionality by changing its parameters (e.g., weights) or activation values during run-time.

To ensure a reliable DNN deployment, the first step is to extensively evaluate the performance and functionality of pre-trained DNNs against hardware faults. To achieve this objective, three main approaches are identified throughout the state-of-the-art, in a prior systematic literature review [22]: 1) Fault Injection (FI) methods in which faults are injected into the target system and simulated, 2) Analytical methods where faults effects are mathematically analyzed, and 3) Hybrid methods that combine FI and analytical methods.

It was observed that the majority of existing research works adopt FI (based on simulation, emulation, or irradiation). In FI experiments, faults can be accurately modeled, and the behavior of a system is evaluated in the presence of faults. However, given the growing size of emerging DNNs and their accelerators [15], [24], obtaining a fully precise evaluation is unfeasible and impractical.

Throughout the literature, multiple research works are presented to significantly reduce the simulation complexity of FI experiments for DNNs while maintaining statistical accuracy. It includes software simulation [25], [26], [27], [28], or hardware emulation [29], [30], [31]. Nonetheless, any FI experiment requires a certain level of statistical confidence, which is obtained by a considerably high number of repetitions and a multitude of fault locations [26]. Therefore, FI-based approaches for reliability assessment are inherently unscalable and take days and weeks using powerful computing resources such as GPU.

On the other hand, analytical approaches are presented to tackle the scalability issue. Nonetheless, they cannot provide reliability-oriented metrics including Vulnerability Factors (VFs) which is an essential metric for reliability evaluation. Moreover, the accuracy of analytical methods is not comparable to that of FI-based methods. To tackle the existing issues in the literature, a prior semi-analytical method named DeepVigor [23] is proposed as an alternative to simulation-based FI. DeepVigor provides accurate VF for bits, neurons, and layers of CNNs demonstrated to be faster than FI by utilizing a fault propagation analytical method. However, it analyzes all neurons in a CNN, implying a huge search space to obtain vulnerability values, leading to a scalability issue for large and deep emerging CNNs.

In this paper, we propose a scalable, fast, and accurate alternative to FI named DeepVigor+, which provides VF for layers and models of DNNs. DeepVigor+ takes advantage of an optimal fault propagation analysis within neurons and entire DNNs to derive VFs in an optimized way. To the best of our knowledge, DeepVigor+ is the first scalable semi-analytical alternative to FI with a comparable accuracy for resilience analysis of DNNs in the literature. The contributions of this paper are as follows:

- Introducing a scalable, fast and accurate resilience analysis method for DNNs called DeepVigor+ for deriving VFs for DNNs' layers and models by analyzing both parameters and activations as well as by exploiting an optimal error propagation analysis. DeepVigor+ conducts a novel statistical approach based on stratified sampling unleashing a fast and accurate resilience analysis for large and deep emerging DNNs,
- Extensively demonstrating the effectiveness of DeepVigor+ by examining its error compared to FI, leading to meeting a 1% mean absolute error in obtaining VF. Furthermore, its scalability is evidenced against statistical FI, resulting in 14.9 to 26.9 times fewer simulations than a cutting-edge state-of-the-art statistical FI approach. Derived VF by DeepVigor+ can provide the possibility of reliability comparison and visualization between and within various DNN models in a few minutes for deep and large DNNs.

DeepVigor+ is presented as an open-source tool (https://github.com/mhahmadilivany/DeepVigor), unlocking reliability analysis for emerging DNNs and enabling researchers to quickly assess their reliability and design and develop fault-tolerant DNNs.

Table I presents the abbreviations frequently used in the manuscript. In the rest of the paper, Section II highlights the gap by overviewing the related works on statistical FI as well as analytical methods. Section III presents the methodology of DeepVigor+ leading to fast and accurate reliability analysis for CNNs. Section IV explains how DeepVigor+ is implemented and describes the experiments, and their results are presented in detail in Section V to demonstrate the efficiency of DeepVigor+. Finally, Section VI concludes the paper.

TABLE I: Frequent abbreviations used in this manuscript.

Abbreviation	Description
DNN	Deep Neural Network
CNN	Convolutional Neural Network
FI	Fault Injection
SFI	Statistical Fault Injection
VF	Vulnerability Factor
CVF	Channel Vulnerability Factor
LVF	Layer Vulnerability Factor
MVF	Model Vulnerability Factor
OFMap	Output Feature Map
IFMap	Input Feature Map
VVSS	Vulnerability Values Search Space
EDM	Error Distribution Map
CVV	Candidate Vulnerability Value
VVR	Vulnerability Value Range
MAE	Mean Absolute Error

II. Related Works: Resilience Analysis Methods for DNNs

With the fast growth of emerging DNNs' size regarding their memory and computation requirements, their reliability analysis becomes prohitively complex, time-consuming and resource-hungry. In a recent literature review [22], it has been observed that the majority of papers studying the reliability of

DNNs exploit a Fault Injection (FI) approach. Software-level simulation FI is widely adopted in reliability analysis due to its low design time and fast execution.

Multiple open-source tools for software-level simulation-based reliability analysis are proposed in the literature, and the most adopted ones include PytorchFI [27] and TensorFI [32], [33], enabling FI simulations in PyTorch and TensorFlow respectively. These software-level FI tools support various fault models and provide different metrics for the resilience analysis of DNNs regarding parameters and activations.

Any FI simulation requires multiple simulations while a random set of parameters/activations are faulty and propagated through the forward execution of a DNN. Then, the erroneous outputs are observed, and reliability metrics are acquired. The number of simulations affects the accuracy of the obtained metrics. In exhaustive FI, all possible bitflips in the fault space are flipped individually, leading to a huge simulation space, which is impractical since DNNs possess millions of parameters and activations. Statistical Fault Injection (SFI) attempts to reduce the number of simulations based on statistical analysis, while guaranteeing a certain maximum error margin.

For conventional computing hardware, the number of simulations in an SFI experiment used to be specified by Eq. (1), where n is the number of samples, N is total fault space, e is the error and t is determined based on the expected confidence-level [34].

$$n = \frac{N}{1 + e^2 \times \frac{N-1}{t^2 \times p(p-1)}} \tag{1}$$

However, it is shown that applying Eq. (1) to the entire DNN does not result in statistically accurate results [26]. Authors in [26] introduced multiple methods to improve the accuracy of SFI as well as to reduce the number of simulations. Accordingly, layer-wise SFI, data-unaware SFI and data-aware SFI were presented, which generally apply Eq. (1) to each layer separately. Data-unaware SFI and data-aware SFI methods consider FI at the bit level to improve the statistical experiments. In data-unaware $p = 0.5$ for all bits in Eq. (1), whereas in data-aware SFI requires a pre-analysis to specify the p for each bit position in the data representation.

Yet, SFI requires hundreds of thousands of simulations in any FI method leading to a significantly long simulation time even with leveraging GPUs. Furthermore, the growth of emerging DNNs necessitates an increasing number of simulations. Therefore, reliability analysis requires a paradigm shift in resilience analysis. Analytical methods for reliability analysis of DNNs are proposed to address its scalability issue.

Throughout the literature, multiple techniques are proposed to obtain the resilience of CNNs based on non-FI approaches, called analytical methods [22]. Layer-wise Relevance Propagation (LRP) is proposed to derive the contribution of neurons to the classification output of CNNs expressing their criticality in the presence of faults and errors [35]. The gradient-based analysis is exploited in [36], [37], [38], [39], [40] to obtain the resilience of CNNs against faults. Nonetheless, gradients neither represent the impact of faults on misclassification nor result in a probabilistic outcome of faults' effect on outputs. Also, faults may change the values significantly, while gradients represent small variations in erroneous values.

Therefore, the existing analytical approaches do not result in accurate metrics for the reliability analysis of DNNs. DeepVigor was proposed to address this gap in the literature to provide fast and accurate metrics for the resilience of CNNs against faults [23]. It provides a Vulnerability Factor (VF) for bits, neurons and layers in CNNs by acquiring vulnerability values for each neuron (i.e., output feature maps). Vulnerability values represent how much a fault should change the golden value of a neuron's output to misclassify CNN's golden classification. Although DeepVigor is demonstrated to be faster than FI due to its fault propagation analysis method, it requires a high execution time to obtain VFs. The reason is that DeepVigor attempts to analyze all neurons in a CNN as well as to find the vulnerability values in a huge space of numbers.

This paper proposes DeepVigor+, a new method to quickly obtain VF metrics for layers and DNN models addressing both scalability and accuracy of fault resilience analysis in the literature. DeepVigor+ employs an optimized error propagation analysis in neurons, assuming that a single fault might happen either at its inputs or weights, thus leading to effectively shrinking the search space for vulnerability values. Moreover, DeepVigor+ proposes a stratified sampling to further accelerate the process of resilience analysis without a tangible analysis accuracy mitigation, leading to obtaining VF in a few minutes, even for deep and large emerging DNNs.

III. Methodology: Scalable and Accurate Resilience Analysis for DNNs by DeepVigor+

In this section, the proposed resilience analysis method for DNNs, DeepVigor+, is presented. First, the fault model and its effect on 32-bit floating point values are described. Then, the mathematical model for fault propagation in DNNs is explained, which forms the basis for the DeepVigor+ analysis.

A. Fault Model

Transistor miniaturization leads to more efficient computational digital systems, whereas its reliability side-effects are becoming increasingly profound. With transistor scaling soft error rates increase significantly posing reliability concerns, in particular in the deployment of safety-critical applications [41], [42], [43]. Single Event Upset (SEU) due to soft errors may affect memory cells and flip a bit in a stored value. SEU in DNN accelerators affects either the parameters of a DNN (i.e., weights) or its activations (i.e., layers' inputs), which are stored in memory elements such as registers or on-chip memories [16], [22].

For reliability analysis, the multi-bit fault model is more realistic, however, it requires a huge fault space to consider all combinations, i.e., 3^{n-1} combinations where n is the number of bits. On the other hand, it is demonstrated that the single-bit fault model provides a high fault coverage of multiple-bit faults [44]. Therefore, analyzing the reliability of DNNs based on a single-bit fault model is valid for obtaining the VF of models. This fault model is in line with the other works in the literature [23], [25].

Therefore, in this paper we build our analysis based on the single-bit fault model in weights and activations separately. We assume that a single parameter or a single input to a layer is faulty which is propagated to the output of the corresponding layer. By mathematical analysis, we model the fault propagation to the output of neurons and DNNs, resulting in calculating VF for them.

B. Single Fault Analysis in 32-bit Floating-Point

To model the behavior of faults for the analysis, we assume that at an inference, a single fault may influence the value of a single input activation to a neuron or a single parameter in a CNN. First, we analyze the effect of a single bitflip on the 32-bit floating-point data representation to comprehend how a value might change due to a single fault. IEEE-754 32-bit floating-point data type is shown in Fig. 3. It contains 1 sign bit, 8 exponent bits, and 23 fraction bits, and represents a number based on Eq. (2).

$$number = (-1)^{sign} \times 2^{E-127} \times (1 + \sum_{i=0}^{i=23} b_{23-i} \times 2^{i}) \quad (2)$$

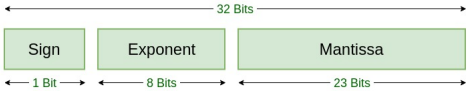


Fig. 3: 32-bit floating point IEEE-754 data representation.

In this regard, we consider the following lemmas:

- *Lemma 1:* Any bitflip from 1 to 0 in a value decreases the value as ϵ while $\epsilon < 0$, and any bitflip from 0 to 1 increases it as ϵ while $\epsilon > 0$.
- *Lemma 2:* In the 32-bit floating-point data representation, an error (ϵ) induced to a value (x) by a bitflip in bit i can be represented as in Eq. (3), whether the bitflip is in sign bit ($2 \times x$), exponent bits ($2^i \times x$), or mantissa bits (2^i), as stated in [45], [46].

$$\begin{cases} x_{faulty} = x + \epsilon \\ \epsilon \in \{2 \times x, 2^{i-23} \times x, 2^i\} \end{cases} \quad (3)$$

- *Lemma 3:* To unify the analysis of the error induced to a value by a single bitflip, we can approximate ϵ by representing it as a power of two, for each section of the data representation, as follows:
  1) Sign: approximate ϵ as $\pm 2^{log_2(2 \times x)}$.
  2) Exponent: approximate ϵ as $\pm 2^{i-23}$, assuming that x is a small value.
  3) Mantissa: approximate ϵ as $\pm 2^i$.

  In all cases, ϵ can be approximated as the nearest value of the power of two to the actual ϵ. This approximation can lead to a unified representation for ϵ. To that end, ϵ might be negative or positive (based on Lemma 1) and might be small or big (based on Lemma 2). Eventually, we can express it as a unified representation as shown in Eq. (4).

$$\epsilon \approx \pm 2^{\rho}; \rho \in \{\pm 1, \pm 2, \pm 3, ...\} \quad (4)$$

- *Lemma 4:* When a faulty value is used in a multiplication operation in CNNs, the erroneous output can also be approximated. It has been observed that the parameters in CNNs are mainly distributed around 0 and in the range of $[-1, 1]$ [47]. In order to approximate the error of multiplying two values when one of them is faulty, we can analyze it based on Eq. (5), where x and y are fault-free values and x' is the faulty value after a bitflip in x.

$$\begin{aligned} let: \quad & x' = x + \epsilon \\ then: \quad & x' \times y = x \times y + \epsilon \times y \\ & \Rightarrow x' \times y = x \times y + \delta \end{aligned} \quad (5)$$

In Eq. (5), when x is a small value, $\delta \approx 0$. Since most values in CNNs' operations are close to 0, the erroneous values in multiplications in a CNN can be approximated based on the unified error representation, as shown in Lemma 3 and Eq. (4).

C. Single Fault Error Propagation in DNNs

We analyze the single bitflip error propagation in CNNs considering their effect on the values of numbers. Each neuron in a convolutional (CONV) layer operates as shown in Eq. (6), where the Output Feature Map (OFMap) in lth layer and kth channel is obtained by the summation of multiplications between weights ($\hat{w}$) and Input Feature Map (IFMap, $\hat{x}$) plus bias (b). In CONV layers, $\hat{w}$ and $\hat{x}$ are 3-Dimensional (3D) $c_{in} \times n \times n$ arrays, where c_{in} is the number of inputs channels to the layer, and n is the kernel size.

$$OFMap^{l}_{k}(\hat{x}, \hat{w}, b) = (\sum_{i=0}^{c_{in}} \sum_{j=0}^{n} \sum_{k=0}^{n} x_{ijk} \times w_{ijk}) + b \quad (6)$$

Here, we assume that a fault affects a single IFMap in a single neuron, thus, it produces a single erroneous OFMap. Supposing that a fault occurs in an IFMap x_{ijk}, represented as x'_{ijk}. The fault introduces an error ϵ to the fault-free value of x_{ijk}. Therefore, it can be expressed in Eq. (7).

$$x'_{ijk} = x_{ijk} + \epsilon \quad (7)$$

Hence, once a bitflip occurs in $\hat{x}$ in Eq. (6), the partial multiplication is computed in Eq. (8). In this equation, the term $x_{ijk} \times w_{ijk}$ represents the fault-free multiplication, and $\epsilon \times w_{ijk}$ is an added error to the output by a fault which can be represented as δ.

$$\begin{aligned} x'_{ijk} \times w_{ijk} &= (x_{ijk} + \epsilon) \times w_{ijk} \\ &= x_{ijk} \times w_{ijk} + \epsilon \times w_{ijk} \\ &= x_{ijk} \times w_{ijk} + \delta \end{aligned} \quad (8)$$

Consequently, the erroneous OFMap can be expressed in a way that it is the summation of the fault-free OFMap and δ, while the single faulty IFMap in $\hat{x}$ can be in any index of the corresponding 3D array. Considering Lemma 4, the induced error at the neuron's OFMap can be expressed in Eq. (9).

$$OFMap^l_k(\hat{x}', \hat{w}, b) = OFMap^l_k(\hat{x}, \hat{w}, b) + \delta$$
$$\delta \approx \pm 2^\rho; \rho \in \{\pm 1, \pm 2, \pm 3, ...\} \quad (9)$$

On the other hand, when a fault occurs in a weight, it has the same effect on a single neuron's output. However, the same faulty weight in the corresponding CONV layer is used by all neurons in the layer, as filters slide over all IFMaps. Therefore, it results in an output channel in which all OFMaps are erroneous, as shown in Fig. 4. Considering Lemma 4, the fault propagation for an output channel can be expressed in Eq. (10).

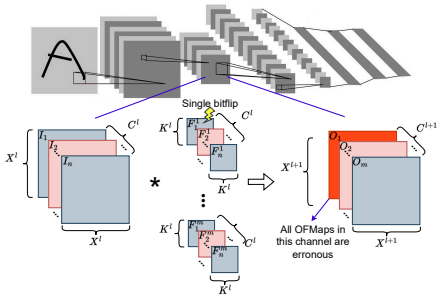


Fig. 4: Fault propagation in a CNN in the case of a single bitflip in a weight.

$$Out\_Channel^l_k(\hat{x}, \hat{w}, b) = \hat{x} * \hat{w} + b$$
$$Out\_Channel^l_k(\hat{x}, \hat{w}', b) = \hat{x} * \hat{w} + b + \epsilon * \hat{x}$$
$$Out\_Channel^l_k(\hat{x}, \hat{w}', b) = Out\_Channel^l_k(\hat{x}, \hat{w}, b) + \delta$$
$$\delta \approx \pm 2^\rho; \rho \in \{\pm 1, \pm 2, \pm 3, ...\} \quad (10)$$

Based on the aforementioned theoretical analysis, to analyze the effect of single faults on CNNs, regardless of the fault occurrence in activations or weights, we only need to identify an added value δ (as a power of two) at the output of the target neuron/channel where it misclassifies the golden class of the CNN for each input image.

D. DeepVigor+: Fault Resilience Analysis for DNNs

In this section, the steps of DeepVigor+ are presented. Fig. 5 illustrates the steps of the DeepVigor+ methodology for fault resilience analysis of DNNs against single faults. The method's objective is to provide the Vulnerability Factor (VF) for layers and the entire DNN model when a single fault happens in activations or weights.

The inputs of the method are a pre-trained CNN and validation data in a dataset. During the analysis, an OFMap or output channel is targeted, and multiple steps are taken to provide the VF metrics:

1) Single Bitflip Analysis: constructs a search space for possible vulnerability values and produces a distribution of erroneous values based on the approximations in Eq. (9) and (10),
2) Vulnerability Values Identification: obtains vulnerability values for the target OFMap/channel
3) Obtaining Vulnerability Factor: exploits identified vulnerability values and error distribution to provide VF for the target neuron/channel.

By obtaining the VF for the analyzed neurons and filters in a DNN, the VF for layers and the entire DNN can be derived. The details of each step are explained in the following. Noteworthy that each step presents a detailed description for analyzing a target neuron and then briefly links it to weights in filters.

Step 1 - Single Bitflip Analysis: As mentioned above, single faults at the inputs of a neuron are considered. This step conducts bitflips in the input activations of the target neuron to construct Vulnerability Values Search Space (VVSS) and Error Distribution Map (EDM) for the target neuron. VVSS is a set of candidate values representing all possible δ in Eq. (9). In other words, VVSS represents output errors produced by single bitflips in any bit locations of the inputs of the target neuron. EDM represents the distribution of δ in Eq. (9) throughout the approximated δ values with a power of two.

Obtaining VVSS and EDM requires performing a single bitflip for each input and multiplying it by its corresponding weight as shown in Eq. (8). Nonetheless, it is an exhaustive operation with high time complexity. This complexity can be remarkably reduced by leveraging Algorithm 1. In this algorithm, for each bitflip, we obtain all possible δ produced at the output of the neuron for all inputs at once.

Since each input might be faulty separately, to produce all errors in a neural operation, we first flip the ith bit in all inputs

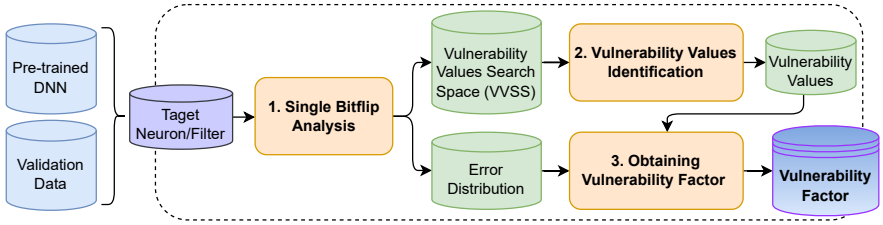


Fig. 5: An overview of the conducted steps in DeepVigor+.

(line 3) and then convert the binary representation to a value (line 4). Then, the difference of the *erroneous_inputs* with the fault-free input is computed (line 5), which represents all possible errors at inputs (ϵ in Eq. (8)) for all inputs added by a single bitflip in either of them. Then, a point-wise multiplication between the *erroneous_inputs* and *weights* results in producing all possible errors (δ in Eq. (9)) at the output (line 6).

Algorithm 1 Error Analysis for a Neuron

Input: Target neuron's inputs and weights as 3D matrices;
Output: All possible errors at the output;
Assume: δ is the error added to each golden output; All values are in 32-bit floating-bit;
1: binary_representation = float32_to_binary(inputs);
2: **for** $i \in [0, 31]$ **do**:
3: flip bit i in binary_representation;
4: erroneous_inputs = binary_to_float32(binary_representation);
5: input_errors = erroneous_inputs - inputs;
6: output_errors_list.append(input_errors $\odot$ weights);
7: **end for**;

Algorithm 1 produces all errors at the output of the target neuron. Thereafter, based on the presented theory in the subsections III-B and III-C, we generate the distribution of all produced errors as an Error Distribution Map (EDM) based on the Candidate Vulnerability Values (CVVs) which are a set of numbers as a power of two, shown in Eq. (11). Based on the experimental observations, we limit the CVV in the analysis between -2^{10} and 2^{10} for large values. The error values between -2^{-10} and 2^{-10} are merged as their propagation effects on CNNs are negligible and masked.

$$CVV = \{\pm 2^{\rho}\}; \rho \in \{0, \pm 1, \pm 2, \pm 3, \cdots, \pm 9, \pm 10\} \quad (11)$$

EDM contains the *distribution ratio* of the existing values over the values in *output_errors_list* in Algorithm 1, between each consecutive values in CVV, as shown in Eq. (12). EDM provides the ratio of error distribution with respect to each candidate vulnerability value, demonstrating how much each CVV represents the produced errors at the output of the target neuron. Based on the EDM, each CVV whose *distribution ratio* is not zero, is added to the Vulnerability Value Search Space (VVSS). This means that VVSS contains a subset of CVV that are the approximated errors produced by single faults in the inputs and might lead to misclassification at the output of the CNN.

$$\forall i \in CVV; distribution\_ratio_i = \frac{count(CVV_{i-1} < output\_errors\_list < CVV_i)}{count(output\_errors\_list)} \quad (12)$$

The process of single bitflip analysis for a target filter is similar to the one for a neuron. Nonetheless, in the case of a bitflip in a filter, all OFMaps in an output channel are affected (as described in subsection III-C and Eq. (10)). Therefore, this step is performed separately on every bit of all weights in the target filter, so that *VVSS* and *EDM* for the corresponding filter are obtained.

Step 2: Vulnerability Values Identification: This step exploits VVSS to identify the vulnerability values for the target neuron (i.e., OFMap) by exploring its constructed VVSS. As mentioned, VVSS represents output errors induced by a single fault occurring at the inputs of the target neuron. The objective is to identify the *maximum negative* and *minimum positive* vulnerability values among the existing ones in VVSS for a neuron that misclassify the DNN's outputs from its golden classification for each input data.

To explore the vulnerability values efficiently, we divide them into four different exploration spaces:

1) $VVSS_{[-\infty,-1]}$: contains CVVs between $[-\infty, 1]$ that have a non-zero distribution ratio.
2) $VVSS_{(-1,0)}$: contains CVVs between $(-1, 0)$ that have a non-zero distribution ratio.
3) $VVSS_{(0,1)}$: contains CVVs between $(0, 1)$ that have a non-zero distribution ratio.
4) $VVSS_{[1,+\infty]}$: contains CVVs between $[1, +\infty]$ that have a non-zero distribution ratio.

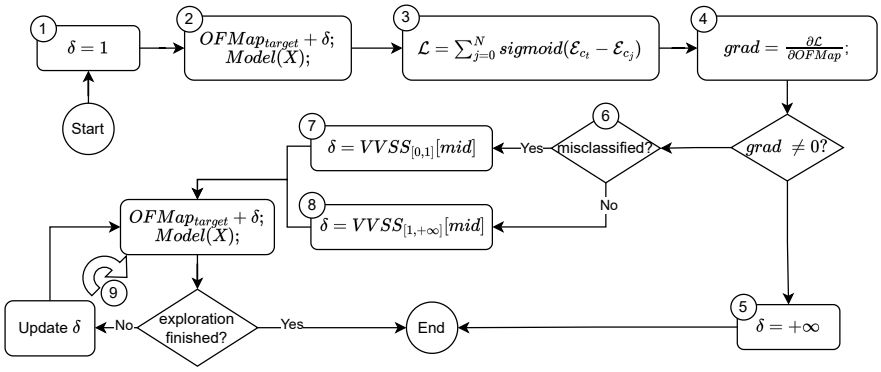


Fig. 6: Vulnerability value identification for a target neuron with a single input data for positive errors.

The flowchart in Fig 6 illustrates the algorithm of vulnerability value identification for *positive vulnerability values* for a single input data $\mathbf{X}$. In this flowchart, δ is the vulnerability value added to the target OFMap. First, δ equals 1 (box 1) is added to the target OFMap. Thereafter, the forward pass of the CNN is performed to obtain its output logits ($\mathcal{E}$) while a neuron is erroneous (box 2).

In box 3, a loss function $\mathcal{L}$ is calculated to obtain the effect of added δ to the target OFMap on the output classes based on the summation of the differences between all output class's logits from the golden class. In this loss function, $\mathcal{E}_{c_t}$ represents the erroneous output logit of the golden top class and $\mathcal{E}_{c_j}$ is the erroneous output logit of any other class.

In box 4, the gradient of loss function $\mathcal{L}$ w.r.t. the target OFMap is calculated to check if the CNN might be misclassified or not, when the target OFMap is erroneous. If the gradient is 0, faults producing positive deviation in the corresponding neuron do not lead to misclassification. In this case, the neuron's vulnerability value is considered the biggest CVV assumed value for positive numbers (i.e., 2^{10}) in box 5, and the algorithm ends. If the gradient is not equal to 0, a value can be found for δ in the target OFMap which misclassifies the DNN. The rest of the algorithm attempts to find the minimum positive δ in VVSS.

In box 6, it is examined if the CNN misclassifies the input from its golden classification when $\delta = 1$ for the target neuron, determining the initialization for δ in the next steps. If the input is misclassified when $\delta = 1$, it means that its vulnerability value is less than 1 and we should explore $VVSS_{(0,1)}$ (box 7), otherwise, $VVSS_{[1,+\infty)}$ should be explored (box 8). In the case of exploring $VVSS_{(0,1)}$, δ is set to the middle element of the set, and if it does not misclassify the golden output, the next bigger CVV should be explored. This process continues until the first value which misclassifies the output is found (loop 9). The final positive vulnerability value (δ^+) for the target neuron is the minimum value in positive VVSS that does not misclassify the input for CNN from its golden classification.

To identify the maximum negative vulnerability value (δ^-) that does not misclassify the input from its golden classification, the same procedure is conducted to explore negative values in VVSS. As a result, the Vulnerability Value Range (VVR) for the corresponding neuron is obtained and is expressed in Eq. (13). VVR represents all induced error values to the outputs of a neuron leading to a misclassification. Noteworthy that since bitflips in 32-bit floating-point might lead to large values, we assumed that any value larger than $2 * 10$ and less than 2^{-10} are critical for CNNs, therefore, these values represent $+\infty$ and $-\infty$ respectively.

$$VVR = (-\infty, \delta^-] \cup [\delta^+, +\infty) \tag{13}$$

The step of vulnerability values identification for a target filter is similar to the one for a neuron. It is conducted similarly to the illustrated flowchart in Fig. 6 to obtain VVR for the corresponding output channel resulting in VVR for each target channel in a DNN.

Step 3: Obtaining Vulnerability Factors: As mentioned, an Error Distribution Map (EDM) is obtained for the target neuron in step 1, representing the distribution ratio for each vulnerability value. On the other hand, the Vulnerability Value Range (VVR) is obtained in step 2. This step aims to map VVR to EDM to provide the VF for the target neuron, expressing the probability of misclassification in the case of a single bitflip in its inputs.

Fig. 7 depicts how VF can be obtained by mapping VVR to aggregated EDM. Aggregated EDM represents the summation of the *distribution ratio* for all values less than a CVV. As shown in Fig. 7, all values between (δ^-, δ^+) are non-vulnerable meaning that if an error deviates the OFMap as big as any value in this range, it will not misclassify the golden output. Otherwise, the error value is vulnerable leading to a misclassification.

VF is obtained by subtracting the distribution ratio of the minimum positive vulnerability value ($\delta+$) and the maximum negative vulnerability value (δ^-) as shown in Eq. (14). This equation expresses what portion of all errors produced at the output are critical for the CNN in terms of misclassification which is equivalent to the probability that a fault misclassifies the CNN's output.

$$VF_{target\_neuron} = 1 - (distribution\_ratio_{\delta+} - distribution\_ratio_{\delta-}) \tag{14}$$

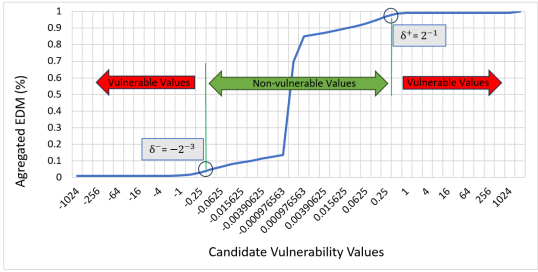


Fig. 7: Mapping obtained Vulnerability Value Range (VVR) to aggregated Error Distribution Map (EDM) for VF calculation.

The VF for a filter is obtained similarly. the EDM and VVR for the target filter are obtained in the previous steps. In this step, they are exploited to provide the VF for the target filter. Deriving the VF for individual neurons and filters leads to obtaining the VF for CONV layers and the entire CNN model. Since the analysis is performed for neurons and filters separately, it can provide separate VF for layers and for the entire model based on filters and neurons. The Layer Vulnerability Factor (LVF) is the average of VF for all activations/filters, and the Model Vulnerability Factor (MVF) is the average of LVFs within it.

Furthermore, to obtain the total MVF of the entire model using the obtained detailed VFs based on the activations and filters analysis, Eq. (15) is introduced, where L is the total number of layers in a CNN, N_l and W_l are the total number of output activations and weights in layer l. This equation is a layerwise weighted average of LVFs throughout the layers of a CNN including both activations and filters vulnerability

analysis, leading to the MVF for the entire model, representing the probability of misclassification if a fault occurs either in activations or weights.

$$MVF_{total} = \frac{\sum_{l=1}^{L}(\frac{N_l}{N_l+W_l} \times LVF_{act_l} + \frac{W_l}{N_l+W_l} \times LVF_{weight_l})}{L} \quad (15)$$

E. Stratified Random Sampling for Vulnerability Analysis:

The DeepVigor+ analysis can be conducted for all neurons/filters in a CNN. However, performing a complete analysis is obstructive and time-consuming, in particular for very deep and emerging CNNs. To address the scalability issue in resilience analysis for huge CNNs, we exploit the stratified sampling concept to reduce the sampling size for statistical DeepVigor+ analysis. Stratified random sampling is a method to increase the accuracy of estimation in sampling by dividing the population into subgroups called *strata* and random samples can be selected within them [48].

In DeepVigor+, the analysis is performed layer by layer. To take the benefits of stratified sampling for reducing the execution time of the analysis, we assume that each output channel in a layer is one stratum. Thereafter, a portion of all output channels specified by *channel_sampling_ratio* is considered and some random channels are selected. Within each channel, the number of analyzed neurons is determined by the $log_2(\#neurons)$ in the target channel, which is selected randomly. Therefore, the number of analyses can be described with Eq. (16).

$$\#analyzed\ neurons = \sum_{l \in layers} \lceil channel\_sampling\_ratio \times out\_channel_l \rceil \times log_2(\#neurons^l_{channel}) \quad (16)$$

IV. Experimental Setup

A. DeepVigor+ Implementation

DeepVigor+ is fully implemented using Python and the Pytorch library. The source code of DeepVigor+ is fully open-source in https://github.com/mhahmadilivany/DeepVigor as a tool to enable researchers and engineers to adopt it for the resilience analysis of DNNs. The user can obtain the VF for the CONV layers of a target CNN based on analyzing neurons or weights by determining it through some inputs. User can specify the following inputs for the analysis:

- Target pre-trained CNN: the tool loads the pre-trained set of weights.
- Dataset: the tool loads the validation data from the dataset.
- Analysis method: the tool performs resilience analysis for neurons (OFMaps) or filters (weights).
- Sampling method: the tool performs either complete analysis or stratified random sampling based on *channel sampling ratio*.
- Channel sampling ratio: in the case of stratified random sampling, the channel sampling ratio should be specified.

With the determined inputs, the tool performs the analysis and outputs the Layer Vulnerability Factor (LVF) for each layer and the Model Vulnerability Factor (MVF) for the entire CNN model. In the paper, we perform the DeepVigor+ analysis on one batch of 100 images in the test set. Note, that all experiments consider 32-bit floating-point data representation.

B. Validating DeepVigor+

We use Fault Injection (FI) to validate the results of DeepVigor+. To that end, we validate the VF for channels by analyzing filters of CNNs using complete DeepVigor+ and weights FI. In this regard, before an inference, a random 3D filter in a specified layer is selected. In the FI campaign, one bit of one weight in the target filter is flipped considering 32-bit floating-point data representation and the inference is performed on the same data as the DeepVigor+ analysis was conducted. The same FI process is carried out for all bits of weights in the target filter resulting in the Channel Vulnerability Factor (CVF) calculated by the ratio of output misclassifications compared to the golden outcomes. The obtained CVF in FI for each channel is compared with the one in complete DeepVigor+ for the corresponding channel, and their absolute errors are reported. In all DNNs under the experiment, 15% of all channels throughout the CONV layers are passed through the FI campaign for validation.

To show the accuracy of the result in exploited stratified sampling in DeepVigor+, we compare the obtained VF for channels (CVF) and layers (LVF) in sampling DeepVigor+ vs. complete DeepVigor+ for both activations and filters. The absolute difference between obtained VFs is derived to show the VF estimation in sampling analysis. In order to achieve more accurate results, different *channel_sampling_ratio*s are experimented including 5%, 10%, 15%, and 20%. Since neurons and channels are selected randomly in the stratified sampling, each sampled analysis is repeated 50 times and the maximum and mean absolute errors are reported.

Furthermore, to show the efficiency of DeepVigor+ analysis, we present the number of simulations for DeepVigor+ and how it outperforms FI. First, we compare the number of simulations for DeepVigor+'s complete analysis against exhaustive FI. Also, we compare the number of simulations for sampling DeepVigor+ with state-of-the-art Statistical FI (SFI) methods for CNNs proposed in [26], considering layer-wise SFI, data-unaware SFI, and data-aware SFI. Finally, the paper demonstrates the execution time for the complete and sampling DeepVigor+ on an NVIDIA A100 GPU to showcase the efficiency of the proposed method.

C. DNNs Under Study

In this paper, Deepvigor+ is executed and validated on six pre-trained DNNs using various datasets. DNNs under analysis include VGG-11 trained on CIFAR-10, VGG-16, ResNet-18-C and MobileNetV2 trained on CIFAR-100, ResNet-18-I and ResNet-34 trained on ImageNet. The baseline accuracy, number of channels and neurons for each DNN are shown in Table II. All experiments in this paper are performed on an NVIDIA A100 GPU accompanied by AMD EPYC 7742 64-core CPU.

TABLE II: The DNNs under study for DeepVigor+ validation.

DNN	Dataset	Baseline accuracy	# of conv layers	# of channels	# of neurons
VGG-11	Cifar-10	92.52%	8	2,752	232,448
VGG-16	Cifar-100	66.97%	13	4,224	185,344
ResNet-18-C	Cifar-100	70.26%	20	4,800	666,624
MobileNetV2	Cifar-100	61.27%	54	17,188	854,064
ResNet-18-I	ImageNet	69.19%	20	4,800	2,182,656
ResNet-34	ImageNet	73.04%	36	8,512	3,437,056

V. Experimental Results

A. DeepVigor+ Accuracy with FI

To analyze the accuracy of the obtained VFs by DeepVigor+, we perform fault injection experiments on 15% of randomly selected channels in all DNNs in Table II and present the Mean Absolute Error (MEA), as described in subsection IV-B. Table III presents the MEA results comparing the Channel Vulnerability Factor (CVF) by the complete DeepVigor+ filters analysis vs full FI into 15% of channels in each DNN. Noteworthy that the acceptable mean error is 1% [26].

The results in Table III indicate that the mean absolute error by DeepVigor+ compared to exact FI throughout the DNNs is between 0.819% to 0.978%, i.e., always less than 1%. These results demonstrate that DeepVigor+ is able to provide precise VFs and meets the expectation of an acceptable error with respect to exhaustive FI experiments.

TABLE III: Absolute error for CVF in DeepVigor+ and fault injection for 15% of the channels in DNNs.

DNN	VGG-11	VGG-16	ResNet -18-C	Mobile-NetV2	ResNet -18-I	ResNet -34
Mean Absolute Error (%)	0.819	0.938	0.933	0.874	0.978	0.878

The main sources of error in VF calculation in DeepVigor+ are:

1) As discussed in subsection III-C, DeepVigor+ approximates the error propagation in neurons based on the values of power of 2. This error approximation introduces an error to the resilience analysis which is also reflected in VF calculation.
2) DeepVigor+ assumes that bitflips resulting in big values are critical for DNNs and the *distribution ratio* for large values are always considered critical. However, in some cases, these values don't result in misclassification. This phenomenon is another reason for a minor difference between the VF by FI and DeepVigor+.

B. Sampling Analysis vs. Complete Analysis

This subsection presents the results for sampling DeepVigor+ and compares its VF results against the complete analysis to show how accurate sampling DeepVigor+ is. Channel Vulnerability Factor (CVF) and Layer Vulnerability Factor (LVF) are derived as described in subsection III-D and the maximum and mean absolute error for obtained CVFs and LVFs in complete and sampling DeepVigor+ for neurons and filters are presented separately. In sampling DeepVigor+, *channel sampling ratio* is explored.

Table IV indicates the absolute error results over various *channel sampling ratio* for DNNs under study, in both neurons and filters analysis. Each sampling analysis is repeated 50 times to observe the effect of random selections in stratified sampling. Based on the results, the minimum absolute error throughout the experiments for both CVF and LVF is very close to 0. As observed, the difference between obtained CVFs and LVFs throughout the results is minimal, demonstrating the effectiveness of the exploited stratified sampling.

In all experiments, the Mean Absolute Error (MAE) for CVF does not vary since the sampling method within channels is similar (i.e., the logarithm of the number of OFMaps). Also, it is observed that the MEA CVF is less than 0.065% in all experiments for both activations and filters sampling analysis. It means that the averaged VF for the logarithm-based random sampling within each channel results in a highly accurate CVF. This phenomenon is a result of the unified distribution of weights within a channel in a pre-trained DNN. The maximum observed error in neuron analysis is 0.004% to 1.443% throughout DNNs, whereas the mean error remains low, meaning that for most of the channels, obtained CVFs are highly accurate leading to an overall high accuracy for obtained CVFs.

On the other hand, *channel sampling ratio* directly affects the accuracy of LVF calculations. As observed in Table IV, LVF error decreases with the increase of *channel sampling ratio*. Considering both MEA and maximum error for LVF, a 10% *channel sampling ratio* can guarantee a minimal error for VF calculations. Based on the MEA of complete DeepVigor+ compared to exhaustive FI Table III, sampling DeepVigor+ with 10% channel sampling ratio ensures that all the overall error of obtained VFs for all DNNs will be lower than 1%.

In conclusion, the proposed stratified sampling in DeepVigor+ results in highly accurate VF for channels and layers obtained from both activations and filter analysis with a *channel sampling ratio* of 10%. Sampling DeepVigor+ results in a higher error than state-of-the-art statistic FI approaches such as data-aware and data-unaware, yet it meets the requirement of average error for resilience study which is 1%.

C. Run-Time and Scalability Investigation

To show the excellence of DeepVigor+ in terms of complexity, scalability and execution time against FI, first, we investigate the complexity of each based on the required number of simulations (i.e., forward pass executions) to obtain VF. Then we present the execution time for the complete and sampling DeepVigor+ on an NVIDIA A100 GPU.

Exhaustive FI is the most accurate method for determining precise VFs. In exhaustive FI, the required number of simulations equals the number of activations/weights times bit-width (i.e., 32 bits). Therefore, its complexity is proportional linearly to the size of DNNs. Whereas the complete DeepVigor+ analysis estimates VFs with high accuracy and significantly lower complexity. Although its complexity is affected by the size of DNNs, DeepVigor+ analysis exploits various

TABLE IV: Average absolute error analysis over 50 executions for sampling DeepVigor+ compared to complete analysis.

		Activations analysis				Filters analysis			
DNN	channel sampling ratio	MAE CVF	Max error CVF	MAE LVF	Max error LVF	MAE CVF	Max error CVF	MAE LVF	Max error LVF
VGG-11	5%	0.0004%	0.011%	0.0010%	0.012%	0.064%	0.171%	0.019%	0.125%
	10%	0.0004%	0.008%	0.0008%	0.008%	0.064%	0.148%	0.015%	0.097%
	15%	0.0004%	0.004%	0.0007%	0.007%	0.063%	0.130%	0.011%	0.083%
	20%	0.0004%	0.006%	0.0007%	0.007%	0.063%	0.128%	0.008%	0.076%
VGG-16	5%	0.037%	0.141%	0.033%	0.250%	0.045%	0.216%	0.017%	0.183%
	10%	0.038%	0.144%	0.022%	0.188%	0.044%	0.116%	0.010%	0.094%
	15%	0.038%	0.118%	0.018%	0.168%	0.043%	0.137%	0.008%	0.064%
	20%	0.038%	0.130%	0.015%	0.140%	0.043%	0.141%	0.007%	0.065%
ResNet-18-C	5%	0.029%	0.097%	0.054%	0.633%	0.045%	0.153%	0.016%	0.137%
	10%	0.029%	0.092%	0.038%	0.386%	0.045%	0.123%	0.011%	0.076%
	15%	0.029%	0.093%	0.031%	0.272%	0.045%	0.127%	0.010%	0.083%
	20%	0.029%	0.071%	0.026%	0.300%	0.045%	0.124%	0.008%	0.085%
MobileNetV2	5%	0.031%	1.443%	0.079%	2.803%	0.030%	0.893%	0.013%	0.983%
	10%	0.033%	0.841%	0.053%	1.252%	0.030%	0.401%	0.009%	0.372%
	15%	0.032%	0.586%	0.044%	0.742%	0.030%	0.468%	0.007%	0.346%
	20%	0.032%	0.465%	0.037%	0.720%	0.030%	0.401%	0.006%	0.320%
ResNet-18-I	5%	0.005%	0.071%	0.007%	0.168%	0.041%	0.115%	0.016%	0.114%
	10%	0.005%	0.070%	0.005%	0.212%	0.041%	0.096%	0.010%	0.061%
	15%	0.005%	0.054%	0.004%	0.108%	0.041%	0.083%	0.008%	0.062%
	20%	0.005%	0.045%	0.003%	0.082%	0.042%	0.089%	0.006%	0.032%
ResNet-34	5%	0.003%	0.061%	0.005%	0.140%	0.045%	0.136%	0.016%	0.087%
	10%	0.003%	0.059%	0.003%	0.084%	0.045%	0.127%	0.011%	0.097%
	15%	0.003%	0.054%	0.002%	0.089%	0.046%	0.122%	0.009%	0.080%
	20%	0.003%	0.048%	0.002%	0.063%	0.045%	0.120%	0.007%	0.061%

optimizations to reduce the number of simulations resulting in significantly lower complexity than exhaustive FI. This is evidenced by the results in Table V where the required number of simulations is compared between Exhaustive FI and complete DeepVigor+ analysis, for activations and filters separately.

TABLE V: Number of simulations for exhaustive FI vs complete DeepVigor+ for activations and filters analysis.

	Activations		Filters	
DNNs	Exhaustive FI	DeepVigor+	Exhaustive FI	DeepVigor+
VGG-11	7,438,336	1,636,414	294,967,296	11,035
VGG-16	5,931,008	2,018,447	470,734,848	13,704
ResNet-18-C	21,331,968	3,387,189	357,095,424	15,906
MobileNet-V2	27,330,048	3,029,125	74,176,512	42,234
ResNet-18-I	69,844,992	20,730,314	357,341,184	23,450
ResNet-34	109,985,792	28,821,489	680,564,736	43,098

As observed, for each DNN, DeepVigor+ complete analysis requires significantly lower executions either in activations or filters analysis. Throughout the results, activations analysis by complete DeepVigor+ is 2.93 to 9.02 times faster than exhaustive FI. According to our detailed investigations, activations analysis by DeepVigor+ requires 6 forward simulations per neuron, on average, throughout the DNNs under study. In this regard, the introduced loss function to obtain the vulnerability values (box 4 in Fig 6) contributes to skipping the analysis for up to 26% of neurons. Moreover, the vulnerability value can be obtained by a single forward simulation for up to 55% of neurons due to separating VVSS exploration (boxes 7 and 8 in Fig. 6).

In the complete filters analysis, the DeepVigor+ fault propagation modeling implies a huge impact on the number of simulations in the orders of magnitude. DeepVigor+ can derive VF for filters 1,756 to 34,350 times faster than exhaustive FI. To obtain the vulnerability values for channels, up to 1% of channels are skipped by leveraging the loss function, and up to 34% of channels require one forward simulation. These results indicate that complete DeepVigor+ provides accurate VF for neurons and channels of CNNs with significantly lower complexity and shorter execution time than exhaustive FI enabled by its optimal fault propagation modeling and analysis.

On the other hand, sampling DeepVigor+ is proposed to further reduce the execution time and complexity of resilience analysis and achieve a scalable method. To show its performance against statistical FI (SFI), Table VI compares the number of simulations for various state-of-the-art SFI [26] with sampling DeepVigor+. As presented, data-aware SFI leads to the least number of executions in FI-based simulation. It is observed that DeepVigor+ sampling activations analysis with 10% channel sampling ratio leads to 8.72 to 20.5 times fewer simulations compared to data-aware SFI. For the filters analysis, DeepVigor+ obtains their VF with 59.4 up to 96.2 times fewer simulations than data-aware SFI.

To obtain the Model Vulnerability Factor (MVF) both activations and filters should be analyzed separately, based on Eq. (15). Therefore, sampling DeepVigor+ accelerates the process from 14.9 up to 26.9 times throughout the DNNs. The scalability and speed of the method are achieved by both channel sampling ratio and logarithmic sampling within them. It can be observed that with the remarkable growth of DNNs under analysis in their number of parameters, the number of simulations in DeepVigor+ does not grow linearly.

To demonstrate the execution time of DeepVigor+ analysis, we employed an A100 NVIDIA GPU and performed sampling DeepVigor+ for activations and filters with different channel sampling ratios. Table VII and Table VIII present the average execution time over 50 executions of the method for complete and sampling DeepVigor+ with different channel sampling ratios, for activations and filters respectively. It is observed

TABLE VI: Comparison of required simulations for statistical FI [26] and sampling DeepVigor+.

	Activations Analysis				Filters Analysis			
Analysis method	Layer-wise	Data-unaware	Data-aware	Sampling DeepVigor+ 10%	Layer-wise	Data-unaware	Data-aware	Sampling DeepVigor+ 10%
VGG-11	74,863	1,562,657	71,173	**4,596**	75,351	2,141,913	66,934	**1,038**
VGG-16	117,992	1,839,889	106,513	**10,283**	123,266	3,588,834	112,151	**1,476**
ResNet-18-C	188,644	4,264,406	181,911	**15,996**	189,772	5,253,096	164,159	**1,706**
MobileNetV2	493,871	8,402,977	452,650	**22,077**	475,407	8,307,671	259,614	**4,364**
ResNet-18-I	191,205	5,407,659	189,325	**21,680**	190,896	5,358,315	167,447	**2,178**
ResNet-34	344,042	9,624,374	340,383	**39,002**	344,285	10,031,494	313,484	**4,003**

that VFs can be obtained in a few minutes for any DNNs. Considering 10% channel sampling ratio for both activations and filters analysis, the total MVF for VGG-11, VGG-16, ResNet-18-C, MobileNet-V2, ResNet-18-I and ResNet-34 is obtained almost in 8.7 minutes, 9.6 minutes, 12.7 minutes, 43.5 minutes, 19.4 minutes and 46 minutes, respectively. It is worth mentioning that the complete DeepVigor+ analysis for the DNNs under study takes almost 22 hours for VGG-16 and 18.5 days for ResNet-34, on the same GPU. Fast execution and accurate estimation of VF obtained by sampling DeepVigor+ analysis provide a remarkable opportunity for a high-speed resilience analysis for any DNN.

TABLE VII: Average execution time over 50 repetitions on A100 GPU for DeepVigor+ activations analysis, with different channel sampling ratios.

DNN	5%	10%	15%	20%	complete analysis
VGG-11	203 sec (≈ 3.4 min)	415 sec (≈ 6.9 min)	625 sec (≈ 10.4 min)	838 sec (≈ 13.9 min)	66,083 sec (≈ 0.7 days)
VGG-16	223 sec (≈ 3.7 min)	453 sec (≈ 7.5 min)	676 sec (≈ 11 min)	915 sec (≈ 15 min)	43,326 sec (≈ 0.5 days)
ResNet-18-C	284 sec (≈ 4.7 min)	580 sec (≈ 9.5 min)	876 sec (≈ 14.5 min)	1,171 sec (≈ 19.5 min)	94,939 (≈ 1.1 days)
MobileNetV2	1,037 sec (≈ 17 min)	2,097 sec (≈ 35 min)	3,153 sec (≈ 52.5 min)	4,071 sec (≈ 68 min)	211,868 (≈ 2.4 days)
ResNet-18-I	448 sec (≈ 7.4 min)	917 sec (≈ 15 min)	1,390 sec (≈ 23 min)	1,866 sec (≈ 31 min)	634,872 (≈ 7.3 days)
ResNet-34	1,100 sec (≈ 18 min)	2,237 sec (≈ 37 min)	3398 sec (≈ 56 min)	4549 sec (≈ 75 min)	1,402,362 (≈ 16.2 days)

TABLE VIII: Average execution time over 50 repetitions on A100 GPU for DeepVigor+ filters analysis, with different channel sampling ratios.

DNN	5%	10%	15%	20%	complete analysis
VGG-11	57 sec (≈ 0.95 min)	111 sec (≈ 1.8 min)	170 sec (≈ 2.8 min)	219 sec (≈ 3.6 min)	23,031 sec (≈ 0.26 days)
VGG-16	61 sec (≈ 1 min)	125 sec (≈ 2.1 min)	168 sec (≈ 3.1 min)	245 sec (≈ 4.1 min)	35,634 sec (≈ 0.4 days)
ResNet-18-C	92 sec (≈ 1.5 min)	184 sec (≈ 3 min)	275 sec (≈ 4.6 min)	367 sec (≈ 6.1 min)	60,410 sec (≈ 0.7 days)
MobileNetV2	257 sec (≈ 4.3 min)	513 sec (≈ 8.5 min)	759 sec (≈ 12.6 min)	1,025 sec (≈ 17.1 min)	39,713 sec (≈ 0.46 days)
ResNet-18-I	117 sec (≈ 1.9 min)	250 sec (≈ 4.1 min)	370 sec (≈ 6.1 min)	506 sec (≈ 8.4 min)	134,164 sec (≈ 1.5 days)
ResNet-34	259 sec (≈ 4.3 min)	524 sec (≈ 8.7 min)	795 sec (≈ 13.2 min)	1,064 sec (≈ 17.7 min)	201,305 sec (≈ 2.3 days)

D. Reliability Visualization and Comparison for DNNs

It has been shown that VF for DNNs' layers and the entire model can be obtained accurately in a few minutes by DeepVigor+. The obtained VF results by DeepVigor+ can be used to visualize the vulnerability of layers within a DNN and identify more vulnerable ones. Fig. 8 illustrates the LVF comparison for ResNet-18 trained on CIFAR-100 and ImageNet datasets, while total LVF for each layer is obtained based on the numerator of Eq. (15). This visualization sketches how much each layer is vulnerable to single faults compared to each other. As observed, in both ResNet18-C and ResNet18-I first layers are more vulnerable than the latter ones. Therefore, DeepVigor+ enables vulnerability visualization and comparison within a DNN.

Furthermore, DeepVigor+ results in total MVF based on Eq. (15) which is the weighted average of obtained LVFs providing a comprehensive examination of vulnerability between different DNNs. Fig. 9 indicates MVFs for activations and filters of DNNs, separately, as well as their total MVF. As a result, it is observed that activations are more vulnerable than weights. However, weights generally contribute more to the total MVF since their memory footprint is higher than that of activations in DNNs. Based on total MVF, VGG-16 is the least vulnerable DNN (MVF = 1.19%) and MobileNet-V2 is the most vulnerable one (MVF = 2.76%).

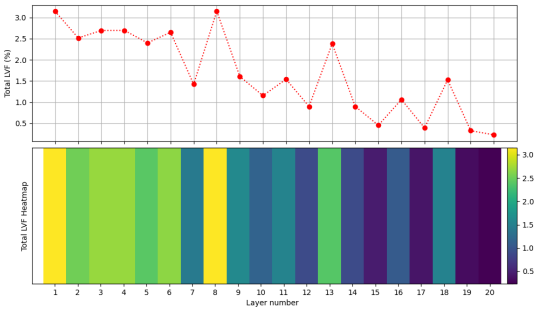
E. Impact of Input Data on the Quality of Results

To obtain the VFs in DeepVigor+, we considered one batch of 100 data. Nonetheless, to investigate the impact of data on the quality of analysis results, we repeat the experiments for different batches of data with the size of 100 input data and derive DNNs' total MVF. Fig. 10 illustrates the obtained total MVF for DNNs over 8 different batches of data. As observed, the variation between the total MVF for each DNN is negligible for different batches of data, demonstrating that analyzing DNNs' resilience with one batch of data provides confident results.

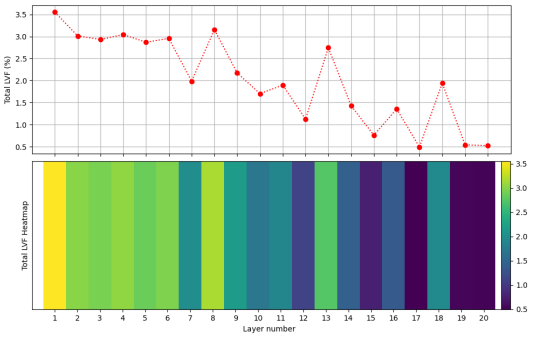
F. DeepVigor+ Applications and Considerations

As shown, DeepVigor+ achieves a fast, scalable and accurate resilience analysis for emerging DNNs. The analysis provided by DeepVigor+ is not limited to fault resilience assessment, but it can be exploited for designing fault-tolerant and resilient DNNs. Identifying more vulnerable channels and layers can lead to cost-effective selective fault mitigation techniques as well as a comparative investigation between different architectures of DNNs. Moreover, it enables design space exploration to identify more resilient DNNs against faults.

The other output of DeepVigor+ is VVR representing the values that a fault should affect neuron/weight to misclassify DNN's golden results. These values can be used for obtaining VFs and identifying more vulnerable components as well as for



(a) ResNet-18-C



(b) ResNet-18-I

Fig. 8: LVF visualization and comparison for ResNet-18 trained on a) CIFAR-100 and b) ImageNet.

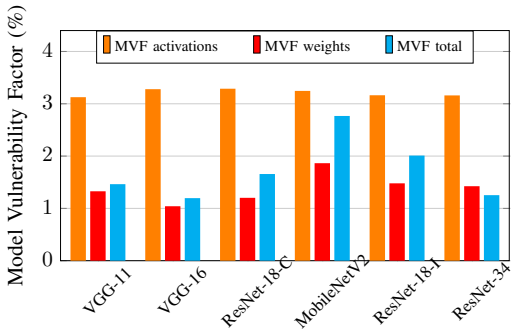


Fig. 9: MVF comparison for DNNs based on activations, filters and the entire model derived by DeepVigor+.

fault detection at inference and identifying bit-level resilience analysis since they are represented in the power of 2 values and can be mapped down to bits.

Yet, there are some constraints in this method to be considered and extended in future research:

- It is assumed that the parameters within the layers of DNNs under analysis are unifiedly distributed among channels and their places are not resorted after training. In such cases, a higher channel sampling ratio is needed to obtain accurate VF results.
- DeepVigor+ supports a single-bit fault model in input activations and weights of convolutional layers and obtained VF corresponds to this fault model. For multi-bit fault models, the corresponding error propagation should be applied.
- The error propagation analysis presented in this paper is based on 32-bit floating point data representation. However, the same concept can be extended and applied to fixed-point and integer data representations for QNNs resilience analysis, as in [49].

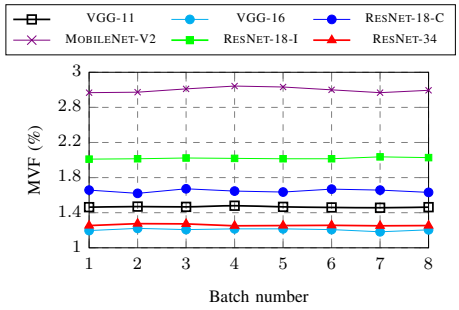


Fig. 10: Total MVF variation over different batches of data for all DNNs.

VI. Conclusion

This paper addresses one of the major challenges in fault resilience analysis for DNNs in the literature. It introduces DeepVigor+, the first semi-analytical scalable alternative method to fault injection, quantifying emerging DNN's resilience accurately in a short time. DeepVigor+ is facilitated by optimal fault propagation modeling in DNNs accompanied by stratified sampling tackling the scalability problem for resilience analysis for DNNs. This open-source method unleashes a fast resilience assessment, enabling fine-grain evaluation and design space exploration for various fault-tolerant and cost-effective designs for DNNs.

The results in the paper indicate that DeepVigor+ derives vulnerability factors for layers and the entire model of DNNs with less than 1% error, with 14.9 up to 26.9 times fewer simulations than the best-known state-of-the-art statistical FI. It is shown that DNN's Model Vulnerability Factor can be obtained within minutes by analyzing their activations and weights. DeepVigor+ is presented as an open-source tool for researchers and engineers to enable them to exploit it for DNNs' fault resilience assessment and enhancement.

References

[1] Y. Wang and S. H. Chung, "Artificial intelligence in safety-critical systems: a systematic review," *Industrial Management & Data Systems*, vol. 122, no. 2, pp. 442–470, 2022.

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

[3] J. Athavale, A. Baldovin, R. Graefe, M. Paulitsch, and R. Rosales, "Ai and reliability trends in safety-critical autonomous systems on ground and air," in *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*. IEEE, 2020, pp. 74–77.

[4] I. Moghaddasi, S. Gorgin, and J.-A. Lee, "Dependable dnn accelerator for safety-critical systems: A review on the aging perspective," *IEEE Access*, 2023.

[5] "Proposal for a regulation of the european parliament and of the council laying down harmonised rules on artificial intelligence (artificial intelligence act) and amending certain union legislative acts," https://data.consilium.europa.eu/doc/document/ST-5662-2024-INIT/en/pdf, 2024, [Online].

[6] "H.r.6216 - national artificial intelligence initiative act of 2020," https://https://www.congress.gov/bill/116th-congress/house-bill/6216, 2020, [Online].

[7] M. Rausand, *Reliability of safety-critical systems: theory and applications*. John Wiley & Sons, 2014.

[8] R. Baumann, "The impact of technology scaling on soft error rate performance and limits to the efficacy of error correction," in *Digest. International Electron Devices Meeting,.* IEEE, 2002, pp. 329–332.

[9] J. Henkel, L. Bauer, N. Dutt, P. Gupta, S. Nassif, M. Shafique, M. Tahoori, and N. Wehn, "Reliable on-chip systems in the nano-era: Lessons learnt and future trends," in *Proceedings of the 50th Annual Design Automation Conference*, 2013, pp. 1–10.

[10] S. Safari, M. Ansari, H. Khdr, P. Gohari-Nazari, S. Yari-Karin, A. Yeganeh-Khaksar, S. Hessabi, A. Ejlali, and J. Henkel, "A survey of fault-tolerance techniques for embedded systems from the perspective of power, energy, and thermal issues," *IEEE Access*, vol. 10, pp. 12 229–12 251, 2022.

[11] H. Y. Yatbaz, M. Dianati, and R. Woodman, "Introspection of dnn-based perception functions in automated driving systems: State-of-the-art and open research challenges," *IEEE Transactions on Intelligent Transportation Systems*, 2023.

[12] H. Hussain, P. Tamizharasan, and C. Rahul, "Design possibilities and challenges of dnn models: a review on the perspective of end devices," *Artificial Intelligence Review*, pp. 1–59, 2022.

[13] R. Desislavov, F. Martínez-Plumed, and J. Hernández-Orallo, "Trends in ai inference energy consumption: Beyond the performance-vs-parameter laws of deep learning," *Sustainable Computing: Informatics and Systems*, vol. 38, p. 100857, 2023.

[14] L. Yuan, Y. Chen, T. Wang, W. Yu, Y. Shi, Z.-H. Jiang, F. E. Tay, J. Feng, and S. Yan, "Tokens-to-token vit: Training vision transformers from scratch on imagenet," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2021, pp. 558–567.

[15] T. Mohaidat and K. Khalil, "A survey on neural network hardware accelerators," *IEEE Transactions on Artificial Intelligence*, 2024.

[16] 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.

[17] F. F. Dos Santos, L. Carro, and P. Rech, "Understanding and improving gpus' reliability combining beam experiments with fault simulation," in *2023 IEEE International Test Conference (ITC)*. IEEE, 2023, pp. 176–185.

[18] T. Garrett, S. Roffe, and A. George, "Soft-error characterization and mitigation strategies for edge tensor processing units in space," *IEEE Transactions on Aerospace and Electronic Systems*, 2024.

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

[20] 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.

[21] F. Su, C. Liu, and H.-G. Stratigopoulos, "Testability and dependability of ai hardware: Survey, trends, challenges, and perspectives," *IEEE Design & Test*, vol. 40, no. 2, pp. 8–58, 2023.

[22] 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.

[23] ——, "Deepvigor: Vulnerability value ranges and factors for dnns' reliability assessment," in *2023 IEEE European Test Symposium (ETS)*. IEEE, 2023, pp. 1–6.

[24] A. Canziani, A. Paszke, and E. Culurciello, "An analysis of deep neural network models for practical applications," *arXiv preprint arXiv:1605.07678*, 2016.

[25] 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*, 2019, pp. 1–23.

[26] A. Ruospo, G. Gavarini, C. De 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)*. IEEE, 2023, pp. 1–6.

[27] 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)*, 2020, pp. 25–31.

[28] O. Weng, A. Meza, Q. Bock, B. Hawks, J. Campos, N. Tran, J. M. Duarte, and R. Kastner, "Fkeras: A sensitivity analysis tool for edge neural networks," *Journal on Autonomous Transportation Systems*, 2024.

[29] 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.

[30] 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.

[31] 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.

[32] Z. Chen, N. Narayanan, B. Fang, G. Li, K. Pattabiraman, and N. DeBardeleben, "Tensorfi: A flexible fault injection framework for tensorflow applications," in *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2020, pp. 426–435.

[33] S. Laskar, M. H. Rahman, and G. Li, "Tensorfi+: a scalable fault injection framework for modern deep learning neural networks," in *2022 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. IEEE, 2022, pp. 246–251.

[34] 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.

[35] 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.

[36] J. Wang, J. Zhu, X. Fu, D. Zang, K. Li, and W. Zhang, "Enhancing neural network reliability: Insights from hardware/software collaboration with neuron vulnerability quantization," *IEEE Transactions on Computers*, 2024.

[37] C. Amarnath, M. Mejri, K. Ma, and A. Chatterjee, "Soft error resilient deep learning systems using neuron gradient statistics," in *2022 IEEE 28th International Symposium on On-Line Testing and Robust System Design (IOLTS)*. IEEE, 2022, pp. 1–7.

[38] ——, "Error resilience in deep neural networks using neuron gradient statistics," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2023.

[39] M. Sabih, F. Hannig, and J. Teich, "Fault-tolerant low-precision dnns using explainable ai," in *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*. IEEE, 2021, pp. 166–174.

[40] 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.

[41] I. Chatterjee, B. Narasimham, N. Mahatme, B. Bhuva, R. Reed, R. Schrimpf, J. Wang, N. Vedula, B. Bartz, and C. Monzel, "Impact of technology scaling on sram soft error rates," *IEEE Transactions on Nuclear Science*, vol. 61, no. 6, pp. 3512–3518, 2014.

[42] 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," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, 2021, pp. 641–653.

[43] 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.

[44] M. Bushnell and V. Agrawal, *Essentials of electronic testing for digital, memory and mixed-signal VLSI circuits*. Springer Science & Business Media, 2004, vol. 17.
[45] J. Elliott, F. Mueller, F. Stoyanov, and C. Webster, "Quantifying the impact of single bit flips on floating point arithmetic," North Carolina State University. Dept. of Computer Science, Tech. Rep., 2013.
[46] J. Zhan, R. Sun, W. Jiang, Y. Jiang, X. Yin, and C. Zhuo, "Improving fault tolerance for reliable dnn using boundary-aware activation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 10, pp. 3414–3425, 2021.
[47] Z. Huang, W. Shao, X. Wang, L. Lin, and P. Luo, "Rethinking the pruning criteria for convolutional neural network," *Advances in Neural Information Processing Systems*, vol. 34, pp. 16 305–16 318, 2021.
[48] R. Singh, N. S. Mangat, R. Singh, and N. S. Mangat, "Stratified sampling," *Elements of survey sampling*, pp. 102–144, 1996.
[49] 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.

Mohammad Hasan Ahmadilivani is a PhD student in the Computer Systems Department at Tallinn University of Technology (Taltech), Estonia. He earned his MSc in Computer Architecture Systems from the University of Tehran, Iran, in 2020. His research focuses on developing analytical methods to measure and enhance the hardware reliability of deep neural networks (DNNs). His research interests include exploiting DNNs in safety-critical applications, robust computer vision, and efficient and reliable DNN accelerator design.

Jaan Raik is a Full Professor at the Department of Computer Systems and Head of the Center for Dependable Computing Systems at the Tallinn University of Technology (Taltech), Estonia. He received his M.Sc. and Ph.D. degrees from Taltech in 1997 and in 2001, respectively. His research interests cover a wide area in electrical engineering and computer science domains including reliability of deep learning, hardware test, functional verification, fault-tolerance and security as well as emerging computer architectures. He has co-authored more than 400 scientific publications. He is a member of IEEE Computer Society, HiPEAC and of Steering/Program Committees of numerous conferences within his field. He served as the General Chair to IEEE European Test Symposium '25, '20, IFIP/IEEE VLSI-SoC '16, DDECS '12), Vice General Chair IEEE European Test Symposium '24, DDECS '13 and Program Co-Chair DDECS '23, '15, CDN-Live '16 conferences. He was awarded the Global Digital Governance Fellowship at Stanford (2022) and HiPEAC Paper Award (2020).

Masoud Daneshtalab Masoud Daneshtalab is currently a full Prof. at Mälardalen University in Sweden, Adj. Prof. at TalTech in Estonia and leads the Heterogeneous System research group. He is on the Euromicro board of directors, an editor of the MICPRO journal, and has published over 200 refereed papers. His research interests include HW/SW/Algorithm co-design, dependability and deep learning acceleration.

Maksim Jenihhin is a tenured associate professor of computing systems reliability and Head of the research group Trustworthy and Efficient Computing Hardware (TECH) at the Tallinn University of Technology, Estonia. He received his PhD degree in Computer Engineering from the same university in 2008. His research interests include reliable and efficient hardware for AI acceleration, methodologies and EDA tools for hardware design, verification and security, as well as nanoelectronics reliability and manufacturing test topics. He has published more than 170 research papers, supervised several PhD students and postdocs and served on executive and program committees for numerous IEEE conferences (DATE, ETS, DDECS, LATS, NORCAS, etc.). Prof. Jenihhin coordinates European collaborative research projects HORIZON MSCA DN "TIRAMISU" (2024), HORIZON TWINN "TAICHIP" (2024) and national ones about energy efficiency and reliability of edge-AI chips and cross-layer self-health awareness of autonomous systems.

Appendix 4

IV
M. H. Ahmadilivani, M. Taheri, J. Raik, M. Daneshtalab, and M. Jenihhin. Enhancing Fault Resilience of QNNs by Selective Neuron Splitting. In *IEEE 5th International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, pages 1–5. Hangzhou, China, 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.

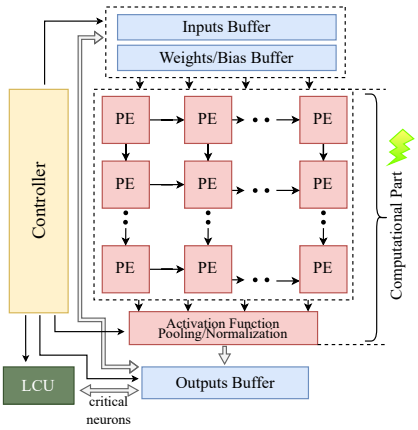


Fig. 1: An abstract view of the accelerator and where the faults may happen.

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

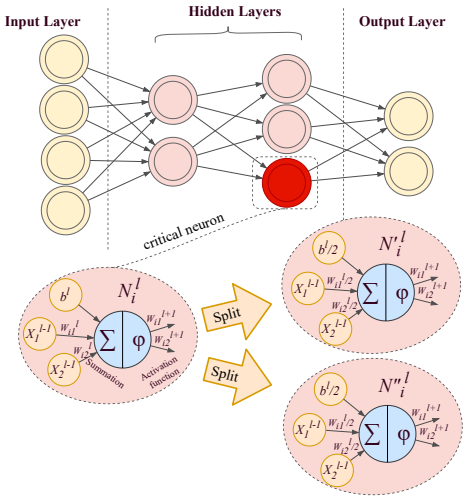


Fig. 2: Operation splitting for a neuron in a QNN involves halving the input parameters while keeping the output parameters non-modified. A critical neuron is replaced with its corresponding split neurons in the QNN.

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

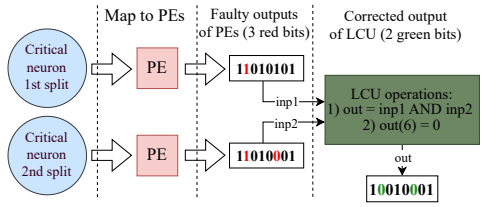


Fig. 3: An example of how LCU corrects faulty critical neurons.

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%

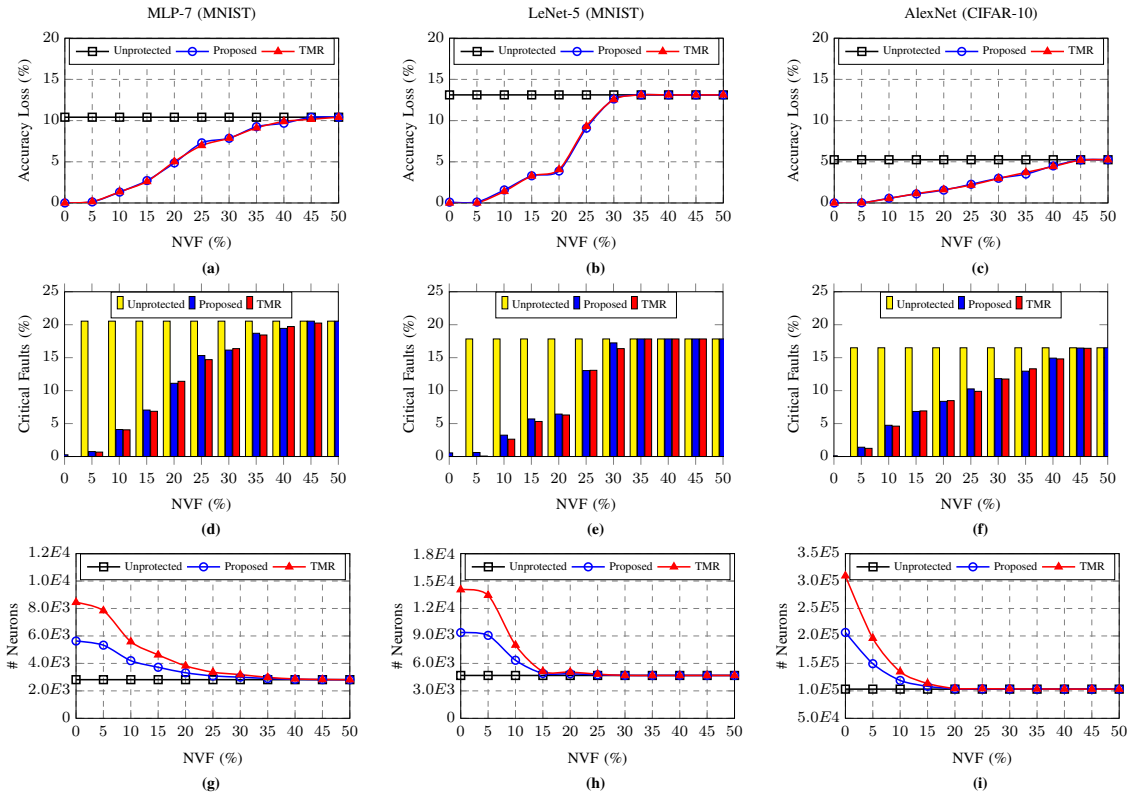


Fig. 4: QNNs comparison in terms of accuracy loss (a-c), critical faults (d-f), and network size (g-i) under different levels of protection: unprotected, proposed protection, and TMR, considering different thresholds for NVF from 0% to 50%.

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 5

V
S. Mousavi, M. H. Ahmadilivani, J. Raik, M. Jenihhin, and M. Daneshtalab. ProAct: Progressive Training for Hybrid Clipped Activation Function to Enhance Resilience of DNNs. *Under review*, pages 1–12, 2024

ProAct: Progressive Training for Hybrid Clipped Activation Function to Enhance Resilience of DNNs

Seyedhamidreza Mousavi[1], Mohammad Hasan Ahmadilivani[2], Jaan Raik[2], Maksim Jenihhin[2], and Masoud Daneshtalab[1,2]

[1]Mälardalen University, Västerås, Sweden
[2]Tallinn University of Technology, Tallinn, Estonia
[1]{seyedhamidreza.mousavi, masoud.daneshtalab}@mdu.se
[2]{mohammad.ahmadilivani, jaan.raik, maksim.jenihhin}@taltech.ee

***Abstract*—Deep Neural Networks (DNNs) are extensively employed in safety-critical applications where ensuring hardware reliability is a primary concern. To enhance the reliability of DNNs against hardware faults, activation restriction techniques significantly mitigate the fault effects at the DNN structure level, irrespective of accelerator architectures. State-of-the-art methods offer either neuron-wise or layer-wise clipping activation functions. They attempt to determine optimal clipping thresholds using heuristic and learning-based approaches. Layer-wise clipped activation functions cannot preserve DNNs' resilience at high bit error rates. On the other hand, neuron-wise clipping activation functions introduce considerable memory overhead due to the addition of parameters, which increases their vulnerability to faults. Moreover, the heuristic-based optimization approach demands numerous fault injections during the search process, resulting in time-consuming threshold identification. On the other hand, learning-based techniques that train thresholds for entire layers concurrently often yield sub-optimal results.**

In this work, first, we demonstrate that it is not essential to incorporate neuron-wise activation functions throughout all layers in DNNs. Then, we propose a hybrid clipped activation function that integrates neuron-wise and layer-wise methods that apply neuron-wise clipping only in the last layer of DNNs. Additionally, to attain optimal thresholds in the clipping activation function, we introduce ProAct, a progressive training methodology. This approach iteratively trains the thresholds on a layer-by-layer basis, aiming to obtain optimal threshold values in each layer separately. Throughout the progressive training, we utilize Knowledge Distillation (KD) to transfer the output class probability from the unbounded activation model (teacher model) to the bounded activation model (student model). ProAct enhances the fault resilience of DNNs, achieving improvements of up to 6.4x at high bit error rates. Moreover, ProAct reduces memory overhead by 91.26x and 134.28x compared to the state-of-the-art neuron-wise activation restriction technique, as evaluated on ResNet50 and VGG16 models, respectively. The source codes of the methods experimented in this paper are released at: https://github.com/hamidmousavi0/reliable-relu-toolbox.git

I. Introduction

Machine Learning (ML) and, in particular, Deep Neural Networks (DNNs) have recently emerged to play a significant role in various applications [1], [2], [3], [4]. DNN hardware accelerators are widely leveraged in safety-critical applications, e.g., autonomous driving and healthcare, where hardware reliability is a major concern [5], [6], [7], [8].

The reliability of DNN accelerators expresses their ability to produce correct outputs in the presence of hardware faults originating from various phenomena, e.g., soft errors that are caused by colliding high energy particles to digital devices and result in bitflips either in memory or logic [7], [9]. In this regard, resilience pertains to the ability of DNNs to maintain their prediction accuracy in the presence of faults [8], [10], [11]. Due to technology miniaturization, soft error rates have increased in recent years, in particular in SRAM-based memories, leading to significant accuracy drops in DNNs due to corrupted parameters that are stored in memory [7], [12], [13].

Since DNN accelerators store parameters (i.e., weights and biases) in memory, which is prone to soft errors, the accuracy of DNNs is jeopardized [14], [15]. Parameters in memories are not being overwritten as frequently as values in the data path, input buffers, and logic elements (e.g., buffers in PEs), thus, bitflips originating from soft errors in parameters are accumulative and persistent, leading to constantly producing errors throughout the inference of a DNN accelerator. Fig. 1 depicts an example of how soft errors in memories storing DNNs' parameters can lead to a catastrophic outcome for an autonomous vehicle by misclassifying a pedestrian as a bird, which can result in a loss of life.

For the sake of simplicity, without loss of generality, in this paper we are addressing Single Event Upset (SEU) fault effects, i.e. soft errors latched to memory elements. SEUs cover a vast majority of Single Event Transient (SET) effects on the logic cells and are significantly more probable than SETs due to the frequent masking of the latter [16]. Moreover, in the reliability assessment, we resort to fault injection into parameters of the DNN, while SEU faults may also occur in the neurons and loop counters. These faults, albeit critical, have a marginal impact on the overall reliability assessment result due to the significantly smaller size of the counter registers versus the parameters memory and thereby marginal fault probabilities [17], [18].

Fault-tolerant techniques to enhance the reliability of DNN accelerators against soft errors in memories are carried out at the architecture and algorithm level. Architecture-level techniques are accelerator-specific and exploit hardware redundancy with performance and cost. Whereas they do not

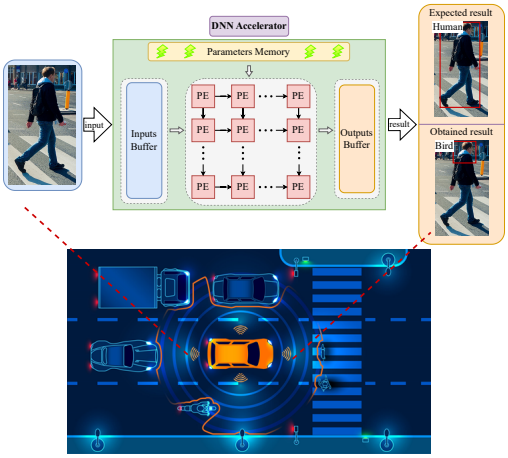


Fig. 1: Example of the impact of memory faults on the output classification in a safety-critical application.

apply to the commercial-off-the-shelf and pre-designed IPs. However, algorithm-level techniques modify the DNN models in the software executable by any accelerators. Throughout the literature, several cost-effective algorithm-level fault tolerance techniques for enhancing the reliability of DNNs are presented, such as activation restriction methods [19], [20], [21].

In the activation restriction methods, the rationale is to limit the activation values within layers to mitigate the effect of fault propagation on the output of DNNs mostly produced by large erroneous values. To this end, a clipping threshold is obtained for the ReLU activation functions in DNNs, to clip the higher values than the threshold to 0.

In Ranger [19] and FT-ClipAct [20], a single threshold is obtained for each layer while fitAct [21] assigns a threshold to the ReLU of each neuron throughout a DNN. To obtain the clipping thresholds, Ranger merely considers them as the maximum activation value in each layer over validation data. FT-ClipAct presents a heuristic search algorithm to identify the corresponding clipping thresholds. On the other hand, FitAct proposes a training-based method to obtain the neuron-wise clipped activation function.

Regarding the related research works, the main shortcomings in the previously proposed activation restriction methods are as follows:

- The granularity of the existing clipping activation functions is not optimal: 1) Layer-wise method does not effectively prevent errors from propagating to outputs; 2) Neuron-wise method imposes significant overhead on DNNs and increases the number of possible fault locations in the memory.
- The existing optimization methods for obtaining clipping thresholds are not only highly time-consuming but also obtain sub-optimal clipping thresholds.

In this work, we attempt to address the identified issues in the literature. To the best of our knowledge, for the first time, a novel activation restriction method is introduced that combines layer-wise and neuron-wise clipping incorporated with progressive training employing Knowledge Distillation (KD) [22], [23] to achieve significant resilience with negligible memory overhead in DNNs. The contributions of this work are as follows:

- Proposing Hybrid Clipped ReLU (HyReLU) activation function restricting the activation values by trainable threshold parameters in a neuron-wise way at the last layer and performs layer-wise in the other layers of DNNs. The proposed HyReLU imposes a negligible memory overhead on DNNs.
- Introducing progressive training to obtain trainable clipping thresholds in HyReLU for each layer separately. It transfers the knowledge from baseline DNN models (teacher) to the clipped DNNs by HyReLU (student), leading to more optimal and effective clipping threshold values ensuring high resilience for DNNs.
- Comparing the results extensively with the state-of-the-art activation-restriction methods. Results demonstrate that the ProAct method improves the resilience of DNNs up to 6.4x at high bit error rates, compared to the state-of-the-art methods with a remarkable reduction in memory overhead, i.e., up to 134.28x.
- Publishing the source codes, not only for the proposed ProAct method but also the implemented state-of-the-art activation restriction methods presented in this paper. To our knowledge, there is no open-source tool in the literature providing activation restriction methods for resilience enhancement of DNNs. The source code and usage instructions for all activation restriction methods, including ProAct, are accessible for researchers on the corresponding GitHub repository[1].

Progressive Training for Hybrid Clipped Activation Function Thresholds (ProAct) empowers DNNs to mitigate error propagation to the output to a significantly greater extent and with considerably lower memory overhead than the state-of-the-art methods.

We analyze the distribution of activation values in layer-wise clipped activation functions within fault-free and faulty models to demonstrate that layer-wise clipping suffices for initial layers, while neuron-wise clipping becomes necessary for the final layers. Subsequently, we propose a hybrid neuron-/layer-wise clipped activation function, wherein only the last layer of DNNs employs neuron-wise clipping activation functions, while the preceding layers utilize layer-wise clipping to mitigate memory overhead.

We demonstrate that the optimization methods utilized in current state-of-the-art approaches fail to attain optimal threshold values. Subsequently, we introduce a progressive training technique based on knowledge distillation for clipped thresholds, executed layer by layer, to improve the resilience of DNNs.

The remainder of the paper is organized as follows. Section II overviews the literature on the fault-tolerant techniques for

[1] https://github.com/hamidmousavi0/reliable-relu-toolbox.git

DNNs. Section III presents the preliminaries regarding the activation restriction and knowledge distillation topics. The motivation for the research is presented in Section IV. The ProAct method is described in Section V. The experimental setup and results are presented in Section VI. Finally, the paper is concluded in Section VII.

II. Related Works

In this section, the research works attempting to improve the reliability of DNNs are overviewed. Techniques for improving the reliability of DNN accelerators against soft errors in memories can be considered at two levels of system abstraction:

- Architecture level: The computing units or memory components of the DNN accelerators are either designed to be reliable (i.e., hardened) or redundant units are included [24], [25], [26], [27].
- Algorithm level: The structure of the DNN model is manipulated or fault-aware training is performed to improve the resiliency of the DNN [28], [29], [30], [21].

Architecture-level techniques are designed specifically for concrete DNN accelerators, and they apply hardware redundancy leading to performance, area, and power overheads. However, algorithm-level techniques are concerned with the DNN model itself and can be applied to any accelerator [31], [32]. Throughout the literature, the effective algorithm-level techniques for improving the reliability of DNNs are presented as follows:

- Fault correction: Faults are corrected using Error Correction Codes (ECC) [33], [34], or Algorithm-Based Fault Tolerance (ABFT) [35] approaches.
- Inherent resilience improvement: Different approaches to improve the inherent resilience of DNNs including quantization and outlier regularization [36], fault-aware training [29], [37], and activation restriction [21], [31], [38], [19], [20].

ECC and ABFT methods are conventional and well-established fault tolerance methods to detect and correct faults using numerical and computational overheads. However, the efficacy of these techniques in fault correction highly depends on the overhead they introduce, whereas the state-of-the-art approaches tend to improve the inherent resilience of DNNs in more effective and efficient ways [29], [21], [36]. Nonetheless, the mentioned approaches for DNNs' inherent resilience improvement are orthogonal to one another and can be applied as a combination to enhance their inherent resilience.

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. Nonetheless, quantization necessitates the utilization of hardware accelerators specifically designed to handle the operations associated with the respective data types. Quantization can be deployed on general-purpose devices as well, however, these carry out the floating-point arithmetic leading to the reliability issues of floating-point data types.

In the research works leveraging fault-aware training methods, all the DNN's parameters are retrained while fault injection is being performed. Fault-aware training for stuck-at and transient faults at activations are presented in [28] and [29], respectively. It is shown, including by radiation experiments, that fault-aware training methods effectively improve the resilience of DNNs [39], [40]. However, these methods retrain the entire DNN which requires updating all the model parameters. This is excessively complex and compute-expensive and requires the possibility of retraining for a pre-trained DNN.

As mentioned, activation restriction methods attempt to restrain the activation values within layers to alleviate the effect of fault propagation on the DNNs outputs, produced by large erroneous activation values. Assuming faults are either in parameters or computations, neurons' erroneous output activations can be detected and handled after the generation of their respective outputs using activation restriction methods [20]. These methods do not require retraining the entire model and do not suffer from the complexity of fault-aware training. In addition, they are non-intrusive in the sense that they do not require any changes to an accelerator.

Piece-wise Rectified Linear Unit (ReLU) is proposed in [38] that finds thresholds to split ReLU into different ranges by training and applies predefined coefficients to its outputs. However, the effect of large faulty activation values remains. Zhan et al. [31] have proposed another method to find the thresholds of ReLU in a layer called BReLU and clip the output to the threshold value itself. BReLU maps the faulty activations in a layer to a non-zero value within that layer, while DNNs are shown to be more resilient if faulty values are replaced with a value near 0.

Ranger [19] presents a clipped ReLU that bounds the layer activations' output to 0 in case their value is higher than a fixed threshold. The threshold values are obtained from the maximum values at each layer seen in a validation set of the dataset. The method of clipping out-bound values to 0 in Ranger is demonstrated to be effective, however, the method does not provide optimal clipping thresholds.

Hoang et al. have analyzed various boundary values on the model's accuracy [20]. Their method, named FT-ClipAct, attempts to find optimal boundary values for each layer that are not necessarily the maximum values of the layers' activations and are smaller than the maximum bounds. The authors propose a heuristic interval search algorithm based on the fault injection process to find appropriate threshold values for the ReLU activation function at each layer. FT-ClipAct incurs significant computational overhead in determining the thresholds for DNNs' layers due to the injection of faults at each step of the search algorithm. Therefore, it is unfeasible to employ it for every single neuron in a DNN.

FitAct [21] proposes an activation function based on the *sigmoid* function that is differentiable to boundary values to optimize them with a gradient-based algorithm. FitAct considers the boundary values for each neuron and smoothly maps the activation outputs to 0. Furthermore, Fitact demonstrates that for fixed-point representation, the optimal threshold values that maintain the baseline accuracy of the fault-free model tend to be smaller. While FitAct effectively enhances the

resilience of DNNs, it concurrently elevates both memory overhead and the likelihood of faults occurring in the activation functions' parameters. This indicates that as the number of parameters in DNNs grows, the likelihood of faults occurring in the threshold values also increases, thereby diminishing the resilience. Furthermore, FitAct trains all the threshold parameters in the clipping activation function simultaneously, which decreases the possibility of providing the optimal threshold for each activation function.

Therefore, there is a need for new activation restriction methods in which more optimal thresholds can be obtained and less memory overhead is incurred to DNNs. This paper attempts to address these shortcomings in the literature.

III. Preliminaries

A. Clipping Activation Functions

Deep Neural Networks (DNNs) that are exploited for image classification mainly consist of two main types of computational layers: convolution and the fully-connected. An activation function follows these layers to take non-linearity into account. ReLU is the most frequently used activation function in DNNs which is defined as follows:

$$ReLU(x) = \max(0, x) \tag{1}$$

ReLU has a remarkable impact on DNN training in terms of efficiency and accuracy. However, it passes all positive values leading to resilience issues once a fault produces large values in the activations. This phenomenon decreases the classification accuracy of the model significantly [20]. Therefore, it is possible to create a clipped ReLU activation function to increase the resilience of DNNs.

The primary strategy for restricting ReLU's output values is to use threshold values for each layer to prevent crossing the large values. To achieve this, a clipped version of the ReLU activation function for each layer is proposed in [19], [20]:

$$ReLU_{clipped}(x) = \begin{cases} x & \text{if } 0 \leq x \leq \lambda \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

where λ is the clipping threshold and any value above this threshold is considered faulty, and its value is clipped to 0.

As mentioned, Ranger [19] obtains the threshold values (λ) based on the maximum value in the activation of each layer on validation data. FT-ClipAct [20] finds the thresholds for each layer by performing an interval search algorithm, which results in a lower accuracy drop than Ranger in the presence of faults. FitAct [21] introduces a neuron-wise smooth activation function and is used as a gradient-based optimization method to obtain all thresholds efficiently and provides better results than FT-ClipAct in terms of accuracy drop, with considerably higher memory overhead.

B. Knowledge Distillation

Knowledge Distillation (KD) is a teacher-student paradigm, where the student model learns to mimic the output class probability of the teacher model [41]. The most common KD technique for mimicking the output of the teacher model is known as *soft targets* that is based on the output logits (activation outputs before *softmax*) of teacher and student models [22]. The smooth, extended version of the *softmax* function transforms the logits of a neural network into soft probabilities P. The formulation is as follows:

$$P(z_i, T) = \frac{\exp(z_i/T)}{\sum_j \exp(z_j/T)} \tag{3}$$

where z is the logits, i is the top output class and j refers to all output classes, and T indicates is a hyper-parameter called temperature.

KD can be expressed as a minimization problem that minimizes an expected ($\mathbb{E}_{X\sim D}$) function based on soft targets, as:

$$\min_{\theta} \mathbb{E}_{X\sim D}[KL(P(f_s(X,\theta),T)||P(f_t(X),T))] \tag{4}$$

where $f_s(\cdot,\theta)$ and $f_t(\cdot)$ are the logit outputs of the student model (s) with parameters θ and pretrained teacher model (t), respectively. KL shows the KL-divergence distance which measures the difference between two distributions, and X is an input to the model that is drawn from the data distribution D.

The main objective of knowledge distillation is to reduce the gap between the outputs of the student and teacher models in the last layers. By optimizing Eq. (4), the logit outputs of the student model eventually coincide with those of the teacher model.

IV. Research Motivation

The existing activation restriction methods in the literature incur a notable accuracy drop at high Bit Error Rates (BERs) and memory overhead accompanied by a time-consuming and computationally expensive process to obtain the clipping thresholds. The primary reason for the mentioned shortcomings stems from the optimization process to determine the clipping thresholds for clipping activation functions within DNN layers or neurons.

It is stated that the optimal clipping threshold value for an activation function (layer-wise or neuron-wise) is the minimum possible value that maintains the accuracy of the fault-free DNN model [20], [21]. The heuristic optimization method in FT-ClipAct [20] demands extensive computation overhead as it involves conducting fault injections at every step of the search process and finding sub-optimal thresholds due to the limited number of search steps.

The gradient-based optimization method in FitAct [21] improves the resilience of DNNs compared to FT-ClipAct as well as reduces its computational overhead. However, the obtained thresholds for neurons are not optimal since all of them are trained simultaneously in each backward pass. As DNNs possess numerous neurons, this optimization process may not necessarily lead to an optimal local minimum for the clipping thresholds of activation functions for all neurons of a DNN.

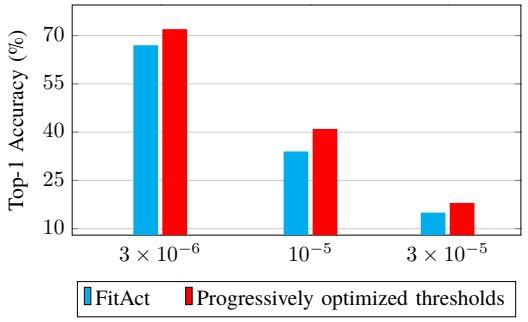


Fig. 2: Top1-Accuracy of AlexNet under different BERs employing FitAct and progressively optimized thresholds.

To illustrate the aforementioned shortcoming, we calculate the clipping threshold values for activation functions in the AlexNet using FitAct. Afterwards, we attempt to progressively reduce the clipping threshold values for the neurons layer by layer while ensuring that the model's baseline accuracy remains unaffected. We accomplish this by training the threshold parameters for each layer individually for 5 epochs, using a higher weight decay hyper-parameter. Fig. 2 illustrates the results for AlexNet resilience based on its accuracy under different BERs into parameters, after minimizing the neurons' thresholds in each layer progressively.

It is observed that the obtained clipping thresholds by FitAct are not the optimal values and it is possible to identify more appropriate clipping threshold values to improve the resilience of DNNs. These results demonstrate the necessity for a new training mechanism to optimize clipping threshold values. In this paper, we introduce the ProAct algorithm, which progressively trains threshold values layer by layer accompanied by Knowledge Distillation (KD).

Moreover, the main factor contributing to memory overhead in FitAct is the implementation of clipped thresholds at the level of individual neurons. Applying an individual clipping threshold to each neuron not only enlarges memory overhead but also increases the probability of memory faults as there are more stored parameters. To comprehend the impact of neuron-wise activation restriction, we examine error propagation through *FitAct*-instrumented AlexNet by illustrating the distribution of activations of each layer without and with faults into parameters (BER = $3e-5$) in Fig. 3. To enhance the visualization, the distribution of values is partitioned into two ranges, the left-hand side column presents the activation values between $[0, 1]$ and the right-hand side one shows them in $(1, \infty)$, respectively.

The distribution of activations in both fault-free and faulty models exhibits a similarity in the initial layers. While, by proceeding through the depth of the model, the disparity between the distribution of activations in fault-free and fault models becomes more pronounced. This phenomenon reveals that the errors are mostly amplified in the last layers of DNNs where it is crucial to harness them. This observation suggests that the output activations in the initial layers of DNNs can be restricted by layer-wise clipping thresholds and the last

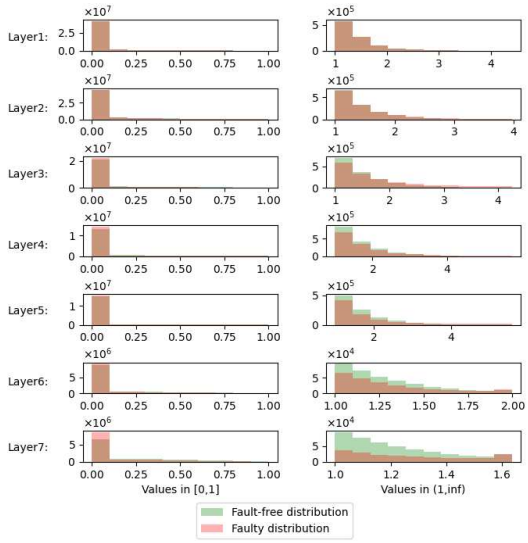


Fig. 3: The distribution of output activation values for the AlexNet model on the CIFAR-10 dataset after applying the FitAct algorithm to find threshold parameters.

layer can be restrained by neuron-wise ones. As a solution, we introduce a hybrid clipped activation function that incorporates neuron-wise thresholds specifically for the last layer of DNNs and layer-wise thresholds for the rest of the layers, aiming to decrease memory overhead and enhance resilience.

V. Methodology

Building upon the insights from the previous section, we introduce ProAct, a progressive training approach for clipping activation function thresholds, implemented in a hybrid manner: neuron-wise exclusively in the last layer of DNNs and layer-wise in all other layers. Progressive training aims to minimize clipping thresholds for activation functions and enhance DNNs resilience while maintaining their baseline accuracy. Moreover, the hybrid clipped activation function mitigates memory overhead and reduces the occurrence of faults in memory locations.

A. *Hybrid Clipped ReLU and Its Memory Overhead*

To propose a hybrid clipped ReLU activation function, it is essential to incorporate neuron-specific thresholds for the neurons in the final layer of DNNs, while employing layer-specific thresholds for the neurons in the preceding layers. In addition, to utilize the gradient-based optimization method, it is necessary to create a differentiable version of the activation function. To achieve these objectives, we introduce a hybrid clipped ReLU activation function (HyReLU), which draws inspiration from the sigmoid activation function (σ), in Eq. (5). This function guarantees smooth transitions around the threshold values (λ) utilized in the clipped ReLU.

$$HyReLU(x,\lambda,l) = \begin{cases} \max\{0, x_i[1-\sigma(k[\lambda_i - x_i])]\} \; if \; l = L \\ \max\{0, x[1-\sigma(k[\lambda - x])]\} \;\; otherwise \end{cases} \tag{5}$$

In Eq. (5), x is an input activation, λ is a trainable parameter representing the value of the clipping threshold in the respective neuron/layer and k is a hyper-parameter determining the slope for the smooth transition to 0, which is obtained through cross-validation. L indicates the last layer index. This equation expresses that the values larger than λ in a layer are considered erroneous and are smoothly clipped to 0.

HyReLU is employed across all neurons of the last layer of DNNs (i.e., the layer preceding the output layer), with each having a distinct trained λ_i. For other layers, the function is applied separately, with each layer possessing its own trained λ. Consequently, the memory overhead introduced to a DNN with the HyReLU is formulated in Eq. (6).

$$Memory\ Overhead\ = \frac{\#Layers\ +\ \#Neurons_{last\ layer}}{\#Parameters_{DNN}} \tag{6}$$

B. *ProAct: Progressive Training for HyReLU Activation Function*

To obtain the best clipping thresholds (λ) in HyReLU for each layer/neuron in a DNN, we propose ProAct, a layer-wise progressive training method exploiting knowledge distillation. Figure 4 depicts an outline of the ProAct approach, where the purpose is to find an optimal λ for each HyReLU without breaching the maximum permitted accuracy drop and memory overhead.

ProAct trains the clipping threshold of each layer separately, from the last to the first layer. ProAct includes two main steps to find the threshold parameters 1) Preprocessing and 2) Progressive training. In the first step, we start by profiling the model on validation data to determine the initial values for the threshold parameters. Specifically, we initialize the threshold parameters with the maximum activation value observed by the corresponding layer/neuron on the validation dataset. Then, we replace all ReLU activation functions with the proposed HyReLU, using the initial threshold parameters (preprocessing step in Algorithm 1 (1-8 lines)). Within the progressive training step, we progressively select the layers from the last layer to the first one and train the threshold parameters (λs) in the HyReLU of the selected layer through the KD-based training. The clipping threshold of the target layer is trained using KD and this process continues down to the first layer (Training step in Algorithm 1 (9-15 lines)).

The proposed training method utilizes KD in a way that the clipping thresholds in HyReLU are trained based on the supervision of the unbounded fault-free (baseline) model. Through this process, the purpose is to mimic the output values of the unbounded fault-free model in the modified model with the HyReLU activation function. The pre-trained baseline model including ReLU is used as the teacher model that includes the golden output values and the modified DNN is the student model that has the same structure as the teacher model, but ReLU replaced by HyReLU.

The loss function $\mathcal{L}_{KD}$ is computed based on the Kullback–Leibler divergence (KL) distance [42] between the distribution of output values in the student and the teacher model as:

$$\mathcal{L}_{KD}(X,s,t) = \mathbb{E}_{x\sim D} KL\Big(P(f_s(x,T,\lambda))||P(f_t(x),T)\Big) \tag{7}$$

where $P(f(.,T))$ show the soft output logits with temperature parameter T.

Therefore, the whole loss function (L) that we use to train the threshold parameters in the selected layer is:

$$\mathcal{L}(X,Y,\lambda) = \mathcal{L}_{KD}(X,s,t) + \mathbb{E}_{x,y\sim D} L_{ce}(f_s(x,\lambda),y) + \gamma R(\lambda) \tag{8}$$

where the L_{ce} and $R(\lambda)$ show the cross entropy loss function and l_2-regularization. l_2-regularization helps to constrain the magnitude of the threshold parameters, preventing them from becoming excessively large.

Algorithm 1 ProAct: Progressive Training for HyReLU Activation Function

Input: The unbounded teacher and bounded student models (t, s), learning rate (α), Regularization parameter (γ), number of epochs (N), BERs = $[10^{-6}, 3\times10^{-6}, 10^{-5}, 3\times10^{-5}, 10^{-4}]$;
Output: Resilience DNN;
Preprocessing Step
1: **for** $l \in [1, 2, \cdots, L]$ **do**:
2: **if** $l = L$ **then**:
3: Profile the model to find max value in each ***neuron***;
4: **else**:
5: Profile the model to find max value in each ***Layer***;
6: **end if**;
7: Initial the threshold parameters (λ) based on max values;
8: **end for**
Progressive Training Step
9: **for** $l \in [L, L-1, \cdots, 1]$ **do**:
10: **for** $i \leftarrow 1$ to N **do**:
11: Compute $\mathcal{L}$ (loss function) based on Eq. (8);
12: Compute $\partial\mathcal{L}/\partial\lambda$ where $\lambda : \nabla_\lambda \mathcal{L}(X,Y,\lambda)$;
13: Update λ via Adam optimizer;
14: **end for**;
15: **end for**;

VI. Experiments

A. Experimental Setup

The proposed method ProAct is applied to and evaluated on three DNNs: AlexNet [43], VGG-16 [44], and ResNet-50 [45], all trained on both CIFAR-10 and CIFAR-100 datasets. Their baseline classification accuracy on the test sets is shown in

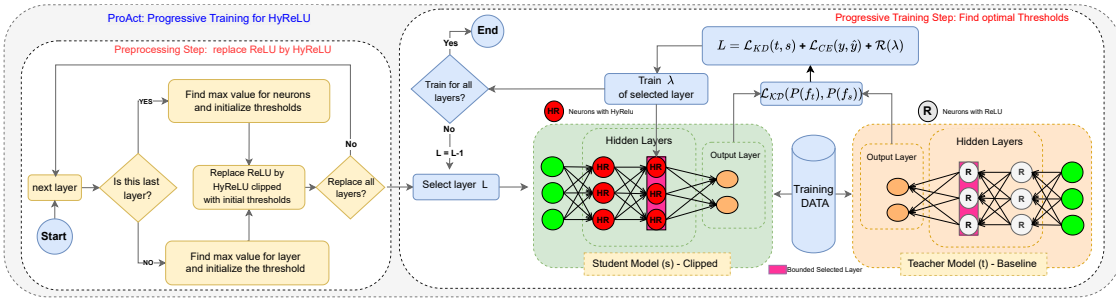


Fig. 4: Hybrid Progressive training based on Knowledge Distillation

TABLE I: Baseline accuracy for each baseline CNNs

Networks	AlexNet	VGG-16	ReNet-50
Accuracy for CIFAR-10	81.67%	89.87%	91.11%
Accuracy for CIFAR-100	55.44%	65.45%	74.37%

Table I. It is noteworthy that experiments in this work exploit 32-bit fixed point data type representation in which the Most Significant Bit (MSB) is for sign, 15 bits are for the integer and 16 bits are for the fraction. All experiments in this paper are performed on Nvidia® A4000-16GB GPU.

To demonstrate the excellence and effectiveness of ProAct with respect to the state-of-the-art, results are compared to Ranger [19], FT-ClipAct [20], and FitAct [21] methods. We implement Ranger in, both, layer-wise and neuron-wise manners and use a random small part of training data (3000 out of 60000 in both CIFAR-10 and CIFAR-100 datasets) as the validation data to find the maximum values for the layers/neurons. FT-ClipAct is implemented layer-wise and FitAct is implemented neuron-wise, as they are presented in [20] and [21], respectively.

To obtain a quantitative comparison with the existing works, we carry out the reliability assessment by injecting random bit-flip faults into the parameters of CNNs, including weights, bias, and parameters of clipping activation functions as the fault space. Bits are randomly selected and flipped based on the 32-bit fixed-point data representation. We consider different Bit Error Rates (BERs) to flip multiple bits to model the accumulative effect of faults into the memory through time. The experimented BERs are $[10^{-7}, 3\times10^{-7}, 10^{-6}, 3\times10^{-6}, 10^{-5}, 3\times10^{-5}]$.

Fault injection experiments are repeated 500 times for each BER and average results for Top-1 accuracy are reported and compared. For the fault injection, we adopt and extend PyTorchFI [46] to consider clipping thresholds in the fault space which is developed on top of PyTorch. The training iterations consist of 50 epochs for neuron-specific clipping thresholds in the final layer and 20 epochs for layer-wise HyReLUs. This ensures comparable computational overhead to FitAct, which employs 150 epochs. We initialize the learning rate at 0.01, halving it every 10 epochs, and utilize a batch size of 128.

B. Experimental Results

1) ***Effect of Activation Restriction Methods on DNNs' Baseline Accuracy- and Memory Footprint****:* As mentioned, the clipping thresholds in any activation restriction method are obtained through validation data (not test data). On the other hand, the main requirement of applying them is that they are required not to drop the baseline accuracy of fault-free DNNs over unseen test data. Table II shows the impact of activation restriction methods on the baseline accuracy for each DNN after application. It is observed that:

- Ranger has the least effect on the accuracy drop compared to the other methods. Since Ranger considers the clipping threshold as the maximum value seen in validation data (either for neurons or layers), the obtained clipping thresholds are large enough not to affect the inference of the test data. As a result, in fault-free DNNs in Table II Ranger reduces the accuracy by less than 0.2%. However, this method does not improve DNNs' resilience as effectively as other methods, as shown in the next subsection.
- FT-ClipAct introduces the largest accuracy drop through all methods, from 0.9% in AlexNet trained up to 4.68% in ResNet-50 both trained on CIFAR-10. Such accuracy drop is significant for DNNs, especially in safety-critical applications, and decreases the effectiveness of the applied activation restriction method. Since the heuristic search algorithm is very complex and slow, it is exploited with a small subset of the training data (1000 out of 60000 in both CIFAR-10 and CIFAR-100). Therefore, the obtained thresholds are not optimal and influence the accuracy considerably.
- The accuracy drop induced by applying ProAct is always less than 1% which is negligible. Moreover, in all cases, ProAct reduces the baseline accuracy of DNNs less than both FT-clipAct and FitAct methods. This is due to the progressive training method, which ensures that the optimal threshold for each layer is found separately without sacrificing accuracy.

The existing methods are either neuron-wise or layer-wise, which lay different memory overhead on DNNs. Layer-wise approaches introduce new clipping threshold parameters to DNNs proportional to the number of layers which is a negligible overhead. Whereas neuron-wise approaches lay a remarkable

TABLE II: Baseline Accuracy drop of DNNs by different activation function restriction methods.

Activation restriction method	Ranger (layer-wise)	Ranger (neuron-wise)	FT-ClipAct (layer-wise)	FitAct (neuron-wise)	ProAct (hybrid)
AlexNet CIFAR-10	0.00%	0.00%	0.90%	0.78%	0.29%
AlexNet CIFAR-100	0.01%	0.03%	2.40%	0.64%	0.51%
VGG-16 CIFAR-10	0.00%	0.02%	1.15%	1.14%	1.00%
VGG-16 CIFAR-100	0.00%	0.07%	1.69%	0.83%	0.35%
ResNet-50 CIFAR-10	0.15%	0.19%	4.69%	0.30%	0.22%
ResNet-50 CIFAR-100	0.00%	0.08%	1.60%	0.03%	0.10%

TABLE III: Comparison of memory overhead for neuron-wise, layer-wise, and hybrid activation restriction methods

Activation restriction method	neuron-wise	layer-wise	**Hybrid (ProAct)**
AlexNet CIFAR-10	0.29	3×10^{-5}	**0.017**
AlexNet CIFAR-100	0.21	3.5×10^{-5}	**0.020**
VGG-16 CIFAR-10	1.88	8.84×10^{-5}	**0.014**
VGG-16 CIFAR-100	0.85	4.46×10^{-5}	**0.012**
ResNet-50 CIFAR-10	12.23	2×10^{-4}	**0.134**
ResNet-50 CIFAR-100	12.23	2×10^{-4}	**0.134**

overhead as the number of neurons in the DNN is huge, leading to an increase in fault locations within the memory. However, the ProAct memory footprint is limited since it is a hybrid neuron-wise and layer-wise activation function.

Table III compares the memory overhead of neuron-wise, layer-wise, and the proposed hybrid approach for activation restriction methods. It is observed that ProAct significantly reduces memory overhead compared to neuron-wise techniques such as FitAct, ranging from 10.5x to 134.28x, while still ensuring enhanced accuracy in protecting DNNs against faults.

*2) **Resilience Analysis of Activation Restriction Methods Using Fault Injection**:* Fig. 5 and Fig. 7 depict the Top-1 accuracy of DNNs leveraging different activation restriction methods on CIFAR-10 and CIFAR-100 respectively, under fault injection campaigns as described in Subsection VI-A. The right column in both figures magnifies the results to highlight the impact of ProAct against the state-of-the-art methods, in particular FT-ClipAct and FitAct. It is observed that equipping DNNs with ProAct remarkably enhances the resilience of DNNs compared to the other state-of-the-art methods.

Regarding Fig. 5 and Fig. 7, at all BERs, the accuracy

TABLE IV: Comparing the accuracy drop of DNNs using different activation restriction methods under fault injection.

DNNs	BER	FT-ClipAct	FitAct	**ProAct**
AlexNet CIFAR-10	1E-6	7.67%	4.34%	**2.52%**
VGG-16 CIFAR-10	3E-6	3.19%	2.61%	**1.88%**
ResNet-50 CIFAR-10	1E-6	9.09%	1.53%	**1.42%**
AlexNet CIFAR-100	3E-7	7.89%	6.31%	**5.28%**
VGG-16 CIFAR-100	1E-7	6.74%	6.34%	**4.40%**
ResNet-50 CIFAR-100	3E-7	12.75%	11.24%	**9.37%**

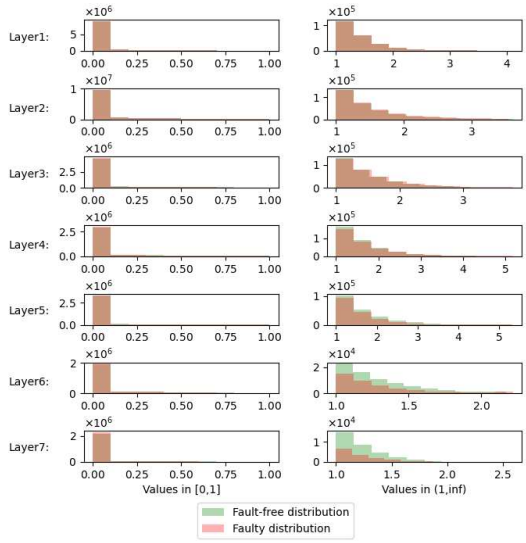


Fig. 6: The distribution of output activation values for the AlexNet model on the CIFAR-10 dataset after applying the ProAct algorithm to find threshold parameters.

of DNNs with ProAct is higher than the DNNs with other activation restriction methods. As it is observed, Ranger provides the least resilient DNNs. According to the results, although FitAct provides better resilience than FT-ClipAct, it introduces a remarkable memory overhead orders of magnitude more than FT-ClipAct. Whereas ProAct achieves a higher resilience than all existing methods with negligible overhead.

Moreover, it is observed that in model-wise FI experiments, all activation restriction methods can effectively improve the resilience of DNNs compared to unprotected DNNs. However, they fail to provide highly resilient DNNs at high BERs. When faulty weights are spread throughout a DNN, several neurons in various layers are affected. Consequently, in a high BER, the values of affected neurons are restricted by their activation functions and make numerous erroneous activations propagate to the DNN output resulting in a considerable accuracy drop. However, ProAct surpasses the other activation restriction techniques in terms of providing superior accuracy for DNNs in model-wise FI.

Table IV summarizes the results for accuracy drop of experimented DNNs with respect to their own baseline accuracy in Table I, hardened by FT-ClipAct, FitAct and ProAct, at the BERs where the accuracy drop of ProActed DNNs for CIFAR-10 is less than 5%, and for CIFAR-100 is less than 10%. This comparison implicitly includes the accuracy drop due to activation restriction methods exhibiting the overall benefit of ProAct. According to the results, it is observed that ProAct reduces the accuracy drop of DNNs from 1.36x up to 6.4x compared to FT-ClipAct and from 1.07x up to 1.72x compared to FitAct.

*3) **Activation Distribution in ProActed DNNs**:* As discussed in Section IV and illustrated in Fig. 3, there is a significant

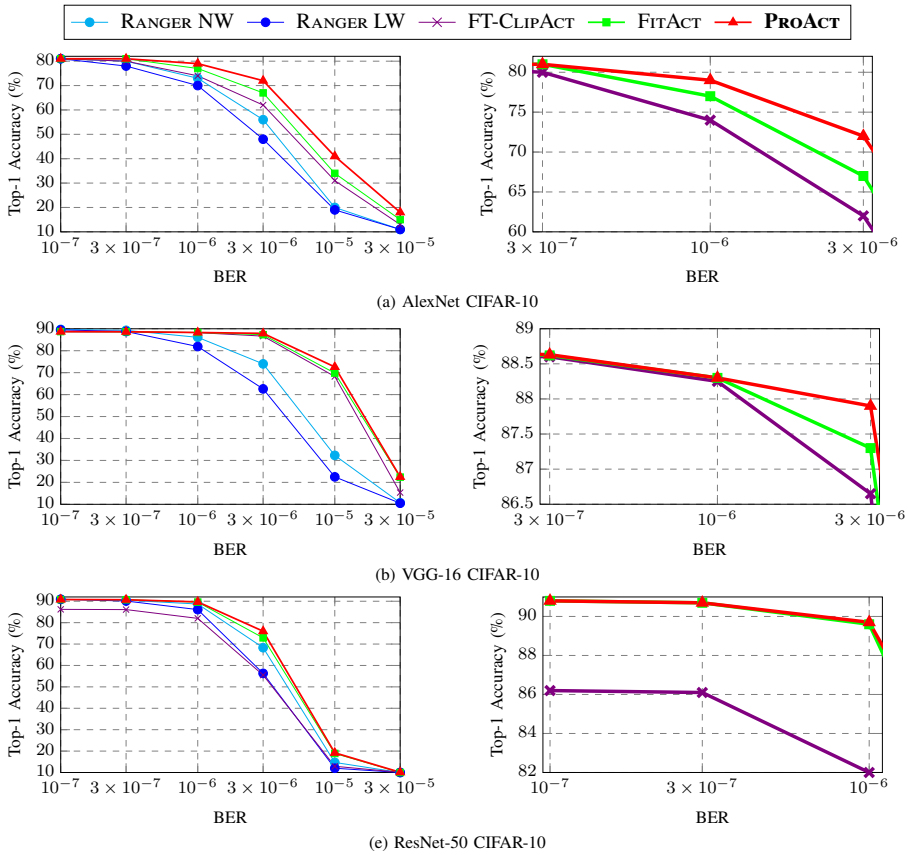


Fig. 5: Top-1 accuracy comparison of DNNs using ProAct with Ranger neuron-wise , Ranger layer-wise , FT-ClipAct, and FitAct methods under fault injection.

TABLE V: Average L_2 distance of different DNNs and mitigation techniques, layer-wise FI, BER = $1E-4$.

	L_2 **Distance**		
Method	**AlexNet**	**VGG-16**	**ResNet-50**
FitAct	65.6	80.3	93.7
ProAct	**45.9**	**72.4**	**86.5**

difference in the distribution of activation values between the fault-free and faulty models, particularly in the last layer. ProAct addresses this issue by reducing the gap between these distributions through the identification of more effective thresholds for the hybrid activation functions, using a progressive training mechanism.

Fig. 6 presents the distribution of fault-free and faulty values after applying ProAct to the AlexNet model on the CIFAR-10 dataset. The figure demonstrates that the distributions are more similar, indicating that ProAct generates a model that closely resembles the fault-free model. Specifically, incorporating Knowledge Distillation (KD) in the training process helps identify threshold values that maintain the similarity between the distributions of the fault-free model and the faulty model.

To numerically evaluate the similarity of the activation values between the clipped model and the fault-free model, we compute the average L_2 distance metric for all the layers of the models as:

$$L_2 = \frac{1}{N \times L} \sum_{i=1}^{L} \sum_{j=1}^{N} |z_{ij} - f_{ij}|_2^2 \tag{9}$$

where L and N show the number of DNN's layers and neurons in a layer, respectively. Also, z and f indicate the activation value for faulty and fault-free models, respectively.

Table V compares the similarity metric for ProAct and FitAct as the state-of-the-art optimization based method. As observed, the L_2 distance of activations for all models is decreased by the ProAct, meaning it produces activations much closer to the fault-free network than FitAct. Compared to FitAct, ProAct improves the L2 distance by 30%, 9.83%, and 7.68% on AlexNet, VGG-16 and ResNet-50, respectively.

VII. Conclusion

In this work, we introduce ProAct, a progressive training method for determining threshold values in a novel hybrid

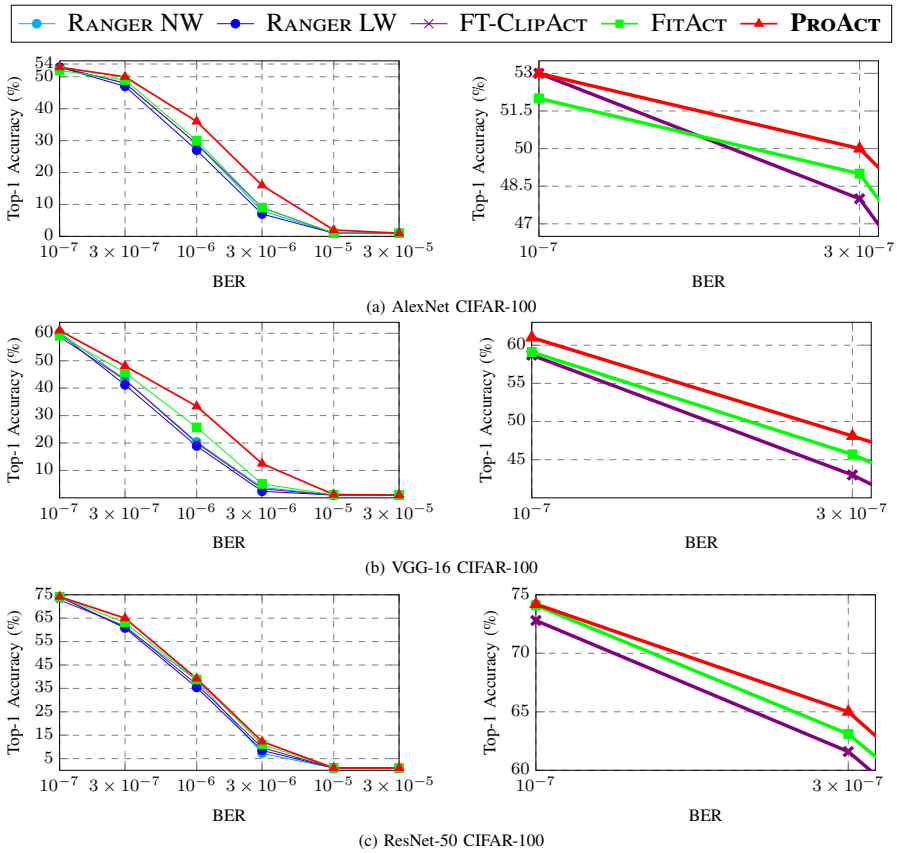


Fig. 7: Top-1 accuracy comparison of DNNs using ProAct with Ranger neuron-wise, Ranger layer-wise, FT-ClipAct, and FitAct methods under fault injection.

clipped ReLU (HyReLU) activation function aimed at enhancing the resilience of DNNs. We demonstrate that existing optimization techniques are computationally intensive and frequently fail to find optimal threshold values. Additionally, neuron-wise clipping methods such as FitAct are shown to incur substantial memory costs. To address these issues, we develop a hybrid clipped ReLU activation function that reduces memory overhead by applying neuron-wise clipping solely in the final layer and layer-wise clipping in the preceding layers. Following this, we proposed a progressive training strategy utilizing knowledge distillation to train the threshold parameters layer by layer, effectively identifying suitable threshold values for the HyReLU. Our experimental results indicate that ProAct significantly improves the resilience of DNNs, with enhancements of up to $6.4x$ in high bit error rates. Furthermore, our approach dramatically reduces memory overhead, achieving reductions of $10.5\times$ to $134.28\times$ compared to the leading neuron-wise activation restriction methods. Furthermore, we have published all source codes in Python, enabling researchers to present more effective approaches in this area.

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] A. Voulodimos, N. Doulamis, A. Doulamis, and E. Protopapadakis, "Deep learning for computer vision: A brief review," *Computational intelligence and neuroscience*, vol. 2018, 2018.

[3] J. Lin, W.-M. Chen, Y. Lin, J. Cohn, C. Gan, and S. Han, "Mcunet: Tiny deep learning on iot devices," *arXiv preprint arXiv:2007.10319*, 2020.

[4] M. Loni, H. Mousavi, M. Riazati, M. Daneshtalab, and M. Sjödin, "Tas: ternarized neural architecture search for resource-constrained edge devices," in *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2022, pp. 1115–1118.

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

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

[8] 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.

[9] C. Bolchini, L. Cassano, A. Miele, and A. Toschi, "Fast and accurate error simulation for cnns against soft errors," *IEEE Transactions on Computers*, 2022.

[10] 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.

[11] 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.

[12] 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)*. IEEE, 2018, pp. 1–10.

[13] E. Malekzadeh, N. Rohbani, Z. Lu, and M. Ebrahimi, "The impact of faults on dnns: A case study," in *2021 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*. IEEE, 2021, pp. 1–6.

[14] M. A. Neggaz, I. Alouani, S. Niar, and F. Kurdahi, "Are cnns reliable enough for critical applications? an exploratory study," *IEEE Design & Test*, vol. 37, no. 2, pp. 76–83, 2019.

[15] T. Spyrou, S. A. El-Sayed, E. Afacan, L. A. Camuñas-Mesa, B. Linares-Barranco, and H.-G. Stratigopoulos, "Reliability analysis of a spiking neural network hardware accelerator," in *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2022, pp. 370–375.

[16] R. Baumann, "Soft errors in advanced computer systems," *IEEE design & test of computers*, vol. 22, no. 3, pp. 258–266, 2005.

[17] A. Lotfi, S. Hukerikar, K. Balasubramanian, P. Racunas, N. Saxena, R. Bramley, and Y. Huang, "Resiliency of automotive object detection networks on gpu architectures," in *2019 IEEE International Test Conference (ITC)*. IEEE, 2019, pp. 1–9.

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

[19] Z. Chen, G. Li, and K. Pattabiraman, "A low-cost fault corrector for deep neural networks through range restriction," in *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2021, pp. 1–13.

[20] 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.

[21] 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)*. IEEE, 2022, pp. 1239–1244.

[22] G. Hinton, O. Vinyals, J. Dean *et al.*, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, vol. 2, no. 7, 2015.

[23] M. Goldblum, L. Fowl, S. Feizi, and T. Goldstein, "Adversarially robust distillation," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 04, 2020, pp. 3996–4003.

[24] C. Liu, C. Chu, D. Xu, Y. Wang, Q. Wang, H. Li, X. Li, and K.-T. Cheng, "Hyca: A hybrid computing architecture for fault-tolerant deep learning," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 10, pp. 3400–3413, 2021.

[25] 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)*. IEEE, 2020, pp. 1–5.

[26] Y. Hong, J. Lian, L. Xu, J. Min, Y. Wang, L. J. Freeman, and X. Deng, "Statistical perspectives on reliability of artificial intelligence systems," *Quality Engineering*, pp. 1–23, 2022.

[27] 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.

[28] 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.

[29] 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)*. IEEE, 2020, pp. 1–10.

[30] F. F. d. 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, 2019.

[31] J. Zhan, R. Sun, W. Jiang, Y. Jiang, X. Yin, and C. Zhuo, "Improving fault tolerance for reliable dnn using boundary-aware activation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 10, pp. 3414–3425, 2021.

[32] C. Schorn, A. Guntoro, and G. Ascheid, "An efficient bit-flip resilience optimization method for deep neural networks," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2019, pp. 1507–1512.

[33] M. Jang and J. Hong, "Mate: Memory-and retraining-free error correction for convolutional neural network weights," *Journal of information and communication convergence engineering*, vol. 19, no. 1, pp. 22–28, 2021.

[34] 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)*. IEEE, 2022, pp. 724–729.

[35] 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.

[36] E. Ozen and A. Orailoglu, "Snr: Squeezing numerical range defuses bit error vulnerability surface in deep neural networks," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 20, no. 5s, pp. 1–25, 2021.

[37] N. Cavagnero, F. Dos Santos, M. Ciccone, G. Averta, T. Tommasi, and P. Rech, "Transient-fault-aware design and training to enhance dnns reliability with zero-overhead," in *2022 IEEE 28th International Symposium on On-Line Testing and Robust System Design (IOLTS)*. IEEE, 2022, pp. 1–7.

[38] M. S. Ali, T. B. Iqbal, K.-H. Lee, A. Muqeet, S. Lee, L. Kim, and S.-H. Bae, "Erdnn: Error-resilient deep neural networks with a new error correction layer and piece-wise rectified linear unit," *IEEE Access*, vol. 8, pp. 158 702–158 711, 2020.

[39] G. Gambardella, N. J. Fraser, U. Zahid, G. Furano, and M. Blott, "Accelerated radiation test on quantized neural networks trained with fault aware training," in *2022 IEEE Aerospace Conference (AERO)*. IEEE, 2022, pp. 1–7.

[40] 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 20nm kintex ultrascale™ fpga," *IEEE Transactions on Nuclear Science*, 2022.

[41] L. Wang and K.-J. Yoon, "Knowledge distillation and student-teacher learning for visual intelligence: A review and new outlooks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

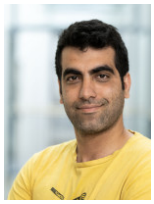
[42] J. Shlens, "Notes on kullback-leibler divergence and likelihood," *arXiv preprint arXiv:1404.2000*, 2014.

[43] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.

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

[45] 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*, 2016, pp. 770–778.

[46] 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)*. IEEE, 2020, pp. 25–31.



Seyedhamidreza Mousavi is currently pursuing the PhD degree in computer science and engineering at the School of Innovation, Design, and Engineering, Mälardalen University, Västerås, Sweden. He is a member of AutoDeep and FASTER-AI projects. His research is focused on designing deep neural network architectures that are both high-performing and compact, while also ensuring their safety using reliable and robust neural architecture search techniques.

Mohammad Hasan Ahmadilivani is a PhD student in the Computer Systems Department at Tallinn University of Technology (Taltech), Estonia. He earned his MSc in Computer Architecture Systems from the University of Tehran, Iran, in 2020. His research focuses on developing analytical methods to measure and enhance the hardware reliability of deep neural networks (DNNs). His research interests include exploiting DNNs in safety-critical applications, robust computer vision, and efficient and reliable DNN accelerator design.

Jaan Raik is a Full Professor at the Department of Computer Systems and Head of the Center for Dependable Computing Systems at the Tallinn University of Technology (Taltech), Estonia. He received his M.Sc. and Ph.D. degrees from Taltech in 1997 and in 2001, respectively. His research interests cover a wide area in electrical engineering and computer science domains including reliability of deep learning, hardware test, functional verification, fault-tolerance and security as well as emerging computer architectures. He has co-authored more than 400 scientific publications. He is a member of IEEE Computer Society, HiPEAC and of Steering/Program Committees of numerous conferences within his field. He served as the General Chair to IEEE European Test Symposium '25, '20, IFIP/IEEE VLSI-SoC '16, DDECS '12), Vice General Chair IEEE European Test Symposium '24, DDECS '13 and Program Co-Chair DDECS '23, '15, CDN-Live '16 conferences. He was awarded the Global Digital Governance Fellowship at Stanford (2022), HiPEAC Paper Award (2020), the Order of the White Star 4th class medal by the President of Estonia (2016) and Estonian Academy of Science's Bernhard Schmidt Award for innovation (2007).

Maksim Jenihhin is a tenured associate professor of computing systems reliability and Head of the research group Trustworthy and Efficient Computing Hardware (TECH) at the Tallinn University of Technology, Estonia. He received his PhD degree in Computer Engineering from the same university in 2008. His research interests include methodologies and EDA tools for hardware design, verification and debug, and security, as well as nanoelectronics reliability and manufacturing test topics. He has published more than 150 research papers, supervised several PhD students and postdocs and served on executive and program committees for numerous IEEE conferences (DATE, ETS, DDECS, LATS, NORCAS, etc.). Prof. Jenihhin coordinates European collaborative research projects HORIZON MSCA DN "TIRAMISU" (2024), HORIZON TWINN "TAICHIP" (2024) and national ones about energy efficiency and reliability of edge-AI chips and cross-layer self-health awareness of autonomous systems.

Masoud Daneshtalab Masoud Daneshtalab is currently a full Prof. at Mälardalen University in Sweden, Adj. Prof. at TalTech in Estonia and leads the Heterogeneous System research group. He is on the Euromicro board of directors, an editor of the MICPRO journal, and has published over 200 refereed papers. His research interests include HW/SW/Algorithm co-design, dependability and deep learning acceleration.

Appendix 6

VI
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. In *The 30th IEEE International Symposium on On-Line Testing and Robust System Design (IOLTS)*, pages 1–6. Rennes, France, 2023

Cost-Effective Fault Tolerance for CNNs Using Parameter Vulnerability Based Hardening and Pruning

Mohammad Hasan Ahmadilivani[1], Seyedhamidreza Mousavi[2],
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, jaan.raik, maksim.jenihhin}@taltech.ee
[2]{seyedhamidreza.mousavi, masoud.daneshtalab}@mdu.se

2024 IEEE 30th International Symposium on On-Line Testing and Robust System Design (IOLTS) | 979-8-3503-7055-3/24/$31.00 ©2024 IEEE | DOI: 10.1109/IOLTS60994.2024.10616072

***Abstract*—Convolutional Neural Networks (CNNs) have become integral in safety-critical applications, thus raising concerns about their fault tolerance. Conventional hardware-dependent fault tolerance methods, such as Triple Modular Redundancy (TMR), are computationally expensive, imposing a remarkable overhead on CNNs. Whereas fault tolerance techniques can be applied either at the hardware level or at the model levels, the latter provides more flexibility without sacrificing generality. This paper introduces a model-level hardening approach for CNNs by integrating error correction directly into the neural networks. The approach is hardware-agnostic and does not require any changes to the underlying accelerator device. Analyzing the vulnerability of parameters enables the duplication of selective filters/neurons so that their output channels are effectively corrected with an efficient and robust correction layer. The proposed method demonstrates fault resilience nearly equivalent to TMR-based correction but with significantly reduced overhead. Nevertheless, there exists an inherent overhead to the baseline CNNs. To tackle this issue, a cost-effective parameter vulnerability based pruning technique is proposed that outperforms the conventional pruning method, yielding smaller networks with a negligible accuracy loss. Remarkably, the hardened pruned CNNs perform up to 24% faster than the hardened un-pruned ones.**

I. Introduction

Convolutional Neural Networks (CNNs) have found widespread application in various safety-critical domains, owing to their superior accuracy compared to human performance [1], [2]. Hardware devices, including general-purpose processors (e.g., CPUs and GPUs) and specialized accelerators (e.g., FPGAs and ASICs), are employed to efficiently execute CNN models [3], [4]. In many cases, especially with the general purpose accelerators but also with off-the-shelf integrated circuits and hard/firm cores, it is not possible to alter the underlying hardware to improve the fault tolerance of the CNN operation.

The hardware systems deploying CNNs rely on extensive memory resources to store the parameters, making them susceptible to various fault effects due to transistor miniaturization [5]. Consequently, a major concern in deploying CNNs on hardware devices is their resilience to faults in memory, particularly those affecting their parameters. Extensive studies have demonstrated that faults in CNN parameters lead to drastic accuracy drops at very low error rates [6]–[9].

979-8-3503-7055-3/24/$31.00 ©2024 IEEE

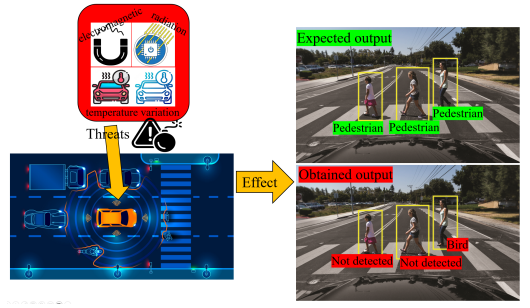


Fig. 1: Potential impact of faults on the output classification in the object detection task of an autonomous vehicle.

Fig. 1 illustrates an example of the effect of faults on the output classification in the object detection task of an autonomous vehicle. The faults can be a result of different causes, including temperature variation, terrestrial or cosmic radiation, circuit aging, or electromagnetic interference. A CNN running on a computing device in the vehicle classifies the input images and due to faults, some parameters are erroneous. As a result, the failure to recognize the pedestrians leads to a catastrophe. Therefore, it is crucial to enhance the fault tolerance of CNN models running on hardware devices to effectively employ them in safety-critical applications [10]–[12].

To mitigate the impact of faults on the deployment of CNNs and at the same time avoid the high overhead in conventional fault-tolerant techniques such as Triple Modular Redundancy (TMR), researchers proposed selective hardening approaches [7], [13]–[16]. Here, the objective is to protect the parameters or neurons that have a larger effect on the neural network's outputs against faults and errors. Therefore, the more vulnerable neurons are identified by resilience analysis and they are executed on hardened processing elements on the target hardware.

Although these methods propose a model-level resilience analysis to identify the more vulnerable parameters/neurons, their protection techniques are restricted to FPGAs and ASICs that can be freely modified and redesigned. Whereas there exist numerous applications from high-performance to edge computing, where general-purpose computing devices such as CPUs and GPUs or hard and firm accelerator cores are deployed that do not support redesigning the hardware for

fault-tolerance [17], [18].

Moreover, *fault-aware pruning* with retraining is another approach for improving fault tolerance of CNNs proposed by [19], [20] in which the parameters that are mapped to corrupted processing elements of the target accelerator are pruned in the network. These methods are not only accelerator-specific but also should be applied to each individual chip with a different fault map separately. Moreover, they cannot be applied in the field but are designed to tolerate faults that have been already diagnosed in the laboratory. Thus, model-level fault tolerance approaches are preferred in terms of their flexibility.

Quantization is shown to be highly effective for the resilience of CNNs [21] since it restricts the numerical range within a CNN, thus eliminating the effect of large values produced due to faults and bitflips in a CNN. Nevertheless, apart from accuracy concerns, deploying quantized CNNs requires dedicated hardware accelerators for handling associated operations. Otherwise, they carry out the floating-point arithmetic of general-purpose computing devices [22] which leads to the reliability issues of floating-point data types, that is contradictory to the purpose of hardening by quantization. Nonetheless, the model-level fault tolerance methods are mostly orthogonal to quantization and they can be employed on top of each other to improve the resilience of DNNs.

Fault-aware training [23], [24] effectively improves the resilience of DNNs. However, it retrains the entire CNN with numerous fault injection scenarios that is not only excessively complex but also requires the possibility of having access to parameters. *Error Correction Codes (ECC)* and *Algorithm-based Fault Tolerance (ABFT)* utilize data encoding/decoding processes for real-time fault detection and correction [18], [25]. However, the practicality of these techniques in fault correction is questionable due to the overhead they introduce to memory and computations, posing a considerable challenge for CNNs that already have substantial memory and computational requirements.

Activation restriction methods [26]–[28] bound the activation values between layers through activation functions (i.e., ReLU) to mitigate error propagation to the outputs of CNNs. They clip the activations to 0 when their values exceed pre-identified ranges. These methods are effective in enhancing the resilience of CNNs, however, they do not provide error correction, and CNNs fail to work at high error rates due to the replacement of numerous feature maps with 0. [29] proposes a correction layer that executes each convolutional layer three times for fault detection and correction which however lays a prohibitive performance overhead to CNNs.

To overcome the previously mentioned issues, this paper introduces a novel model-level hardening solution to modify the architecture of CNNs to allow fault correction at inference inherently. An efficient error correction mechanism is designed enabled by selectively duplicated channels (in both convolutional and fully connected layers) within the structure of CNNs. In the proposed method, the parameter vulnerability of CNNs is analyzed and the more vulnerable ones are duplicated. Thereafter, a correction layer detects and corrects the erroneous output activations based on the two duplicated values.

The proposed hardening mechanism effectively reduces the overhead with respect to the TMR-based hardening solution, possessing the same fault tolerance capabilities. However, it still incurs some overhead to the memory and performance of the hardened CNN. To further reduce this overhead, for the first time, a strategy is proposed for channel pruning based on the vulnerability of parameters to effectively shrink the size of CNNs with a negligible accuracy loss. In particular, we estimate the vulnerability of weight channels in CNNs, eliminate the least vulnerable ones to decrease the network's size, and then apply the hardening mechanism. The presented vulnerability-aware pruning provides the opportunity to eliminate any overhead caused by the protection mechanism on the designed hardened CNNs.

The contributions of this paper are as follows:

- Proposing a model-level hardening method for CNNs to enhance their fault tolerance during inference. The approach involves duplicating the parameters in channels more vulnerable to faults and incorporating a highly effective Error Detection and Correction (EDAC) Layer to correct erroneous feature maps.
- Proposing a channel pruning technique based on the parameter vulnerability that enables achieving a substantial reduction in the overhead incurred by the hardening mechanism.
- Results indicate that the proposed method allows hardened CNNs to perform reliably at error rates several orders of magnitude higher than those tolerated by the baseline CNN, achieved with merely 15% selective parameter duplication. Moreover, leveraging pruning allows hardened pruned CNNs to be more resilient than the unpruned ones, with up to 24% higher performance in terms of execution time.

In the rest of the paper, Section II presents the proposed CNN model hardening through duplicated vulnerable channels and EDAC layer. Section III indicates the results achieved by the proposed method. In Section IV the proposed parameter vulnerability based pruning is presented, and the overhead and resilience analyses are performed, and Section V concludes the paper.

II. CNN Model Hardening

In this section, the proposed hardening method to enhance the fault tolerance of CNN models is presented. This involves CNN architecture modification empowering them to inherently detect and correct faults. The method takes a pre-trained CNN and generates a hardened version that is executable by the target device.

A. Vulnerability Estimation

Vulnerability estimation of CNN's parameters reflects how they affect classification outputs in the presence of faults. Fault injection based approaches are very complex and time-consuming for addressing this task, whereas analytical approaches can estimate vulnerability fast and reasonably accurately [10]. This work adopts a vulnerability estimation approach introduced by [30] and adapts it to the parameters of a channel in a CNN. This approach is accurate with fault

injection results in [30]. Eq. (1) describes the vulnerability estimation for each channel:

$$Vulnerability_{channel} = \sum_{i=1, i\neq t}^{C} \frac{\sum_{w \in channel} |\frac{\partial (Z_i - Z_t)}{\partial w}|^2}{|Z_i - Z_t|^2} \quad (1)$$

In Eq. (1), the *vulnerability of a channel* with multiple weights w in a convolutional (CONV) layer of a CNN with C number of classes is estimated for a single input data. The output logits of the network corresponding to each output class is Z_i and the top class's logit is Z_t. This equation represents the effect of each channel on the output logits as a vulnerability estimation and a higher value represents a higher vulnerability of the corresponding channel. A similar equation is applied to the weights corresponding to a neuron in Fully Connected (FC) layers.

B. CNN Model Hardening Method

Subsequent to obtaining the vulnerability of channels of a pre-trained CNN, the CNN model is hardened by performing two steps:

- Duplication of the more vulnerable channels,
- Insertion of the Error Detection and Correction (EDAC) layer after each CONV/FC.

*1) **Channel Duplication**:* Fig. 2 illustrates how the duplication of parameter channels functions. A channel contains multiple weights for obtaining an output feature map (fmap) F_k resulting from the summation of weighted inputs. In the lth CONV layer with C^l output channels, a channel is a 3-dimensional array of weights X^l, Y^l, C^l. (In an FC layer, an output channel is a 1-dimensional weight array corresponding to a neuron). Duplicating a channel of parameters generates duplicated values in F_k which provides an opportunity to detect and correct errors produced by faults in parameters. In this method, a ratio of more vulnerable channels with respect to Eq. (1) are selected for duplication.

*2) **Error Detection And Correction (EDAC) Layer**:* After duplicating the vulnerable parameter channels, an EDAC layer is inserted into the CNN after each CONV and FC layer. The EDAC layer is meant to detect and correct errors in its incoming F_k from CONV/FC layers within the networks. One of the major challenges with 32-bit floating point data representation in general-purpose devices such as CPU and GPU is that faults may lead to overflows in CNNs producing Not-a-Number (NaN) values and corrupting the outputs. To address this issue, one of the primary operations in the EDAC layer is to replace any produced NaN value with 0 in the feature maps F_k.

Fig. 2 illustrates how the EDAC layer operates. The EDAC layer exploits a detection interval containing the minimum and maximum values in the channels of F_k that are the lower values $\{w_1, w_2, ..., w_n\}$ and the upper values $\{u_1, u_2, ..., u_n\}$, respectively. Detection intervals are obtained by profiling the CNN on the training dataset. It is assumed that the data distribution of training is representative enough to provide generic and valid detection intervals for the unseen data during the inference [31].

EDAC layer is aware of the duplicated and non-duplicated channels. In the duplicated channels, an error is detected and corrected in two cases:

- Both duplicated values in the corresponding channels are in the detection interval but are not equal. In this case, the minimum value between them is selected as the correct output F_k (case A in Fig. 2). The reason behind this correction is that CNNs are more resilient to small numbers [27].
- A value in a channel exceeds the detection interval, thus, the duplicated value that is in the detection interval is the correct value for the output F_k (case B and C in Fig. 2). If both duplicated values are not in the detection interval, the output F_k is set to 0 (case D in Fig. 2).

In the non-duplicated channels, faults are detected and corrected based on the detection intervals. If any value in the channel exceeds the corresponding detection interval output F_k sets to 0 (case E in Fig. 2). The rationale behind zeroing is that it eliminates the propagation of erroneous values within a DNN. Note, that the detection and correction are repeated for each element of the two-dimensional array of the feature map F_k.

To prevent faults from any immediate misclassification at the last layer, all output channels of the last layer in CNNs (i.e., neurons in the last FC layer) are duplicated and protected by an EDAC layer. It is worth mentioning that EDAC is implemented in a highly parallel way in Pytorch so that it can operate detection and correction on all duplicated and non-duplicated channels in parallel. Moreover, the hardened CNNs have the same accuracy as the baseline ones.

III. Resilience and Overhead Results

In this section, the results of fault injection experiments into the parameters of hardened CNNs are presented.

A. Fault Model

The parameters of a pre-trained CNN could be faulty at inference time due to several reasons, including soft errors, temperature or voltage variation, process variation, aging, etc. To examine the resilience of CNNs, we model faults in the parameters by flipping their bits considering different Bit Error Rates (BERs). To this end, any layer in the CNN's parameters, including convolutional, Fully Connected (FC), batch normalization and EDAC layers is subject to a fault injection campaign. We have developed the fault injection on top of Pytorch, and the data representation is IEEE-754 32-bit floating point. The number of bitflips in a layer is equal to $BER \times \#parameters \times 32$ in that layer. The fault injection simulations are performed on an NVIDIA 3090 GPU and any fault injection experiment is repeated 1000 times and the average accuracy drop is reported as the resilience metric. The experimented BERs are 10^{-8}, 5×10^{-8}, 10^{-7}, 5×10^{-7}, 10^{-6}, 5×10^{-6}, 10^{-5}, 5×10^{-5}, and 10^{-4}.

B. Baseline CNNs

The experiments in this work are performed on three deep CNNs: AlexNet and VGG-11 trained on Cifar-10 and VGG-16 trained on Cifar-100. Their baseline accuracy as well as the number of parameters and MAC operations are reported

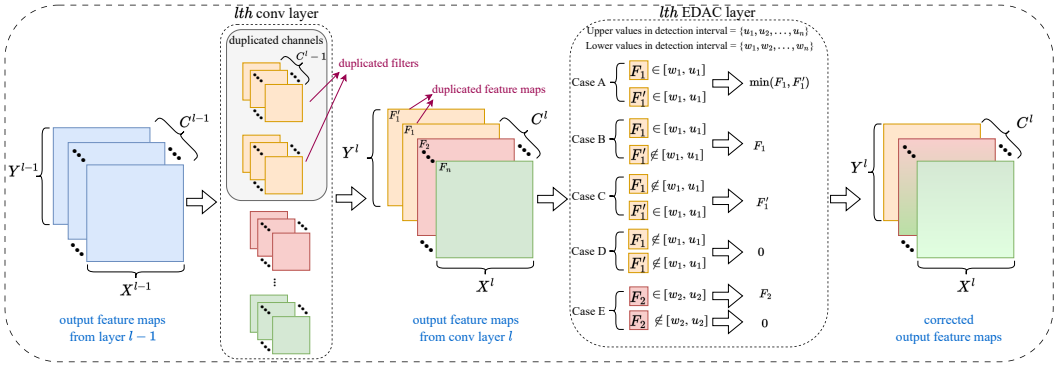


Fig. 2: Channel duplication and EDAC layer

in Table 1. The performance in terms of execution time of the CNNs over their test set is examined on an NVIDIA 3090 GPU coupled with an AMD Threadripper 3960X 24-core processor.

Note, that the accuracy of unprotected CNNs decreases drastically even at relatively low BERs. The unprotected AlexNet drops 26% at $BER = 5 \times 10^{-7}$ and the accuracy of unprotected VGG-11 and VGG16 drops 24.07% and 31.17% at $BER = 5 \times 10^{-8}$, respectively.

TABLE I: The baseline CNNs leveraged in this paper.

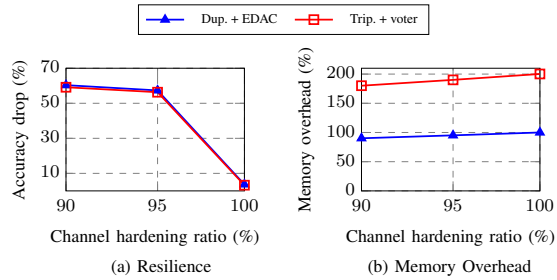
CNN	Dataset	Base accuracy	#parameters	#MACs	Performance (sec)
AlexNet	Cifar-10	73.15%	21,623,562	42,316,288	0.591
VGG-11	Cifar-10	92.85%	9,228,362	153,293,824	0.655
VGG-16	Cifar-100	73.20%	34,015,396	332,756,992	0.782

C. Hardening by Channel Duplication vs. Triplication

First, we demonstrate how EDAC performs if the detection intervals are not exploited for non-duplicated channels and compare it with a triplication-based correction performed by a voter. The voter takes three replicated fmaps in the corresponding channel and outputs the most repeated value. In the case where all three fmaps are different (if at least two replicated filters are faulty), the voter outputs the minimum value.

Fig. 3 presents the results for accuracy drop and memory overhead of *duplication + EDAC* vs. *triplication + voter* for AlexNet at BER=10^{-4} over different channel hardening ratios. A similar trend is observed for VGG-11 and VGG-16. The highlights that can be observed from the Figure are:

- *Duplication + EDAC* achieves a similar resilience to that of *triplication + voter* in terms of accuracy drop, with twice less memory overhead.
- The memory overhead is proportional to the channel duplication and triplication ratio. The memory overhead of the EDAC layer is negligible compared to the total memory and computational requirements of CNNs.
- A high resilience is achieved only at full channel hardening. At lower hardening ratios, although the more vulnerable channels are protected, the unprotected channels incur a high accuracy drop in CNNs due to the high BER.



(a) Resilience (b) Memory Overhead

Fig. 3: Resilience (a) and memory overhead (b) for AlexNet hardened by *duplication + EDAC* vs. *triplication + voter* at BER=10^{-4}, without applying detection intervals to Non-duplicated channels.

As observed, we need to apply a full channel duplication + EDAC to protect CNNs which leads to a significant overhead in the hardened CNNs compared to the unprotected ones. The hardened CNNs have double memory and computational requirements (100% overhead) and the execution time increases up to 1.83 times. To tackle this issue, we exploit the detection intervals in the non-duplicated channels to protect less vulnerable channels which leads to lower hardening ratios. It is presented in the next subsection.

D. Hardening by Selective Channels Duplication and EDAC Layer

In this subsection, we present the results for the selective channel duplication with EDAC layers as described in Fig. 2. In a pre-trained DNN, a ratio of the more vulnerable channels are duplicated and both duplicated and non-duplicated channels exploit detection intervals to be hardened at EDAC layer. Since the hardening method is at the model level, the performance in terms of execution time is influenced. Thus, we present the performance overhead on NVIDIA 3090 GPU in the paper.

Fig. 4 demonstrates the resilience and performance overhead for all the experimented CNNs at the highest BERs where the accuracy drop is not yet significant (lower than 5%). As observed, exploiting detection intervals in unprotected

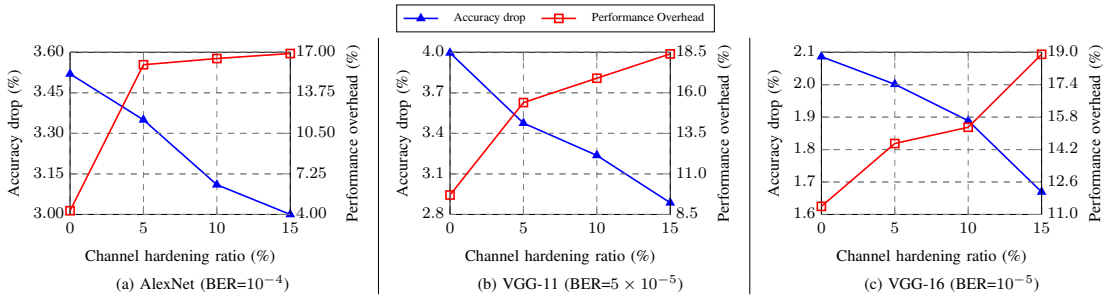


Fig. 4: Accuracy loss and performance overhead comparison for hardened AlexNet (a), hardened VGG-11 (b), and hardened VGG-16 (c), over different channel pruning ratios at the corresponding BERs where accuracy drop is lower than 5%.

channels has a remarkable effect on reducing the hardening ratio to achieve high resilience. It can be observed that by merely using detection intervals without channel duplication (*hardening ratio = 0%*), the accuracy drops at high BERs are lower than 4% with up to 11.4% performance overhead compared to the unprotected baseline CNNs. Nonetheless, increasing the channel hardening ratio improves the resilience without a significant performance overhead.

With a 15% channel hardening ratio the accuracy drop improves 17%, 38%, and 24% for AlexNet, VGG-11, and VGG-16 respectively, achieved by 6.7% to 12% longer execution time, compared to 0% hardening ratio.

As noted, EDAC layers exploiting detection intervals for all channels can significantly reduce the overhead of the hardened CNNs compared to the full duplication. However, a tangible overhead is incurred to CNNs due to hardening. The overheads are caused by both channel duplication and EDAC layer operations. To tackle this issue, we deploy a pruning method to reduce the size of baseline CNNs by removing the least vulnerable channels and applying EDAC to the most vulnerable ones, so the total overhead can be further reduced. This method is presented in the next section.

IV. Overhead Reduction by Parameter Vulnerability Based Pruning

As observed, although the introduced hardening technique exhibits a high resilience to CNNs, it lays a considerable overhead to them. To address this issue, we apply an effective structured channel pruning to CNNs to shrink their baseline size and open room for the hardening mechanism.

A. Vulnerability Based Pruning

Structured pruning is a well-known method for CNN models to reduce their size leading to optimizing their performance and resource utilization. In this method, a metric for the significance of the effect of parameters on the output accuracy is considered and the least important weights are removed from the CNN with a negligible accuracy loss.

Conventionally, the significance of the weights effect is examined by L1-norm which is shown to be effective [32]. In this work, we exploit Eq. (1) as the importance metric for channel parameters and remove a ratio of the least vulnerable channels from CONV and FC layers in CNNs. To avoid losing too much accuracy, we perform lightweight training on the pruned CNNs with 10 epochs using SGD with a learning rate of 0.001 on the training dataset. Fig. 5 shows that our vulnerability-aware pruning method is more effective than L1-norm pruning in terms of removing the channels of CNNs while the accuracy is still close to that of the baseline CNN.

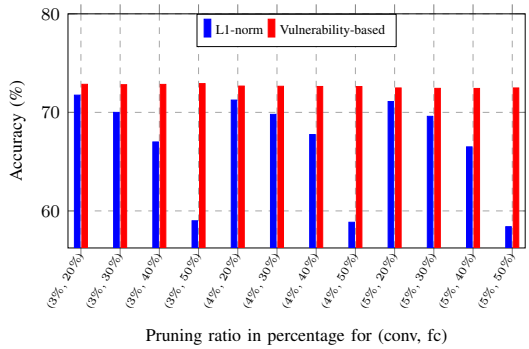


Fig. 5: Comparison of L1-norm pruning and vulnerability-based pruning in AlexNet.

To obtain the highest possible pruning ratios for each CNN, we perform an extensive exploration over different pruning ratios of CONV and FC layers to minimize the number of parameters and MAC operations maintaining the test accuracy within 1% of its unprotected baseline. Table II shows the selected pruning ratios for the experimented CNNs and their improved memory and computational requirements compared to the baseline ones. As it is observed, the pruned CNNs achieve from 1.18 to 6.19 times fewer parameters, 1.03 to 2.06 times fewer MAC operations, and 1% to 11.1% less execution time than the baseline ones.

TABLE II: Pruning ratio and normalized number of parameters and MAC operations and performance for each CNN.

CNNs	Conv. prun. ratio	FC prun. ratio	Pruned CNN Accuracy	Norm. #params to baseline	Norm. #MACs to baseline	Norm. perf. to baseline
AlexNet	5%	80%	72.38%	0.1615	0.4851	0.888
VGG-11	4%	35%	91.96%	0.847	0.9059	0.987
VGG-16	1%	15%	72.4%	0.826	0.9665	0.998

B. Resilience and Overhead Study of the Hardened Pruned CNNs

By shrinking the baseline CNNs using pruning, we have the opportunity to minimize the overhead of hardened CNNs

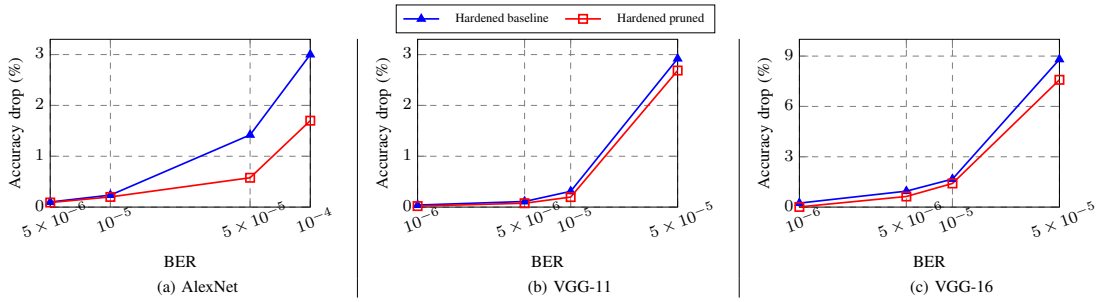


Fig. 6: Resilience comparison in terms of accuracy drop of hardened baseline and hardened pruned CNNs at different BERs with 15% channel hardening ratio.

compared to the baseline ones. Now, the pruned pre-trained CNNs are hardened by the method introduced in Section II. Their channel vulnerability is obtained, the more vulnerable channels are duplicated, and EDAC layers are implanted into the model with the corresponding detection intervals.

Fig. 6 illustrates how resilience is improved in the hardened pruned CNNs against hardened baseline ones over different BERs, with 15% channel hardening ratio. It is observed that the proposed pruning not only reduces the overhead of hardened CNNs but also improves their resilience.

Fig. 7 compares the performance overhead in terms of the execution time of different hardened CNNs on NVIDIA 3090 GPU. As observed, the overhead of *triplication + voter* is significantly higher than the other methods. On the other hand, hardened pruned CNNs have the best performance among the hardened CNNs. The resilience of the hardened CNNs is presented in Fig. 3-a, Fig. 4, and Fig. 6.

Throughout the results, the performance of 15% hardened pruned Alexnet, VGG-11, and VGG-16 is improved by 24%, 1%, and 4.7%, respectively, compared to the 15% hardened ones without pruning. It is noteworthy that the hardened pruned AlexNet has 6.06% less execution time than its un-protected baseline. The selective hardened pruned AlexNet, VGG-11, and VGG-16 require 81.40%, 2.67%, and 3.98% less memory, respectively, than their unprotected baseline to store their parameters.

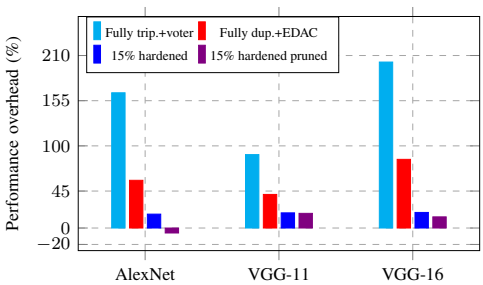


Fig. 7: Performance overhead comparison for hardened CNNs.

V. Conclusions

This paper presents a model-level hardening method for CNNs by selective channel duplication and EDAC layers. The proposed method enables CNNs to detect and correct faults inherently, at inference time. The hardened CNNs perform reliably at orders of magnitude higher error rates than unprotected CNNs with merely a 15% hardening ratio, yet incurring 12% performance overhead. To further minimize the incurred overhead by the hardening method, for the first time, a vulnerability-based pruning that improves resilience is presented. As a result, the hardened pruned CNNs achieve up to 24% higher performance than the un-pruned hardened CNNs.

References

[1] A. Krizhevsky *et al.*, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, 2012.

[2] S. Pouyanfar *et al.*, "A survey on deep learning: Algorithms, techniques, and applications," *ACM Computing Surveys*, vol. 51, no. 5, pp. 1–36, 2018.

[3] Y.-H. Chen *et al.*, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE journal of solid-state circuits*, vol. 52, no. 1, pp. 127–138, 2016.

[4] N. P. Jouppi *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *44th annual international symposium on computer architecture*, 2017, pp. 1–12.

[5] 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.

[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] M. H. Ahmadilivani *et al.*, "Enhancing fault resilience of qnns by selective neuron splitting," in *2023 IEEE 5th AICAS*, 2023, pp. 1–5.

[8] Y. Ibrahim *et al.*, "Soft errors in dnn accelerators: A comprehensive review," *Microelectronics Reliability*, vol. 115, p. 113969, 2020.

[9] F. Su *et al.*, "Testability and dependability of ai hardware: Survey, trends, challenges, and perspectives," *IEEE Design & Test*, 2023.

[10] M. H. Ahmadilivani *et al.*, "A systematic literature review on hardware reliability assessment methods for deep neural networks," *ACM Comput. Surv.*, vol. 56, no. 6, jan 2024.

[11] 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.

[12] M. H. Ahmadilivani *et al.*, "Special session: Reliability assessment recipes for dnn accelerators," in *2024 VTS*. IEEE, 2024.

[13] 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.

[14] 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.

[15] 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.

[16] F. Libano *et al.*, "Selective hardening for neural networks in fpgas," *IEEE Transactions on Nuclear Science*, vol. 66, no. 1, pp. 216–222, 2018.
[17] G. Abich *et al.*, "The impact of soft errors in memory units of edge devices executing convolutional neural networks," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 69, no. 3, pp. 679–683, 2022.
[18] K. Zhao *et al.*, "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.
[19] J. J. Zhang *et al.*, "Analyzing and mitigating the impact of permanent faults on a systolic array based neural network accelerator," in *2018 IEEE 36th VTS*. IEEE, 2018, pp. 1–6.
[20] K. T. Chitty-Venkata and A. K. Somani, "Model compression on faulty array-based neural network accelerator," in *2020 IEEE 25th PRDC*. IEEE, 2020, pp. 90–99.
[21] E. Ozen and A. Orailoglu, "Snr: S queezing n umerical r ange defuses bit error vulnerability surface in deep neural networks," *ACM Transactions on Embedded Computing Systems*, vol. 20, no. 5s, pp. 1–25, 2021.
[22] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer, "A survey of quantization methods for efficient neural network inference," in *Low-Power Computer Vision*. Chapman and Hall/CRC, 2022, pp. 291–326.
[23] N. Cavagnero *et al.*, "Fault-aware design and training to enhance dnns reliability with zero-overhead," *arXiv preprint arXiv:2205.14420*, 2022.
[24] U. Zahid *et al.*, "Fat: Training neural networks for reliable inference under hardware faults," in *2020 IEEE ITC*. IEEE, 2020, pp. 1–10.
[25] S. Lee and J. Yang, "Value-aware parity insertion ecc for fault-tolerant deep neural network," in *2022 DATE*. IEEE, 2022, pp. 724–729.
[26] Z. Chen *et al.*, "A low-cost fault corrector for deep neural networks through range restriction," in *2021 51st Annual IEEE/IFIP DSN*. IEEE, 2021, pp. 1–13.
[27] L. 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.
[28] 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.
[29] M. S. Ali *et al.*, "Erdnn: Error-resilient deep neural networks with a new error correction layer and piece-wise rectified linear unit," *IEEE Access*, vol. 8, pp. 158 702–158 711, 2020.
[30] A. Mahmoud *et al.*, "Hardnn: Feature map vulnerability evaluation in cnns," *arXiv preprint arXiv:2002.09786*, 2020.
[31] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.
[32] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

Appendix 7

VII
M. H. Ahmadilivani, J. Raik, M. Daneshtalab, and A. Kuusik. Analysis and Improvement of Resilience for Long Short-Term Memory Neural Networks. In *IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, pages 1–4. Juan-Les-Pennes, France, 2023

Analysis and Improvement of Resilience for Long Short-Term Memory Neural Networks

Mohammad Hasan Ahmadilivani[1], Jaan Raik[1], Masoud Daneshtalab[1,2], and Alar Kuusik [1]
[1]Tallinn University of Technology, Tallinn, Estonia
[2]Mälardalen University, Västerås, Sweden
[1]{mohammad.ahmadilivani, jaan.raik, alar.kuusik}@taltech.ee
[2]masoud.daneshtalab@mdu.se

2023 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT) | 979-8-3503-1500-4/23/$31.00 ©2023 IEEE | DOI: 10.1109/DFT59622.2023.10313559

***Abstract*—The reliability of Artificial Neural Networks (ANNs) has emerged as a prominent research topic due to their increasing utilization in safety-critical applications. Long Short-Term Memory (LSTM) ANNs have demonstrated significant advantages in healthcare applications, primarily attributed to their robust processing of time-series data and memory-facilitated capabilities. This paper, for the first time, presents a comprehensive and fine-grain analysis of the resilience of LSTM-based ANNs in the context of gait analysis using fault injection into weights. Additionally, we improve their resilience by replacing faulty weights with zero, enabling ANNs to withstand environments that are up to 20 times harsher while experiencing up to 7 times fewer critical faults than an unprotected ANN.**

I. Introduction

The realm of Deep Learning (DL) applications is constantly expanding due to its ability to effectively address various complex tasks. In healthcare, Artificial Neural Networks (ANNs) have found extensive application in disease diagnosis, treatment, and anomaly prediction [1], [2], leveraging their superior image and time-series data processing capabilities [3], [4]. Given the safety-critical nature of healthcare applications, ensuring the hardware reliability of DL devices becomes paramount. Reliability is the probability of hardware functioning correctly in the presence of faults, which may occur due to soft errors, temperature variation, aging, etc. [5].

Long Short-Term Memory (LSTM) neural networks possess remarkable capabilities to retain long-term information, making them highly effective in analyzing time-series data. This unique feature has rendered them highly desirable for anomaly detection in healthcare applications. Gait analysis, a vital diagnostic tool in orthopedics, neurology, wellness assessment, and rehabilitation, benefits greatly from the utilization of LSTMs for processing time-series data typically collected by wearable Micromechanical Motion Sensors (MEMS) [6], [7].

Numerous research works have extensively examined the reliability of Convolutional Neural Networks (CNNs) in safety-critical applications, including in healthcare. These studies have demonstrated that faults in the weights of CNNs can drastically influence their accuracy [8], [9]. Notably, memory faults affecting the weights of CNNs can result in a significant decrease in accuracy [10]. Consequently, various techniques for enhancing reliability have been proposed in the literature [11].

Although several studies have exploited LSTMs in the aforementioned applications, there has been a notable absence of research studying the effects of faults on the LSTM weights and potential approaches for enhancing their resilience [9]. To the best of our knowledge, this work represents the first attempt to address these critical issues.

In this work, we investigate the resilience of different LSTM-based ANNs detecting gait abnormalities in time-series data and study their resilience using Fault Injection (FI) into the weights. Moreover, by analyzing the distribution of weight values and assessing the resilience of the ANNs, we propose a method to enhance their resilience. The contributions are as follows:

- Conducting a comprehensive and fine-grained analysis of the resilience of LSTM-based ANNs by injecting faults into various sets of their weights, leading to identifying the most susceptible ones,
- Introducing a resilience improvement method for LSTM-based ANNs based on replacing the faulty weights with zero, taking into account the distribution of weight values,
- Demonstrating the effectiveness of the proposed method by achieving significantly more resilient ANNs that meet constraints of the application successfully.

Through the rest of the paper, Section II introduces LSTM-based ANNs, Section III proposes the method for their resilience analysis and improvement, Section IV presents and analyzes the results, and Section V concludes the paper.

II. Preliminaries on LSTM-based ANNs

LSTMs have recursive loops among their neurons' interconnections and enable memorizing arbitrary information for a long time. They consist of stacking LSTM layers in which LSTM cells are present. The operations of an LSTM cell are described in Eq. 1, where x_t is a time-series input at time t, C_{t-1} and h_{t-1} are recursive inputs i.e., the last outputs of the LSTM cell, U and W are the weights of the cell that involve with inputs and recursive inputs, respectively, and B is bias parameters.

$$\begin{cases} i_t = sigmoid(x_t U^i + h_{t-1} W^i + B^i) \\ f_t = sigmoid(x_t U^f + h_{t-1} W^f + B^f) \\ g_t = tanh(x_t U^g + h_{t-1} W^g + B^g) \\ o_t = sigmoid(x_t U^o + h_{t-1} W^o + B^o) \\ C_t = f_t \times C_{t-1} + i_t \times g_t \\ h_t = tanh(C_t) \times o_t \end{cases} \tag{1}$$

There are two sets of weight parameters in LSTM: 1) U that is multiplied into the inputs and extracts the features of the current input data, 2) W that is multiplied into the recursive inputs and manages the information from the previous outputs through time to facilitate the cells with memorizing the features.

To classify time-series data in an LSTM-based ANN, a Fully-Connected (FC) layer can be attached to the LSTM layer. An FC layer contains several neurons each functioning as shown in Eq. 2, where *n* input activations from a previous layer *l-1* (denoted as X^{l-1}) are fed to the *i*-th neuron in layer *l* (denoted as N_i^l) which performs a weighted summation of inputs (*W* and *b* represent weights and bias, respectively). Thereafter, the result passes through an activation function which is ***ReLU*** in this paper.

$$N_i^l = ReLU(\sum_{j=0}^{n} X_j^{l-1} \times W_{ij}^l + b^l) \tag{2}$$

The work is supported in part by the EU through European Social Fund in the frames of the "ICT programme" ("ITA-IoIT" topic).

979-8-3503-1500-4/23/$31.00 ©2023 IEEE

In this work, LSTMs are employed for abnormality detection in a human walking motion. Their structure is demonstrated in Fig. 1, comprises an LSTM layer with multiple cells, followed by an FC layer with an equal number of neurons as the LSTM cells, and an output FC layer with 2 output neurons for binary classification of abnormality detection. Input data (x_t) are the samples of walking steps through time.

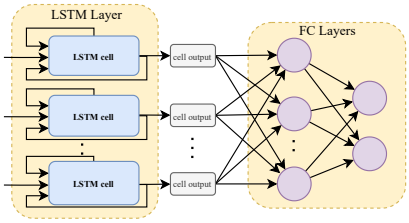


Fig. 1: LSTM-based ANNs in this work for gait abnormality detection.

III. Proposed Method: Resilience Analysis and Improvement for LSTM-based ANNs

A. LSTM-Based ANNs Under Study

Three adopted LSTM-based ANNs in this paper have the structure shown in Fig. 1, each has two output classes representing normal and abnormal steps. The examined ANNs called ANN-*n* where *n* represents the number of cells in the LSTM layer and also the number of neurons in the first FC layer.

B. Fault Injection (FI) Method

Faults in memory can result in a remarkable accuracy drop in ANNs. In this work, faults are assumed to occur in memory that impacts the weight parameters of ANNs. We consider different Bit Error Rates (BERs) to model accumulated bitflips throughout the parameters. Notably, faults in bias are not considered as their effect was negligible in our preliminary experiments.

We perform a random FI into the weights at each BER before a full inference. Then we obtain the outputs of the network for further resilience analysis and repeat the experiments several times in order to reach high-confidence results.

We inject faults into randomly selected weights considering different scenarios: 1) *Model-wise FI:* weights across the ANN, 2) *LSTM FI:* weights of the LSTM layer, 3) *FC FI:* weights of FC layers, 4) *Recursive weights (W) FI:* weights of the LSTM layer that involve the recursive inputs of cells (vector W in Eq. 1), 5) *Input weights (U) FI:* weights of LSTM layer that involve the time-series inputs (vector U in Eq. 1).

C. Resilience Evaluation Method

We evaluate the resilience using accuracy loss and F1-score loss under the FI campaign representing the difference between the fault-free and faulty execution of the ANNs. The average accuracy and F1-score over all the FI experiments are considered.

We analyze the effect of faults on the output to classify them in line with prior studies [9]. In each FI experiment, the outputs are classified into one of the following classes: 1) **Masked:** outputs in faulty and fault-free executions are the same, 2) **Non-critical Silent Data Corruption (SDC):** output values are different in faulty and fault-free executions but the classification result is the same, 3) **Critical SDC:** output values and classification are different in faulty and fault-free executions, 4) **Detected Unrecoverable Error (DUE):** ANN produces 'NaN' values in the output considered as *system exception*.

D. Resilience Improvement Method

First, we observe the weight values distribution. Fig. 2 depicts the value distribution of weights across each network separately. It is observed that in all ANNs the range of weights is limited to a small range as well as most of the values are close to 0.

We propose a resilience improvement method based on this observation considering two modes: 1) offline mode, 2) deployment. In the offline mode (fault-free), the minimum and maximum weight values are obtained. Thereafter, in the deployment, to detect and correct the faulty weights before an inference, we perform these operations iteratively: 1) Comparing all weights with the obtained range, 2) Setting the exceeded weights to 0. Therefore, faulty weights that are larger than the expected values are found and removed at run-time.

IV. Experiments

A. Experimental Setup

Three fault-free pre-trained LSTMs adopted in this work, detect abnormal gaits in a time-series dataset obtained from clinical experiments on 15 patients including their normal and abnormal steps. Table I demonstrates the accuracy and F1-score as well as their number of weights. *It is strictly determined by the application that the accuracy of an ANN should be higher than 80% and its F1-score should be more than 70%.* Also, ANNs are employed by a general-purpose computer that supports 32-bit floating-point IEEE-754 data representation.

Table I: Accuracy, F1-score, and the number of different weight sets of the LSTM-based ANNs in this work.

	Accuracy	F1-score	#weights in U	#weights in W	#weights in FC
ANN-20	93.35%	80.33%	320	1600	440
ANN-30	94.73%	85.39%	480	3600	960
ANN-40	95.79%	88.57%	640	6400	1680

The number of injected faults is determined by BERs between 0.0001 to 0.5 to observe the full possible spectrum of the pieces of evidence. We repeat each FI experiment 2000 times and report the average accuracy and F1-score. For the fault classification results, we save the outputs of each FI campaign and label them according to Subsection III-C, and in the end, the rate of each

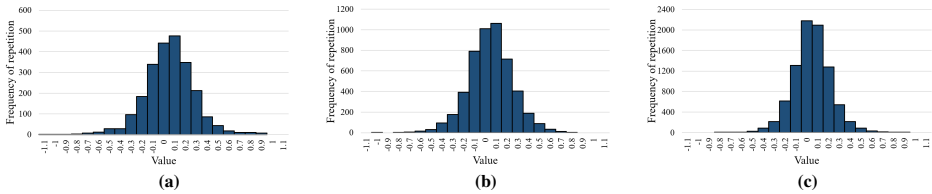


Fig. 2: The distribution of values of weights for a) ANN-20, b) ANN-30, c) ANN-40, respectively.

fault class over the 2000 experiments is reported. All experiments are implemented in PyTorch and are run on an A100 NVIDIA GPU along with an AMD EPYC 64-core CPU.

B. Resilience Analysis Results

1) Model-wise Resilience Analysis: The weights across the model are corrupted in FI experiments using different BERs, and average accuracy loss and F1-score loss are reported in Fig. 3 and Fig. 4, respectively. Regarding the application constraints (accuracy $>= 80\%$ and f1-score $>= 70\%$), all networks fail to function at the BERs higher than 0.005.

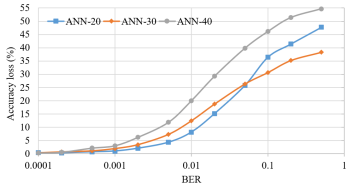


Fig. 3: ANNs' accuracy loss in a model-wise FI.

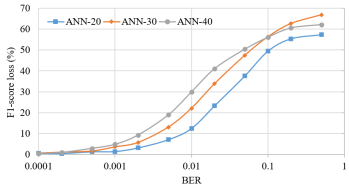


Fig. 4: ANNs' F1-score loss in a model-wise FI.

Assuming that only 0.5% of the weights across the model are faulty (i.e., $BER = 0.005$) the average accuracy of ANN-20, ANN-30, and ANN-40 decreases by 4.29%, 7.26%, and 11.86%, their F1-score is reduced by 7.12%, 13.11%, and 19.02%, respectively. The findings indicate a general trend: larger networks exhibit lower resilience compared to smaller ones. it can be attributed that larger networks possess a greater number of parameters, making them more susceptible to faults.

In addition, faults classification results are depicted in Fig. 5 showing that at low BERs, most of the faults are either masked or non-critical. However, when BER is 0.005 or more, nearly all the faults are propagated to the output as SDC. At $BER = 0.005$, critical SDC for ANN-20, ANN-30, and ANN-40 is 5.44%, 8.79%, and 13.69% and the rest of the faults are mostly non-critical SDC (less than 0.1% are masked). It can be observed that DUE appears only when BER is very high and still its ratio is not remarkable.

2) Resilience Analysis of LSTM vs FC: Fig. 6 presents the accuracy loss of ANNs when weights in the LSTM layer are faulty. It is observed that for BERs below 0.005, the resilience of both, LSTM and FC layers are approximately similar, i.e., the variation in accuracy loss has a difference of less than 0.5%. On the other hand, when BER is 0.005 and higher, in nearly all cases accuracy loss in LSTM FI is remarkably higher. Therefore, in all ANNs, the weights of the LSTM layer are more vulnerable than the ones in the FC layers, especially at high BERs.

Fig. 7 illustrates the fault classification for ANN-20 in LSTM and FC FI, while results for other ANNs have a similar trend. FC layers outperform the LSTM layer in fault-masking, also evidenced by the accuracy loss results. This is due to the dense connections and the fault-masking capability of ReLU in FC layers. Moreover, faulty weights in an LSTM layer lead to more critical cases, since it memorizes input patterns and faults in its parameters compromise ability. In addition, in LSTM FI, the DUE rate is consistently 0, whereas, in FC FI, DUE occurs at high BERs. This is caused by sigmoid and tanh activation functions within LSTM cells, which effectively eliminate large values. In contrast, FC layers employ ReLU, allowing the propagation of large positive values, leading to DUE occurrences.

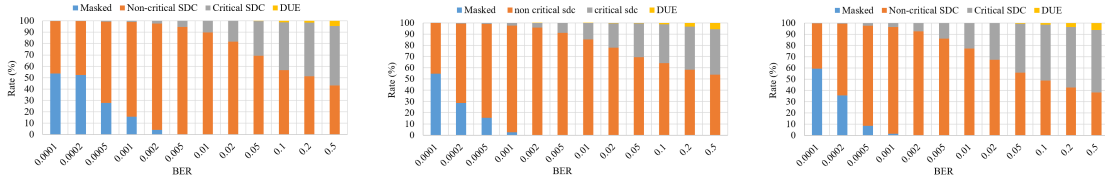
3) Inter-LSTM Resilience Analysis- FI into U vs W: Since LSTM layers are less resilient than FC layers in the experimented ANNs, we have further analyzed the resilience of weights inside LSTM layers. The results for Recursive weights (W) FI and Input weights (W) FI are presented in Fig. 8.

It is observed that in all ANNs, *W* weights are remarkably more vulnerable. At $BER = 0.005$, the accuracy drop resulting from FI into *W* parameters is 2.41x, 3.25x, and 4.55x worse than that of FI into *U*, for ANN-20, ANN-30, and ANN-40, respectively. The reason is that *W* parameters involve memorizing the incoming time-series data through the recursive inputs of the LSTM layer. Therefore, faults in these parameters persist in influencing the recursive inputs through time and corrupt the memorizing ability of the network more than the other weights.

C. Resilience Improvement of LSTM-based ANNs

To show the efficacy of the proposed method, we have applied it to ANNs and performed a model-wise FI to obtain the results of accuracy loss, F1-score loss, and fault classification. Fig. 9 and Fig. 10 show how the proposed method improves the accuracy and F1-score of ANNs in different BERs. Considering application constraints, protected ANN-20 can operate when $BER <= 0.05$ (10x higher than the unprotected one). In addition, the protected ANN-30 and ANN-40 can meet the requirements up to $BER = 0.1$ meaning that they can tolerate up to 20x higher BERs.

Regarding the fault classification results, the critical SDC rate and DUE rate have been significantly reduced in comparison with the unprotected ANNs in Fig. 5. As a result, the critical SDC rate is reduced 7x at $BER = 0.05$ for ANN-20 as well as 3.96x and 5.26x at $BER = 0.1$ for ANN-30 and ANN-40 respectively. Our protection mechanism results in 0.65x and 1.72x less DUE at $BER = 0.05$ for ANN-20 and $BER = 0.1$



(a) **(b)** **(c)**

Fig. 5: Fault classification in model-wise FI for a) ANN-20, b) ANN-30, c) ANN-40.

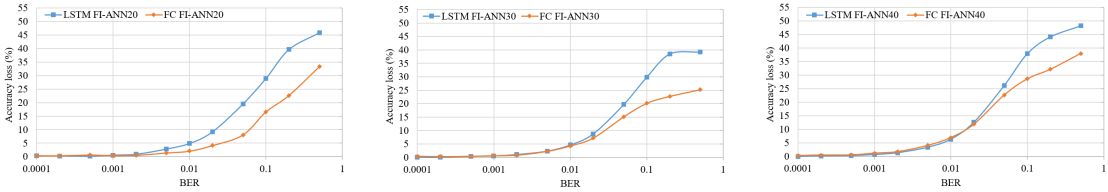
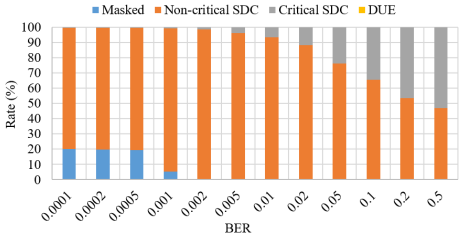
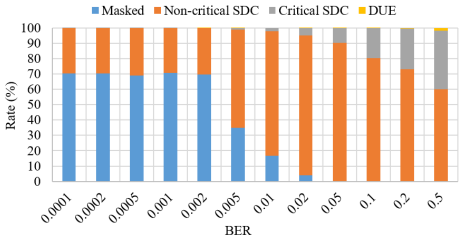


Fig. 6: Comparison of ANNs' accuracy loss with FI in LSTM vs FC for a) ANN-20, b) ANN-30, c) ANN-40.



(a) ANN-20 LSTM FI



(b) ANN-20 FC FI

Fig. 7: ANNs-20's fault classification with FI in LSTM vs FC.

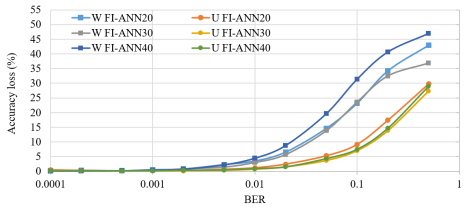


Fig. 8: Comparison of ANNs' accuracy loss with FI in *U* and *W* weights in the LSTM layer.

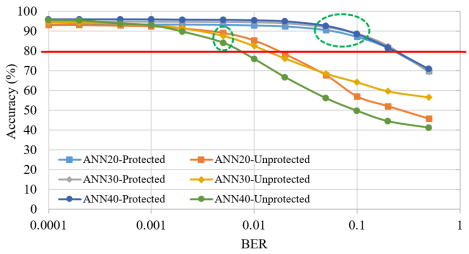


Fig. 9: Accuracy improvement of ANNs.

for ANN-30 respectively. In addition, it has removed any DUE in ANN-40 throughout FI experiments. The results reveal that the straightforward act of replacing zero with the faulty weights can effectively improve the resilience of the LSTM-based ANNs as has been shown to be feasible for CNNs [12].

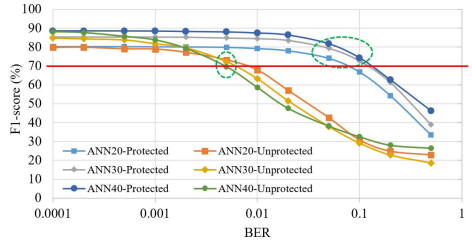


Fig. 10: F1-score improvement of ANNs.

V. CONCLUSION

This paper presents a pioneering study that highlights the criticality of conducting reliability assessments for LSTM-based ANNs. Extensive experiments using fault injection have been performed to thoroughly examine the effect of faults on various sets of weights in ANNs. Notably, our findings demonstrate that recursive weights within LSTM cells are particularly vulnerable parameters in LSTM-based ANNs. Furthermore, a lightweight yet effective resilience improvement technique has been proposed which involves replacing faulty weights with zeros. Remarkably, the implementation of this technique results in ANNs experiencing 7 times fewer critical faults while successfully operating in environments up to 20 times harsher than unprotected networks.

REFERENCES

[1] R. Manne and S. C. Kantheti, "Application of ai in healthcare: chances and challenges," *Current Journal of Applied Science and Technology*, vol. 40, no. 6, pp. 78–89, 2021.

[2] E. J. Harris *et al.*, "A survey of human gait-based ai applications," *Frontiers in Robotics and AI*, vol. 8, p. 749274, 2022.

[3] V. Sze *et al.*, "Efficient processing of dnns: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.

[4] B. Lim and S. Zohren, "Time-series forecasting with dl: a survey," *Philosophical Transactions of the Royal Society A*, vol. 379, pp. 202–209, 2021.

[5] 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.

[6] Y. Su and C.-C. J. Kuo, "On extended lstm and dependent bidirectional rnn," *Neurocomputing*, vol. 356, pp. 151–161, 2019.

[7] R. M. Coelho *et al.*, "Real-time walking gait terrain classification from foot-mounted inertial measurement unit using conv-lstm nn," *Expert Systems with Applications*, vol. 203, p. 117306, 2022.

[8] K. Adam *et al.*, "A selective mitigation technique of soft errors for dnn models used in healthcare applications: Densenet201 case study," *IEEE Access*, vol. 9, pp. 65 803–65 823, 2021.

[9] M. H. Ahmadilivani *et al.*, "A systematic literature review on hardware reliability assessment methods for dnns," 2023.

[10] 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.

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

[12] L.-H. Hoang *et al.*, "Ft-clipact: Resilience analysis of dnns and improving their fault tolerance using clipped activation," in *DATE*, 2020, pp. 1241–1246.

Appendix 8

VIII
B. Parchekani, S. Nazari, M. H. Ahmadilivani, A. Azarpeyvand, J. Raik, T. Ghasempouri, and M. Daneshtalab. Zero-Memory-Overhead Clipping-Based Fault Tolerance for LSTM Deep Neural Networks. In *IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, pages 1–4. Oxforshire, United Kingdom, 2024

Zero-Memory-Overhead Clipping-Based Fault Tolerance for LSTM Deep Neural Networks

Bahram Parchekani[1], Samira Nazari[1], Mohammad Hasan Ahmadilivani[2], Ali Azarpeyvand[1,2], Jaan Raik[2], Tara Ghasempouri[2], and Masoud Daneshtalab [2,3]

[1]University of Zanjan, Zanjan, Iran

[2]Tallinn University of Technology, Tallinn, Estonia

[3]Mälardalen University, Västerås, Sweden

[1]{bahram.parchekani, samira.nazari, azarpeyvand}@znu.ac.ir

[2]{mohammad.ahmadilivani, ali.azarpeyvand, jaan.raik, tara.ghasempouri}@taltech.ee

[3]masoud.daneshtalab@mdu.se

2024 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT) | 979-8-3503-6688-4/24/$31.00 ©2024 IEEE | DOI: 10.1109/DFT63277.2024.10753533

***Abstract*—Long Short-Term Memory (LSTM) Deep Neural Networks (DNNs) have shown superior accuracy in predicting and classifying time-series data. This has made them suitable for many applications, including safety-critical ones, such as healthcare, where fault tolerance is a major concern. Until now, fault resilience and mitigation in LSTMs have not been thoroughly explored, raising concerns about exploiting them in safety-critical use cases. This work, first, extensively explores the effect of faults on LSTM DNNs using fault injection into parameters. Moreover, the paper presents two effective zero-memory-overhead fault tolerance techniques for LSTM DNNs to protect them against random faults in their parameters. Experimental results indicate that the proposed techniques can improve fault tolerance of LSTM-based DNNs up to 278.6 times concerning unprotected ones.**

***Index Terms*—Hardware Reliability, Fault Tolerance, Neural Networks, LSTMs, Healthcare.**

I. Introduction

Deep Learning (DL) is being increasingly employed in safety-critical applications, e.g., healthcare and automotive, due to their outstanding accuracy in classification, prediction, etc. [1], [2]. In this domain of applications, hardware reliability is a significant concern [3]–[5]. Hardware reliability is defined as the probability of hardware performing correctly with the presence of faults. Faults may occur due to temperature variation, aging, soft errors, etc., and flip the bits in logic or memory [3], [6], [7]. Such an effect may lead to catastrophic results in safety-critical applications. With transistor scaling, fault rates in memories have been increased, which threatens the hardware reliability significantly [8].

Healthcare applications exploit Deep Neural Networks (DNNs) extensively for various tasks such as diagnosis, treatment, and prediction of diseases and anomalies [9], [10] because of their outstanding strength in processing time-series data [11]. Long Short-Term Memory (LSTM) DNNs are a subset of Recursive Neural Networks (RNNs) that are remarkably effective in classifying and predicting time-series data. They retain long-term information through time via feedback loops making them highly desirable for disease prediction in healthcare applications [11].

Throughout the literature, several research works have thoroughly studied the reliability of DNNs in safety-critical applications [4], [6]. Fault injection at the software simulation level is the predominant method for analyzing and evaluating the reliability of DNNs due to their fast execution time [3]. The related studies indicate that faults in memory impacting the parameters of DNNs result in a substantial reduction of their accuracy [8], [12]. Therefore, various methods for improving their fault tolerance are proposed. Model-level fault tolerance [13]–[15] significantly impacts their resilience against memory faults.

979-8-3503-6688-4/24/$31.00 ©2024 IEEE

Nearly all existing works study the reliability of Convolutional Neural Networks (CNNs) for image classification and object detection [3]. Although LSTMs are widely deployed in safety-critical applications, their fault tolerance is not extensively explored [3]. In a recent paper, the effect of faults in the computational units of LSTMs for image classification in automotive is studied and a hardware-level fault tolerance approach is proposed [16]. In another prior work, a fine-grain and comprehensive resilience analysis for different sets of parameters in LSTMs is performed [17]. It is shown that their resilience can be remarkably improved by detecting the faulty weights and setting them to 0.

In this paper, we perform resilience analysis on StageNet [18] as a case study in healthcare applications for disease prediction. The contributions of this paper are as follows:

- Performing a comprehensive resilience analysis for various LSTM-based DNNs using fault injection in parameters, leading to observing the effect of different DNN structures and identification of critical bits;
- Proposing two fault-tolerant methods for LSTM-based DNNs, Weights Bit Clipping (WBC) and Activations Value Clipping (AVC), to effectively reduce the impact of faults in parameters on LSTM-based DNNs;
- Demonstrating the efficacy of the proposed methods in DNNs' resilience, leading up to 278.6 times less critical cases in the outputs caused by faulty parameters.

Section II provides a background on StageNet. Section III presents the proposed method for its resilience analysis and improvement. Section IV provides the results, and Section V concludes the paper.

II. Preliminaries on LSTMs and StageNet

StageNet [18] is an LSTM-based DNN designed for disease prediction in healthcare. It predicts the stage of a patient's disease according to the characteristics of the tests performed by the patient through time. Fig. 1 illustrates the overall structure of StageNet. It is composed of an LSTM layer for characterizing the disease stage over time, a convolutional (CONV) module, and an output Fully Connected (FC) layer to output the predicted disease condition. The CONV module contains one layer in parallel with two FC layers, and their outputs are multiplied point-wise.

StageNet receives time-series data as inputs according to the patient visits $(v_1, v_2, \cdots, v_t)$ which contain numerical clinical features at different times $(\Delta_1, \Delta_2, \cdots, \Delta_t)$. They pass through the LSTM layer with multiple cells inferring the variation of a

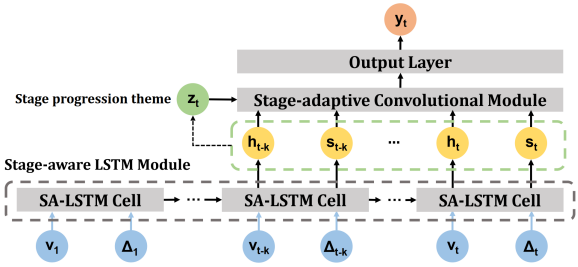


Fig. 1: Overal structure of StageNet [18].

patient's health stage considering their current status. The produced results are forwarded to the CONV module to learn patterns of the disease stages. Afterward, the output layer performs the classification for disease stage and risk prediction.

To measure the performance of StageNet, the following metrics are evaluated: **Accuracy:** This metric represents the percentage of correct predictions of StageNet compared to the expected outputs. **AUROC:** This metric refers to the area under the receiver operating characteristic curve illustrating the trade-off between true positive rate and false positive rate. It shows how a classifier can discriminate the positive and negative classes and it is extensively used when a dataset is imbalanced. Since the distribution of data in the dataset between different disease stages is imbalanced, AUROC is a more suitable metric to show the performance of StageNet. Therefore, for resilience analysis of this LSTM-based DNN, we consider AUROC drop along with accuracy drop.

III. Fault Propagation and Mitigation in LSTM-based DNNs

A. Resilience Evaluation Using Fault Injection

To analyze the resilience of DNNs comprehensively, random bit flips are applied throughout the DNN's weights to assess the overall network's behavior in the presence of faults. To quantify the resilience analysis, once the average performance metrics (Accuracy and AUROC) for a DNN under test are obtained, their difference with the fault-free metrics is considered as accuracy drop and AUROC drop.

Furthermore, the output effect of faults is categorized into the following classes, to quantify the effect of faults on the DNNs' outputs: **Masked:** Outputs remain identical between faulty and fault-free executions. **Non-critical Silent Data Corruption (SDC):** Output values differ between faulty and fault-free executions, while the classification result remains consistent. **Critical SDC:** Both output values and classification results differ between faulty and fault-free executions. **Detected Unrecoverable Error (DUE):** The DNN generates "NaN" values in the output, indicative of a system exception.

To identify the critical bits in DNNs, we perform a bit-wise fault injection experiment throughout the DNNs. In such an experiment, one bit is considered as the target and it is flipped in all parameters and the inference is performed and the performance metrics are measured. The bit that has the highest impact on the performance metrics is identified as the most critical bit.

B. Resilience Improvement for LSTM-based DNNs

We propose two model-level fault tolerance techniques with zero memory overhead: 1) Weights Bit Clipping (WBC), and 2) Activations Value Clipping (AVC). In the WBC method, all weights of fault-free DNN models are profiled and their bit patterns are analyzed. As a result, a consistent bit pattern is revealed in the DNNs under study. Moreover, using fault injection, the most critical bit is identified. To this end, an extensive exploration of different bit flips is carried out and the resilience is measured for each bit. Therefore, the method suggests clipping the most critical bits to a certain value throughout the DNN, before an inference.

In the AVC method, first, the input values to each activation function of the LSTM cells as well as the CONV and FC layers in DNNs are profiled and their maximum and minimum values are obtained, during a fault-free forward pass with validation data. The obtained values are then utilized for detecting faults that is when an input to corresponding activation functions exceeds the determined value range. Once a fault is detected in a forward pass, the corresponding value is clipped. If an activation value falls below the minimum range value, it is set to that minimum threshold. Conversely, if an activation value exceeds the maximum profiled value, it is set to the maximum threshold.

Ultimately, the effectiveness of WBC and AVC is evaluated by fault injection to determine how they mitigate the fault impact and which technique offers superior fault tolerance in LSTM DNNs. By comparing these methods, we aim to identify the more robust approach for enhancing the reliability of LSTM neural networks in the presence of faults.

IV. Experiments

A. Experimental Setup

To evaluate and improve the resilience of LSTM-based DNNs against faults in parameters, four variations of StageNet are experimented. Two variations represent the full StageNet model which includes CONV layer, with different numbers of LSTM cells (384 and 72), and the two variations that exclude the convolutional module, containing only LSTM layer, with 384 and 72 cells.

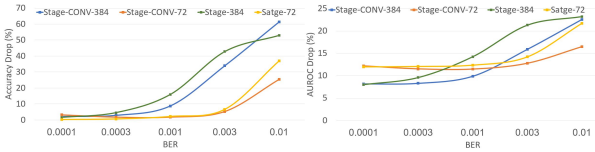
Test data is sourced from the Medical Information Mart for Intensive Care (MIMIC-III) dataset, which includes 17 physiological variables recorded at each visit. It is transformed into a 76-dimensional vector comprising numerical and one-hot encoded categorical clinical features for 33,678 patients.

Baseline metrics, including accuracy and AUROC in fault-free executions, are summarized in Table I, alongside the number of parameters for each model. All models were executed on a CPU supporting 32-bit floating-point IEEE-754 data representation.

Table I: Accuracy, AUROC, and the number of different weight sets of the LSTM-based ANNs in this work.

	Accuracy	AUROC	#weights in LSTM	#weights in CONV	#weights in FC
Stage-CONV-384	94.94%	79.21%	738618	442368	24960
Stage-CONV-72	83.75%	76.75%	48764	51840	1800
Stage-384	90.28%	79.29%	738618	0	384
Stage-72	77.28%	76.97%	48764	0	72

We conduct Fault Injection (FI) across all weights in the DNNs under study. The number of injected faults is determined using a Bit Error Rate (BER) ranging from 0.0001 to 0.01, covering a comprehensive range of potential errors. FI is repeated 1000 times to ensure an acceptable confidence level. For each iteration, a drop in accuracy and AUROC is obtained, and faults are classified according to Subsection III-A. Eventually, the average results over all iterations are reported in the paper.



(a) Accuracy Drop (b) AUROC Drop

Fig. 2: Model-wise FI results for DNNs.

All experiments are implemented and performed using PyTorch and executed on an Intel® Core™ i7-9700 CPU. Through these experiments, we aim to thoroughly assess the reliability of LSTM-based DNNs under faults in parameters and evaluate the effectiveness of the proposed protection techniques in preserving model performance.

B. Resilience Analysis Results

As illustrated in Fig. 2, the performance metrics for all models significantly drop under fault injection campaigns. In Fig 2a, as the BER increases, larger models (i.e., Stage-CONV-384 and Stage-384) demonstrate more accuracy drop than the other models, at the same BERs. However, the AUROC drop metric shows a different behavior. It is observed that Stage-CNN-72 and Stage-72 are remarkably sensitive to faults in terms of their AUROC, at the lowest BER. While at higher BERs, the AUROC drop is higher for larger DNNs.

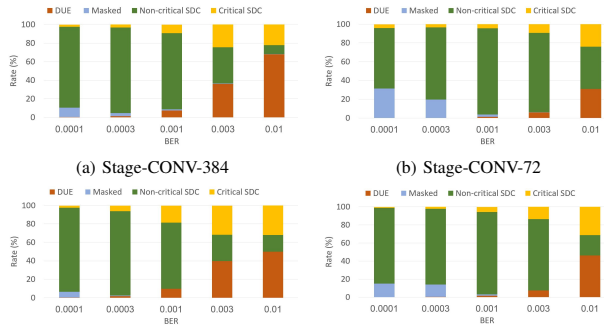
Regarding Fig. 2b, as AUROC expresses the discrimination of classification over different thresholds, this observation shows that smaller DNNs are remarkably sensitive to faults to correctly distinguish the stage of patients' disease. According to the results, the AUROC metric for all DNNs falls below 60% when $BER = 0.01$. At such high BERs, although larger DNNs are shown to be more error-prone, in this safety-critical application, none of them are reliably functioning.

On the other hand, it is observed that DNNs possessing the CONV module are generally more resilient than the ones without the CONV module. It shows that the CONV module increases the capability of fault masking in LSTM-based DNNs and improves their inherent resilience.

According to Fig. 3, in all models, as the BER increases, the fault effects with masked and non-critical SDCs decrease, significantly leading to more critical SDCs and DUEs. This figure also evidences the fact that the DNNs with the CONV module are more resilient to faults than the ones without the CONV module. Obtained results indicate that when $BER = 0.01$, total critical SDC and DUE rate for StageNet-CONV-384, StageNet-CONV-72, StageNet-384 and StageNet-72 is 90.05%, 54.97%, 82.02% and 77.33%.

We perform a bit-level analysis of weights. First, we observe the average value of each bit throughout the weights in all DNNs, as illustrated in Fig. 4a. It is observed that bits 0 to 26 in 32-bit floating point data representation almost have a unified distribution between 0 and 1. while bits number 27, 28, and 29 are always '1' and bit number 30 is always '0'. It shows that we can protect the bits that are constant throughout the weights.

As depicted in Fig. 4b, the accuracy drop for bit 30 in all DNNs is significantly higher than bit-flip in other bits. Consequently, bit 30 is identified as the most critical bit in the DNNs, which is observed in Fig. 4a that its value is always '0'. As a protection mechanism, this bit is always set to '0', which ensures that the most critical bit is consistently safeguarded, enhancing the overall reliability of the LSTM-based DNNs.



(a) Stage-CONV-384 (b) Stage-CONV-72

(c) Stage-384 (d) Stage-72

Fig. 3: Fault classification in model-wise FI for a) Stage-CONV-384, b) Stage-CONV-72, c) Stage-384, d) Stage-72.

C. Fault Tolerance for LSTM-based DNNs

1) Weights Bit Clipping (WBC)

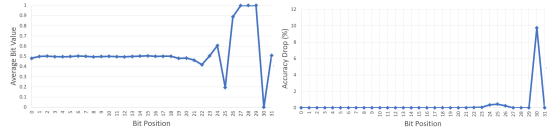
As shown in Fig. 5a and Fig. 5b, weights bit clipping lays a significant improvement on the reliability of LSTM-based DNNs. The accuracy drop is close to zero through all BERs for all protected models. On the other hand, the AUROC drop is remarkably improved for all DNNs. According to the results, the AUROC drop is reduced by up to 3.2x when $BER = 0.01$ and the total critical SDC and DUE rate across DNNs is reduced by up to 278.6x.

2) Activations Value Clipping (AVC)

Fig. 5c and Fig. 5d present how effectively activations value clipping protects the models against faults. According to the results, The accuracy drop and AUROC drop are reduced by up to 15.54x and 1.5x among the DNNs, respectively, when $BER = 0.01$. The fault classification results through the fault injection campaigns on protected DNNs by AVC indicate that this method is also capable of removing all DUE effects. As a result, the total DUE and critical SDC rate for StageNet-CONV-384, StageNet-CONV-72, StageNet-384, and StageNet-72 is 13.88%, 5.18%, 36.47%, and 16.86% resulting in up to 10.6x reduction across DNNs when $BER = 0.01$.

3) Comparison

Fig. 6 compares the proposed zero-memory overhead fault-tolerant techniques for LSTM-based DNNs. As observed, Weights Bit Clipping (WBC) generally demonstrates a more consistent protective effect across different DNNs. It effectively reduces accuracy drop and the incidence of critical faults across the board. However, Activations Value Clipping (AVC) slightly outperforms in a few cases. WBC achieves 2.36x, 1.19x and 2.26x less AUROC drop than AVC in Stage-CONV-384, Stage-CONV-72, and Stage-384, when $BER = 0.01$, whereas AVC provides 1.13x times less AUROC drop than WBC for Stage-72 at the same BER.



(a) Average value of different bits. (b) Finding the most critical bit.

Fig. 4: Monitoring bit values

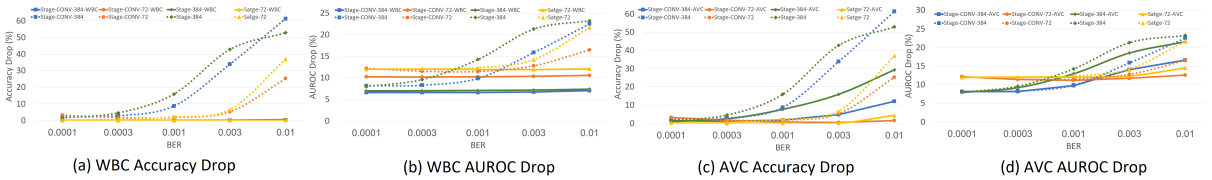


Fig. 5: Weights Bit Clipping protection and Activation Value Clipping and FI results.

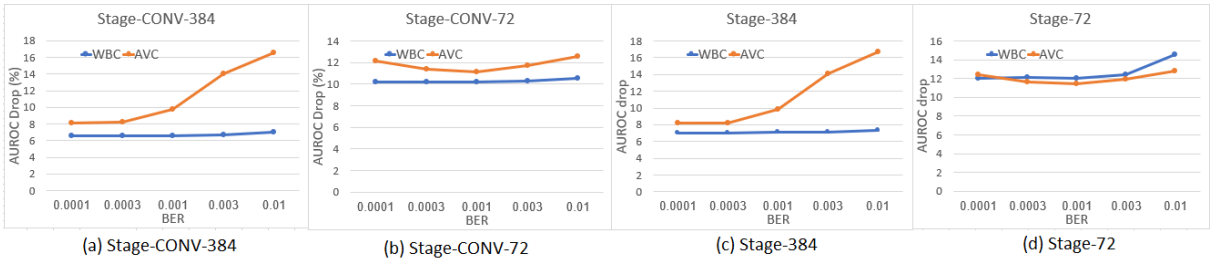


Fig. 6: Weights Bit Clipping (WBC) and Activations Value Clipping (AVC) comparison based on AUROC drop.

Nonetheless, each proposed fault-tolerance technique is applicable in different design scenarios. WBC applies directly to the memory and can be conducted to the bit values of the stored data before an inference. On the other hand, AVC is applied during the inference and prevents errors produced by faulty weights during the inference.

V. Conclusion

In this paper, we study the reliability of various LSTM-based DNNs (variants of StageNet) in healthcare as a case study with different structures and propose two zero-memory overhead fault-tolerance techniques for them. Results indicate that LSTM-based DNNs possessing convolutional layers demonstrate more resilience than the ones without convolutional layers. Furthermore, we performed bit-level analysis resulting in the identification of the most critical bits.

Moreover, two zero-overhead protection techniques to improve their fault tolerance are proposed: weights bit clipping and activations value clipping. It is shown that weights bit clipping can reduce the AUROC drop by up to 3.2x and DUE and critical faults by up to 278.6x compared to the unprotected DNNs at a high BER. Also, activations value clipping reduces the AUROC drop by 1.5x and DUE and critical SDCs by 10.6x, under the same conditions. Thus, the results demonstrate that the weights bit clipping method is extremely effective in mitigating the effect of faults occurring in the parameters of LSTM-based DNNs.

Acknowledgments

This work is supported by PSG837 Estonian national funding.

References

[1] M. Loni, H. Mousavi, M. Riazati, M. Daneshtalab, and M. Sjödin, "Tas: ternarized neural architecture search for resource-constrained edge devices," in *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2022, pp. 1115–1118.

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

[5] 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.

[6] 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.

[7] M. Nourazar, V. Rashtchi, F. Merrikh-Bayat, and A. Azarpeyvand, "Towards memristor-based approximate accelerator: Application to complex-valued fir filter bank," *Analog Integrated Circuits and Signal Processing*, vol. 96, no. 3, pp. 577–588, 2018.

[8] 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.

[9] R. Manne and S. C. Kantheti, "Application of artificial intelligence in healthcare: chances and challenges," *Current Journal of Applied Science and Technology*, vol. 40, no. 6, pp. 78–89, 2021.

[10] E. J. Harris *et al.*, "A survey of human gait-based artificial intelligence applications," *Frontiers in Robotics and AI*, vol. 8, p. 749274, 2022.

[11] B. Lim and S. Zohren, "Time-series forecasting with deep learning: a survey," *Philosophical Transactions of the Royal Society A*, vol. 379, no. 2194, pp. 202–209, 2021.

[12] K. Adam *et al.*, "A selective mitigation technique of soft errors for dnn models used in healthcare applications: Densenet201 case study," *IEEE Access*, vol. 9, pp. 65 803–65 823, 2021.

[13] L.-H. Hoang *et al.*, "Ft-clipact: Resilience analysis of deep neural networks and improving their fault tolerance using clipped activation," in *DATE*, 2020, pp. 1241–1246.

[14] 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.

[15] 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," in *2024 IEEE IOLTS, inpress*, 2024, pp. 1–7.

[16] N. Nosrati and Z. Navabi, "Analysis and enhancement of resilience for lstm accelerators using residue-based ceds," *IEEE Access*, 2024.

[17] M. H. Ahmadilivani, J. Raik, M. Daneshtalab, and A. Kuusik, "Analysis and improvement of resilience for long short-term memory neural networks," in *2023 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*. IEEE, 2023, pp. 1–4.

[18] J. Gao, C. Xiao, Y. Wang, W. Tang, L. M. Glass, and J. Sun, "Stagenet: Stage-aware neural networks for health risk prediction," in *Proceedings of The Web Conference 2020*, 2020, pp. 530–540.

Curriculum Vitae

1. Personal data

Name	Mohammad Hasan Ahmadilivani
Date and place of birth	17 February 1994, Gorgan, Iran

2. Contact information

Address	Tallinn University of Technology, School of Information Technologies, Department of Computer Systems, Akadeemia tee 15a, 12618, Tallinn, Estonia
Phone	+372 5802 8289
E-mail	mohammad.ahmadilivani@taltech.ee

3. Education

2021–2025	Tallinn University of Technology, School of Information Technologies, Dependable Computing Systems, PhD studies
2017–2020	University of Tehran, School of Electrical and Computer Engineering, Computer Systems Architecture, MSc
2012–2017	Iran University of Science and Technology, School of Computer Engineering, Computer Hardware, BSc

4. Language competence

Persian	native
English	fluent
Estonian	beginner

5. Professional employment

2018–2020	Mana Engineering, Embedded Software Developer
2018–2018	Negar Andishegan Pouya, Embedded Software Intern

6. Computer skills

- Operating systems: Windows, Linux
- Document preparation: Latex, Microsoft Word
- Programming languages: Python, C, C++, C#, Matlab
- Hardware Description Languages: VHDL

7. Defended theses

- 2020, Design, Evaluation, and Implementation of Fault Tolerant Techniques for Storage Units in Embedded Processors, MSc, supervisor Prof. Mostafa E. Salehi, University of Tehran.
- 2017, Study and Comparison of Lightweight Encryption Algorithms on ARM Microprocessors, supervisor Prof. Mahdi Fazeli, Iran University of Science and Technology.

8. Field of research

- Efficient and Fault-Tolerant Hardware Design
- Robust Machine Learning and Trustworthy Artificial Intelligence
- Efficient Deep Learning in Embedded Applications and IoT

9. Honors and awards

- 2024, Author of the Research Article of the Year at ICT School of Tallinn University of Technology.

10. Projects

- **DeepVigor Tool**
  - Developed a scalable tool for resilience analysis of DNNs, outputting various necessary metrics, GitHub repo: https://github.com/mhahmadilivany/DeepVigor
- **SentinelNN Tool**
  - Developed a framework for fault resilience assessment and enhancement of CNNs, GitHub repo: https://github.com/mhahmadilivany/SentinelNN
- **AI Chip for a Medical Application**
  - Developed a customized machine learning solution (LSTM) for clinical gaits abnormality detection
  - Optimized the developed LSTM for designing an ASIC AI chip
- **Machine Vision for Collaborative Robots**
  - Technical Lead of two projects supported by AIRE, to develop an AI-based product for automated programming of collaborative robots and production quality control, for industrial manufacturers in Estonia.

Elulookirjeldus

1. Isikuandmed

Nimi	Mohammad Hasan Ahmadilivani
Sünniaeg ja -koht	17 February 1994, Gorgan, Iran

2. Kontaktandmed

Aadress	Tallinna Tehnikaülikool, Infotehnoloogia teaduskond, Akadeemia tee 15a, 12618, Tallinn, Estonia
Telefon	+372 5802 8289
E-post	mohammad.ahmadilivani@taltech.ee

3. Haridus

2021–2025	Tallinna Tehnikaülikool, Infotehnoloogia teaduskond, Usaldusväärsete arvutisüsteemide, doktoriõpe
2017–2020	Teherani ülikool, elektri- ja arvutitehnika teaduskond, Arvutisüsteemide arhitektuur, MSc
2012–2017	Iraani teadus- ja tehnoloogiaülikool, arvutitehnika teaduskond, Arvuti riistvara, BSc

4. Keelteoskus

pärsia keel	emakeel
inglise keel	kõrgtase
eesti keel	algaja

5. Teenistuskäik

2018–2020	Mana Engineering, Embedded Software Developer
2018–2018	Negar Andishegan Pouya, Embedded Software Intern

6. Arvuti oskused

- Operatsioonisüsteemid: Windows, Linux
- Kontoritarkvara: Latex, Microsoft Word
- Programmeerimiskeeled: Python, C, C++, C#, Matlab
- Riistvara kirjelduse keeled: VHDL

7. Kaitstud lõputööd

- 2020, Sardsüsteemide protsessorites salvestusüksuste tõrketaluvate tehnikate hindamine ja rakendamine, MSc, juhendaja Prof. Mostafa E. Salehi, Teherani ülikool.
- 2017, ARM-i mikroprotsessorite kergete krüpteerimisalgoritmide uurimine ja võrdlemine, juhendaja Prof. Mahdi Fazeli, Iraani teadus- ja tehnoloogiaülikool.

8. Teadustöö põhisuunad

- Tõhus ja tõrkekindel riistvara disain

- Robustne masinõpe ja usaldusväärne tehisintellekt
- Tõhus süvaõpe manustatud rakendustes ja asjade Internetis

9. Autasud

- 2024, Infotehnoloogia teaduskonna aasta teadusartikli autor

10. Projektid

- **DeepVigor töörist**
  - Töötas välja skaleeritava tööriista DNN-ide vastupidavuse analüüsiks, väljastades erinevaid vajalikke mõõdikuid, GitHub repo: https://github.com/mhahmadilivany/DeepVigor
- **SentinelNN töörist**
  - Developed a framework for fault resilience assessment and enhancement of CNNs, GitHub repo: https://github.com/mhahmadilivany/SentinelNN
- **AI kiip meditsiiniliseks rakenduseks**
  - Töötas välja kohandatud masinõppelahenduse (LSTM) kliiniliste kõnnakuhäirete tuvastamiseks
  - Optimeeritud väljatöötatud LSTM ASIC AI kiibi kujundamiseks
- **Masinnägemine koostöörobotite jaoks**
  - Kahe AIRE toetatud projekti tehniline juht, mille eesmärk on arendada tehisintellektil põhinevat toodet koostöörobotite automatiseeritud programmeerimiseks ja tootmiskvaliteedi kontrollimiseks, tööstustootjatele Eestis.

ISSN 2585-6901 (PDF)
ISBN 978-9916-80-276-2 (PDF)