TAL TECH

DOCTORAL THESIS

Cross-Layer Reliability Evaluation and Fault Tolerance for Deep Neural Network Hardware Accelerators

Natalia Cherezova

TALLINNA TEHNIKAÜLIKOOL
TALLINN UNIVERSITY OF TECHNOLOGY
TALLINN 2026

TALLINN UNIVERSITY OF TECHNOLOGY
DOCTORAL THESIS
81/2026

Cross-Layer Reliability Evaluation and Fault Tolerance for Deep Neural Network Hardware Accelerators

NATALIA CHEREZOVA

TAL TECH PRESS

TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies
Department of Computer Systems
This dissertation was accepted for the defence of the degree 11/11/2026

Supervisor: Prof. Maksim Jenihhin,
Department of Computer Systems,
School of Information Technologies,
Tallinn University of Technology
Tallinn, Estonia

Co-supervisor: Dr. Artur Jutman,
Department of Computer Systems,
School of Information Technologies,
Tallinn University of Technology
Tallinn, Estonia

Opponents: Prof. Stefano di Carlo,
Politecnico di Torino,
Torino, Italy

Prof. Dimitris Gizopoulos,
University of Athens,
Athens, Greece

Defence of the thesis: 04/12/2026, 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 doctoral or equivalent academic degree.

Natalia Cherezova

---
signature

European Union
European Regional
Development Fund

Investing
in your future

Copyright: Natalia Cherezova, 2026
ISSN 2585-6901 (PDF)
ISBN 978-9916-80-599-2 (PDF)
https://doi.org/10.23658/taltech.81/2026

Cherezova, N. (2026). *Cross-Layer Reliability Evaluation and Fault Tolerance for Deep Neural Network Hardware Accelerators* [TalTech Press]. https://doi.org/10.23658/taltech.81/2026

TALLINNA TEHNIKAÜLIKOOL
DOKTORITÖÖ
81/2026

Süvanärvivõrkude riistvarakiirendite kihtidevaheline töökindluse hindamine ja veakindlus

NATALIA CHEREZOVA

TAL TECH
KIRJASTUS

“Lock up your libraries if you like; but there is no gate, no lock,
no bolt that you can set upon the freedom of my mind.”

– *Virginia Woolf, A Room of One’s Own*

Contents

List of Publications 10

Author's Contributions to the Publications 12

Abbreviations 13

1 Introduction 15
1.1 Motivation 15
1.2 Problem statement 16
1.3 Contributions 18
1.4 Thesis organization 18

2 Background 20
2.1 Deep Neural Networks 20
2.1.1 Convolutional Neural Networks 21
2.1.2 Vision Transformers 22
2.2 Systolic Arrays 23
2.2.1 Systolic array architecture 23
2.2.2 Convolution on systolic array 24
2.3 Approximate Computing 25
2.4 Hardware Reliability 26
2.4.1 Faults, Errors, and Failures 26
2.4.2 Soft errors 27
2.4.3 Reliability evaluation 27

3 Understanding Fault-tolerance Vulnerabilities in Advanced SoC FPGAs for Critical Applications 29
3.1 Introduction 29
3.2 Faults and fault effects in SoC FPGAs 29
3.2.1 Typical structure of modern FPGA-based SoC 29
3.2.2 Faults and fault effects in different components 30
3.3 Vendor-provided fault tolerance mechanisms 34
3.4 Analysis and conclusions about the gaps 37
3.4.1 Vendor-provided fault tolerance methods analysis 37
3.4.2 Gaps in vendor-provided protection 38
3.5 Mapping known user-space methods 39
3.6 Chapter conclusions 41

4 Reliability Assessment of DNNs on Systolic Arrays 42
4.1 SAFPA: Fast Analytical Reliability Assessment for DNN Inference on Systolic Arrays 42
4.1.1 Introduction 42
4.1.2 Methodology 43
4.1.2.1 Fault propagation in systolic arrays 43
4.1.2.2 Fault propagation analysis for transient faults 44
4.1.2.3 Fault propagation analysis for permanent faults 48
4.1.2.4 Implementation 49
4.1.3 Results 49
4.1.3.1 Evaluation 49

4.1.3.2 Fault injection 50
4.2 Reliability-Aware Reconfiguration for Systolic Arrays 50
4.2.1 Introduction 50
4.2.2 Methodology 51
4.2.2.1 RSA Architecture 51
4.2.2.2 Reliability and Performance Analysis for the RSA 51
4.2.2.3 Resizing Strategies 52
4.2.3 Experimental Results 53
4.3 Reliability Assessment of Vision Transformers on Systolic Arrays via Fault Propagation Analysis 55
4.3.1 Introduction 55
4.3.2 Methodology 56
4.3.2.1 Fault Propagation in Systolic Arrays for Matrix Multiplication 56
4.3.2.2 Fault Propagation Analysis 56
4.3.3 Experimental setup 58
4.3.4 Empirical analysis 58
4.3.4.1 Operation-dependent vulnerability pattern 58
4.3.4.2 Significant criticality of the value matrix 59
4.3.4.3 Reliability of embeddings 60
4.3.4.4 Effect of architectural regularity on reliability 60
4.3.4.5 Fault distribution 62
4.4 Chapter conclusions 62

5 FORTALESA: Fault-Tolerant Reconfigurable Systolic Array for DNN Inference 64
5.1 Introduction 64
5.2 Fault-tolerant reconfigurable systolic array 65
5.3 Evaluation 69
5.3.1 FORTALESA hardware implementation parameters 69
5.3.2 Reliability analysis 70
5.3.3 Performance vs. reliability trade-off 72
5.3.4 Comparison with static redundancy 73
5.3.5 Comparison with the state-of-the-art 75
5.4 FORTALESA-based ASIC design 75
5.5 Chapter conclusions 76

6 Approximation-aided Reliability of DNN Accelerators 77
6.1 Introduction 77
6.2 AdAM Architecture 78
6.2.1 Definitions 78
6.2.2 Hardware implementation 80
6.2.2.1 Adaptive adder 81
6.2.2.2 LOD design 82
6.3 Experimental Results 82
6.3.1 Experimental Setup 82
6.3.2 Hardware utilization 83
6.3.3 DNN accuracy 84
6.3.4 Reliability analysis 86
6.4 Chapter conclusions 89

7 Conclusions 90
List of Figures 92
List of Tables 94
References 95
Acknowledgements 113
Abstract 114
Kokkuvõte 116
Appendix 1 119
Appendix 2 131
Appendix 3 137
Appendix 4 149
Appendix 5 161
Appendix 6 171
Appendix 7 183
Curriculum Vitae 197
Elulookirjeldus 199

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 N. Cherezova, K. Shibin, M. Jenihhin, and A. Jutman. Understanding fault-tolerance vulnerabilities in advanced SoC FPGAs for critical applications. *Microelectronics Reliability*, 146:115010, 2023
DOI: https://doi.org/10.1016/J.MICROREL.2023.115010

II M. Taheri, N. Cherezova, S. Nazari, A. Rafiq, A. Azarpeyvand, T. Ghasempouri, M. Daneshtalab, J. Raik, and M. Jenihhin. AdAM: adaptive fault-tolerant approximate multiplier for edge DNN accelerators. In *2024 IEEE European Test Symposium (ETS)*, 2024
DOI: https://doi.org/10.1109/ETS61313.2024.10567161

III M. Taheri, N. Cherezova, S. Nazari, A. Azarpeyvand, T. Ghasempouri, M. Daneshtalab, J. Raik, and M. Jenihhin. AdAM: Adaptive approximate multiplier for fault tolerance in DNN accelerators. *IEEE Transactions on Device and Materials Reliability*, 25(1):66–75, 2025
DOI: https://doi.org/10.1109/TDMR.2024.3523386

IV N. Cherezova, S. Pappalardo, A. Ruospo, B. Deveautour, L. Fezza, A. Jutman, E. Sanchez, A. Bosio, M. S. Reorda, and M. Jenihhin. Special session: Reliability assessment of DNN models and inference on systolic arrays. In *IEEE VLSI Test Symposium (VTS)*, 2026
DOI: https://doi.org/10.1109/VTS69484.2026.11563366

V N. Cherezova, M. Jenihhin, and A. Jutman. Scalable reliability assessment of vision transformers on systolic arrays via fault propagation analysis. In *IEEE 32nd International Symposium on On-Line Testing and Robust System Design (IOLTS)*, pages 1–7, 2026
DOI: https://doi.org/10.1109/IOLTS69666.2026.11633827

VI M. H. Ahmadilivani, N. Cherezova, M. Glaß, J.-D. Guerrero-Balaguera, M. Jenihhin, A. Kritikakou, R. L. Sierra, L. B. Poelhs, J. Raik, L. Roquet, F. F. dos Santos, F. A. da Silva, M. S. Reorda, A. Veronesi, E. C. V. Castillo, and J. E. R. Condia. Reliability assessment of deep neural networks and accelerators across design stages. In *IEEE 27th Latin American Test Symposium (LATS)*, pages 1–10, 2026
DOI: https://doi.org/10.1109/LATS70329.2026.11480274

VII N. Cherezova, A. Jutman, and M. Jenihhin. FORTALESA: Fault-tolerant reconfigurable systolic array for DNN inference. *Microprocessors and Microsystems*, 119:105222, 12 2025
DOI: https://doi.org/10.1016/J.MICPRO.2025.105222

Other related publications

VIII M. Taheri, N. Cherezova, M. S. Ansari, M. Jenihhin, A. Mahani, M. Daneshtalab, and J. Raik. Exploration of activation fault reliability in quantized systolic array-based DNN accelerators. In *2024 25th International Symposium on Quality Electronic Design (ISQED)*, pages 1–8, 2024

IX N. Cherezova, M. Jenihhin, and A. Jutman. IJTAG-compatible symptom-based SEU monitors for FPGA DNN accelerators. In *2024 IEEE International Conference on Design, Test and Technology of Integrated Systems (DTTIS)*, pages 1–6, 2024

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 *2024 IFIP/IEEE 32nd International Conference on Very Large Scale Integration (VLSI-SoC)*, pages 1–4, 2024

XI S. Devadze, C. E. Nielsen, N. Cherezova, D. Mihhailov, and P. Ellervee. Architectural exploration and implementation of CERN LHC trigger algorithm with FPGA. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 33(11):2998–3007, 2025

XII M. Taheri, P. Patne, N. Cherezova, A. Mahani, C. Herglotz, and M. Jenihhin. RL-agent-based early-exit DNN architecture search framework. In *2025 IEEE 28th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS)*, pages 145–148, 2025

XIII R. M. Kodamanchili, N. Cherezova, M. Taheri, and M. Jenihhin. Adaptive fault resilience for early-exit DNNs. In *2025 IEEE International Test Conference in Asia (ITC-Asia)*, pages 108–113, 2025

XIV M. Jenihhin, M. Taheri, and N. Cherezova. Quantization-aided cost-efficient reliability of CNN accelerators for edge AI. In S. Liu, P. Reviriego, Z. Gao, and F. Lombardi, editors, *Machine Learning Systems: The Role of Hardware Design for Dependable Computing*. Springer Nature Switzerland, 2026

Author's Contributions to the Publications

I In I, I was the main author, conducted a literature review, developed a taxonomy of the advanced FPGA SoC platforms, analyzed faults and fault effects in different components, and reviewed available reliability solutions. I prepared figures and wrote the manuscript.

II In II, I was a co-author, contributed to conceptualizing the idea, developed a hardware implementation of the multiplier, carried out the simulations, prepared figures, and wrote the manuscript.

III In III, I was the second main author, conceptualized the idea, formalized the approach, developed a hardware implementation of the multiplier, carried out the simulations, prepared figures, and wrote the manuscript.

IV In IV, I was the main author of the TalTech part of the special session, developed and implemented the methodology, conducted experiments, and simulations. I analyzed the results, prepared the figures, and wrote the manuscript.

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

VI In VI, I was a co-author of the TalTech part of the special session, contributed to conceptualizing the idea, developed a hardware implementation of the reconfigurable systolic array, performed simulations, and analyzed the results. I prepared figures and wrote the manuscript.

VII In VII, I was the main author, developed and implemented the methodology for reliability assessment and architecture of the reconfigurable systolic array, developed hardware implementation, carried out synthesis, conducted experiments, and simulations. I analyzed the results, prepared the figures, and wrote the manuscript.

Abbreviations

ABFT	Algorithm-Based Fault Tolerance
AI	Artificial Intelligence
AC	Application Crash
ALU	Arithmetical Logical Unit
AMBA	Advanced Microcontroller Bus Architecture
AHB	Advanced High-performance Bus
ASDC	Application Silent Data Corruption
ASIC	Application-Specific Integrated Circuit
AVF	Architecture Vulnerability Factor
BIST	Built-In Self Test
CNN	Convolutional Neural Network
CM	Configuration Memory
COTS	Commercial Off-The-Shelf
CPU	Central Processing Unit
CRAM	Configuration RAM
CRC	Cyclic Redundancy Check
DMR	Dual-Modular Redundancy
DNN	Deep Neural Network
DRG	Dual Redundancy Grouping
DSE	Design Space Exploration
DSP	Digital Signal Processing
DUE	Detected Unrecoverable Error
ECC	Error Correction Code
EDA	Electronic Design Automation
FC	Fully Connected
FPGA	Field-Programmable Gate Array
FPU	Floating-Point Unit
FSM	Finite State Machine
FT	Fault Tolerance
GPU	Graphical Processing Unit
HDL	Hardware Description Language
HMPSoC	Heterogeneous Multi-Processor System-on-Chip
HW	Hardware
IOB	Input/Output Block
IS	Input-Stationary
JTAG	Joint Test Action Group
LFSR	Linear-Feedback Shift Register
LOD	Leading One Detector
LSB	Least Significant Bit
LUT	Look-Up Table
MAC	Multiply-Accumulate
MARE	Mean Absolute Relative Error
MEU	Multiple Event Upset
MHA	Multi-Head Attention
ML	Machine Learning
MLP	Multi-Layer Perceptron
MSB	Most Significant Bit

MUX	Multiplexor
NN	Neural Network
NoC	Network-on-Chip
OCM	On-Chip Memory
OS	Output-Stationary
PDK	Process Design Kit
PDP	Power-Delay Product
PE	Processing Element
PIP	Programmable Interconnect Point
PS	Processing System
RAM	Random Access Memory
RSA	Reconfigurable Systolic Array
RTL	Register Transfer Level
SDC	Silent Data Corruption
SEE	Single Event Effect
SEFI	Single Event Functional Interrupt
SET	Single Event Transient
SEU	Single Event Upset
SoC	System-on-Chip
SW	Software
TAP	Test Access Port
TMR	Triple Modular Redundancy
TPU	Tensor Processing Unit
TRG	Triple Redundancy Grouping
ViT	Vision Transformer
WS	Weight-Stationary

1 Introduction

1.1 Motivation

Recent years have been marked by a tremendous increase in the adoption of Machine Learning (ML) algorithms, specifically Deep Neural Networks (DNNs), and their deployment in mission- and safety-critical application domains, such as automotive [120], avionics [218], and space [202]. However, the size and computational complexity of DNN models have increased drastically since their introduction. Recently, Transformer models, whose complexity surpasses that of traditional neural networks, have gained significant popularity across a range of tasks, from natural language processing to computer vision. The computational complexity of DNN models necessitates specialized hardware accelerators. However, as nanoelectronics used in hardware accelerators are prone to faults caused by radiation and aging, concerns about their reliability come to the fore [6, 148]. It is of utmost importance to understand how faults in hardware components propagate to the final application. Moreover, accelerators are shown to be more prone to transient faults than other electronic devices because the parallelism they contain tends to propagate a single fault to multiple outputs [80].

With continuous technology scaling and increased silicon density, these accelerators become even more susceptible to faults. With the high integration density of modern nanoelectronic devices, the system error rate is increasing rapidly, as it is proportional to the number of transistors [149].

Furthermore, the current trend is driving research and industry to deploy DNNs in embedded systems to be used at the edge, where power and computational resources are limited. Therefore, deploying a DNN accelerator requires addressing the trade-off between different design parameters and reliability. Although DNNs possess certain intrinsic fault-tolerant and error-resilient characteristics, it is insufficient to conclude the reliability of DNNs without considering the characteristics of the corresponding hardware accelerator.

Especially since nowadays there is a clear industrial trend toward building mission- and safety-critical systems around commercial off-the-shelf (COTS) devices [23]. For instance, in the space segment, this trend is further escalated by the need for in-orbit high-performance computing [109], since rad-hard components are built using outdated technology and cannot provide high computational power compared to the COTS devices. However, vendor-provided fault-tolerance solutions for COTS devices are not intended for harsh environments and safety-critical applications.

The most commonly used platforms for implementing hardware accelerators are FPGAs (Field Programmable Gate Arrays) and custom ASICs (Application-Specific Integrated Circuits). FPGAs are an example of reconfigurable hardware that offer several advantages, such as flexibility, reusability, and moderate development cost. FPGAs are a common part of heterogeneous systems-on-chip (SoCs) that integrate multiple components on a single chip, providing both software and hardware programmability. Multiple toolchains for mapping DNNs to FPGAs have been proposed by the industry and academic community [1, 68, 127, 201]. Examples include the FINN framework [192] by AMD-Xilinx, fpgaConvNet [200], and HLS4ML [166]. ASICs, on the other hand, are custom-built and optimized for a specific task. They offer higher performance and power efficiency than FPGAs and provide more control over system design and reliability at the expense of higher design and manufacturing costs.

Hardware faults can be caused by various reasons: radiation, aging, manufacturing process variations, manufacturing defects, and early-life failures. Nonetheless, radiation-induced soft errors are considered dominant for today's digital systems [13, 80, 149, 199].

The major radiation sources are cosmic rays and radioactive impurities. Cosmic radiation is not only a problem for satellites and spaceships, but it also affects avionic and terrestrial electronics as well. Data centers with large numbers of computers are especially susceptible to radiation-induced faults [172]. Big companies like Google, Meta, and Alibaba call silent data corruption (SDC) faults in their data centers a significant problem and call for action [48, 75, 203]. SDCs can have serious consequences, e.g., bad metadata can result in the loss of an entire file system, a corrupted encryption key can make large amounts of data permanently inaccessible [75], which are greatly amplified by the massive scale of modern data centers and AI workloads.

Moreover, it is important to have an efficient solution that fits the profile of the developed system. It is vital to have a library of solutions for different budgets and requirements. There arises the need for cost-efficient reliability. The knowledge from different hardware and software layers should be considered together to achieve cost-efficient reliability in constrained scenarios [149].

Implementing fault-tolerance solutions on individual system layers without considering the whole picture might result in an overprotected system or even a degradation of the overall reliability. A cross-layer approach to reliability enables avoidance of over-protection by considering fault propagation and manifestation mechanisms. It has been gaining popularity in recent years since it can help to develop a globally optimized fault management system [32, 97, 160, 164, 197].

Furthermore, with the current need for self-aware computing systems that can monitor their state and adapt to changing environments and operating conditions, an important property of hardware design is the ability to adjust or reconfigure in response to changing conditions to improve availability.

1.2 Problem statement

Considering the points presented above, the objective of the research is to develop new methods for reliability assessment and enhancement of modern hardware platforms, specifically DNN accelerators. The fundamental challenges are the complexity of modern hardware platforms and DNN models. The following problems and research questions were identified:

1. Understanding fault tolerance vulnerability in heterogeneous systems-on-chip

The common trend these days is to use advanced computing platforms, such as heterogeneous systems-on-chip (SoCs), for intelligent autonomous systems. Heterogeneous SoCs provide software programmability through scalar and vector processors and hardware programmability through FPGA logic, which allows the implementation of ML accelerators tightly connected to the processor. However, as modern heterogeneous SoC FPGAs become more complex and advanced, integrating more components on a single chip makes them increasingly susceptible to various hardware faults. Additionally, their reliability assessment presents certain challenges: simulation and testing are difficult due to the design's complexity. Efficient deployment of heterogeneous computing platforms requires a sufficient understanding of their sensitivity to possible faults. Considering this problem, the following research questions were formulated:

1. What are the most vulnerable components in the heterogeneous systems-on-chip, such as SoC FPGA?

2. How do faults in different SoC components manifest at the module and system levels?

3. What are the gaps in the fault tolerance of the heterogeneous system-on-chips available off-the-shelf?

2. Scalable reliability evaluation of systolic-array-based DNN HW accelerators

Reliability evaluation is a prerequisite for enhancing reliability since it shows how faults affect the system and identifies the most vulnerable components. This way, it is the principal part of a reliable design process. Both reliability assessment and enhancement of DNNs can be performed at different levels of abstraction: at the model level, without considering the hardware architecture on which it will be executed, or at the hardware level. Typically, model-level assessment methods are fast; however, because they do not account for hardware architecture, they tend to be less accurate. Hardware-level methods evaluate fault effects with higher accuracy but suffer from longer simulation times. The following research questions concerning this problem were formulated:

1. How to perform a scalable and fast reliability analysis of a systolic-array-based DNN HW accelerator?

2. How vulnerable are modern DNN model architectures to faults in systolic-array-based HW accelerators?

3. How to evaluate the propagation of hardware faults through the system levels to the DNN output?

3. Efficient fault tolerance techniques for systolic-array-based DNN HW accelerators

Traditional fault-tolerant techniques rely on redundancy, incurring overhead. The trade-off between fault tolerance and performance in DNN accelerators remains underexplored, leading to inefficient system designs. An efficient reliability solution for modern accelerator platforms should be tailored to the mission profile and operating conditions. It should account for the complexity of the hardware platforms used and the application. The approach to reliability should be hierarchical. Fault-tolerance solutions at higher system layers, such as user software and applications, should consider the characteristics of the underlying hardware, corresponding fault models, and fault effects. At the same time, solutions at the lower architecture and microarchitecture levels should be adaptable to application requirements and characteristics.

Considering this problem, the following research questions were formulated:

1. How to effectively enhance the fault tolerance of systolic-array-based DNN HW accelerators with minimum overhead?

2. How to effectively trade off reliability and performance in systolic-array-based DNN HW accelerators?

3. How to provide a systematic and hierarchical fault tolerance for systolic-array-based DNN HW accelerators?

1.3 Contributions

To address the identified problems and research questions, the following contributions are made:

- **A survey of fault-tolerance vulnerabilities in modern heterogeneous systems for critical applications.** The survey presents an analysis of the most common types of errors caused by faults in different components and identifies the most critical and vulnerable ones. It examines vendor-provided and user-space reliability solutions, highlights existing fault-tolerance gaps in modern heterogeneous systems, and outlines a vision for future solutions. The contribution is based on publication [I].

- **Fast analytical reliability assessment method for DNN inference on systolic arrays.** An analytical fault-propagation-based method, SAFPA, enables fast evaluation and supports multiple systolic array dataflows, and provides several orders of magnitude speedup over microarchitecture-level fault injection. The method supports assessment for two modern DNN architectures for computer vision tasks, namely Convolutional Neural Networks (CNNs) and Vision Transformers (ViT). Moreover, the method enables a design space exploration of reliability-performance trade-offs in CNNs deployed on systolic arrays with reconfigurable size. The contribution is based on publications [VII], [IV], [V], and [VI].

- **Fault-Tolerant reconfigurable systolic array for DNN inference.** A run-time reconfigurable fault-tolerant systolic array architecture, FORTALESA, with three execution modes and four implementation options, is proposed and evaluated. FORTALESA is used to enhance the reliability of DNN inference on a systolic array by heterogeneous mapping of different network layers to different execution modes. The proposed architecture efficiently protects both the registers and the arithmetic units of systolic array processing elements against transient and permanent faults without interrupting inference execution. The contribution is based on publication [VII].

- **Approximation-aided reliability of DNN accelerators.** An adaptive fault-tolerant approximate multiplier tailored for DNN accelerators, called AdAM, is proposed. The multiplier relies on an unconventional use of the leading-one position value of the inputs for fault detection and correction by optimizing unused resources. This multiplier enables a negative overhead fault tolerance compared to the exact multipliers. The contribution is based on publications [II], and [III].

1.4 Thesis organization

This thesis consists of seven chapters. The rest of it is organized as follows:

- **Chapter 2** gives a succinct background for the thesis, introducing the idea behind Deep Neural Networks (DNNs) and modern DNN architectures for computer vision tasks, such as Convolutional Neural Networks (CNNs) and Vision Transformers (ViT); structure and functionality of DNN accelerators based on systolic array; basics of approximate computing; and the concepts of hardware reliability.

- **Chapter 3** presents the survey of fault-tolerance vulnerabilities in modern heterogeneous systems, such as SoC FPGA. It outlines the typical structure of the modern SoC FPGA and provides a discussion on the most common types of errors caused by faults in different components. The most critical and vulnerable components

are identified there as well. It gives an overview of reliability solutions provided by vendors, as well as user-space solutions, and highlights fault-tolerance gaps.

- **Chapter 4** presents an analytical fault-propagation-based method, SAFPA, for reliability assessment of DNN inference on systolic arrays. Experimental results on representative convolutional neural networks demonstrate that the method provides a several-orders-of-magnitude speedup over microarchitecture-level fault injection and highlights differences in dataflow-dependent vulnerabilities. The method is also used for the fine-grained reliability analysis of emerging Vision Transformers, highlighting their differences with conventional convolutional neural networks. Additionally, with a comprehensive reliability evaluation enabled by SAFPA, a layer-wise reconfiguration strategy for CNN inference on a systolic array is proposed to balance reliability and performance. It allows identifying the optimal systolic array size for each CNN layer.

- **Chapter 5** presents the architecture of the FORTALESA systolic array with three execution modes (FORTALESA) for run-time reconfigurable redundancy and flexible reliability vs. performance trade-off, its hardware implementation, and evaluation. The proposed systolic array architecture is a part of a flexible DNN protection methodology that balances reliability and performance. The presented results evaluate the effectiveness of the proposed methodology for DNN reliability. The chapter concludes with an overview of an open-source ASIC (Application-Specific Integrated Circuit) designed using the FORTALESA systolic array.

- **Chapter 6** presents a solution for approximation-aided reliability: an adaptive fault-tolerant approximate multiplier (AdAM) tailored for DNN accelerators. The multiplier is based on Mitchell's logarithmic multiplier that substitutes multiplication with the addition of approximated logarithms of the operands. The proposed multiplier protects higher-order bits of the product based on the leading-one bit position in the multiplier inputs.

- **Chapter 7** presents thesis conclusions and outlines future work.

2 Background

2.1 Deep Neural Networks

A Neural Network (NN) is a Machine Learning (ML) algorithm used for a variety of tasks, including classification and regression. It is a mathematical model inspired by the human brain. Neural networks appeared already in the 1940s [122]; however, after their early success, they were forgotten for several decades due to limitations of those early models and issues with their training. Interest resurfaced in the early 2000s due to increased computational power and the adoption of GPUs for model training [65]. A neural network is built using neurons organized into layers. The mathematical model of a single neuron is called a perceptron [154]. The model of a single neuron is

$$y = f\Big(\sum_{i=1}^{d} w_i x_i + b\Big), \tag{1}$$

where d is the dimension of the input x and the number of weights w, $f(\cdot)$ is an activation function, and b is a bias. Originally, the most popular activation functions were the logistic sigmoid and hyperbolic tangent. These days, the most popular activation function is ReLU (Rectified Linear Unit) $f(x) = \max\{0, x\}$ and its variants.

NNs are trained using a designated dataset. The training process is iterative and aims to update the model parameters (weights and biases). A loss function is employed to measure the difference between the expected and the predicted outputs of the NN. During training, parameters are iteratively updated to minimize the loss function. During deployment, inference is run on new data to predict the output.

NNs consist of multiple layers, namely an input layer, an output layer, and one or more hidden layers in between. The simplest neural network architecture is a Multi-Layer Perceptron (MLP). MLP consists of several fully connected layers, where every neuron in the hidden layer is connected to every neuron in the previous and next layers. An example of MLP is presented in Figure 1.

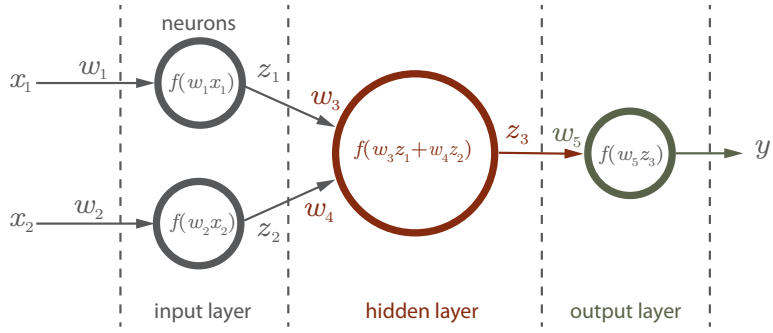


Figure 1: Multi-layer perceptron

Recently, the terms Deep Learning and Deep Neural Networks (DNNs) have been adopted to denote the increased depth and complexity of ML problems and algorithms. Modern DNN architectures include a variety of layers designed to increase accuracy, improve training, and reduce computational complexity. Two notable architectures for computer vision tasks that are considered in the thesis, i.e., Convolutional Neural Networks (CNNs) and Vision Transformers (ViT), are described below.

2.1.1 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a type of DNNs that specialize in image recognition and computer vision tasks. CNNs begin with convolution and pooling operations, breaking down the image into features and analyzing them independently. The result is fed into a sequence of fully-connected layers for a final decision.

Convolution is a linear operation that involves a multiplication of a set of weights with the 2D or 3D input. Convolutional weights are in the form of a 2D or 3D array, called a filter or a kernel. The multiplication is the dot product, which is then summed together, resulting in a single output value. The filter is applied systematically to each overlapping filter-sized window of the input, moving left to right and top to bottom in a sliding-window manner. If the filter is designed to detect a specific type of feature, then applying that filter systematically across the entire input allows discovering that feature anywhere in the image. This way, convolution layer inputs are also called input feature maps, and outputs are called output feature maps.

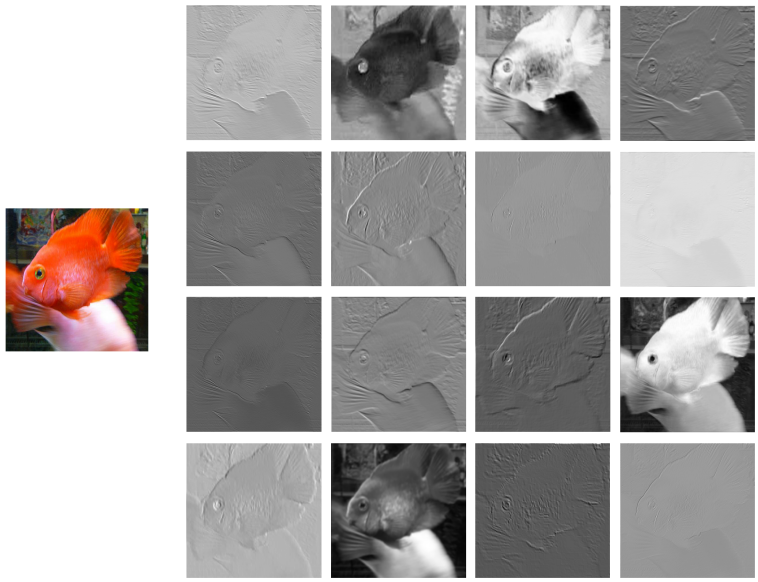


Figure 2: CNN feature visualization (output of the first layer of ResNet-50, the first 16 channels)

The pooling operation reduces the size of the feature map by replacing a rectangular neighborhood with a statistical summary, e.g., the maximum or average value.

CNNs rely on hierarchical feature extraction as convolutional layers use small kernels to capture local features, and the information from various regions of the input is integrated hierarchically through the layers [16].

Notable CNN architectures include LeNet-5 [103], developed for handwritten numbers recognition, it was also the first CNN; AlexNet [99], the first CNN that surpassed the human in ImageNet Large Scale Visual Recognition Challenge (ILSVRC-2012) contest; VGG [171], that showed the effect of increased depth of the network on the accuracy; ResNet [73], that introduced residual connections between layers with increased network depth to improve optimization and accuracy; GoogLeNet [175], that introduced Inception modules to reduce computational complexity; MobileNet [78], substituted traditional convolution with depth-wise separable convolution for light-weight inference. These models are widely used in the academic community as benchmarks. Common dataset bench-

marks include the MNIST dataset of handwritten digits [104], the CIFAR-10 and CIFAR-100 datasets of small color images [98], and ImageNet, a large-scale dataset of high-resolution color images [157].

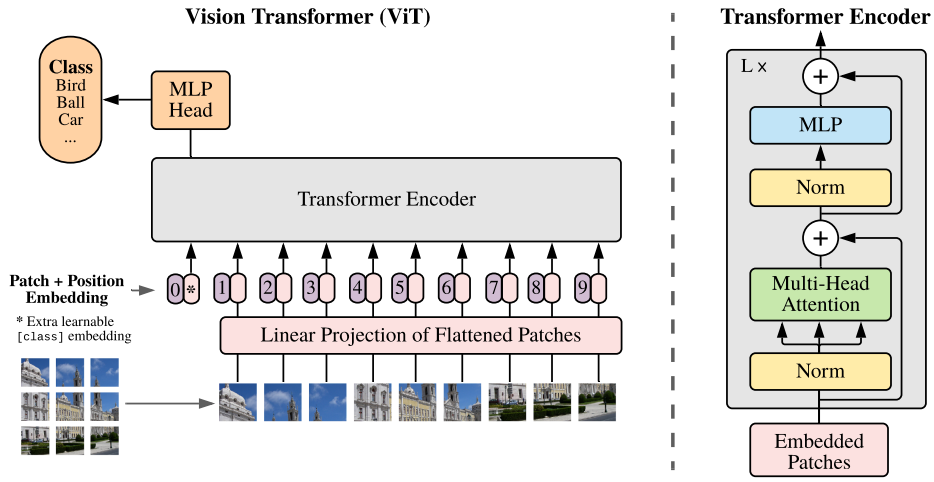


Figure 3: Architecture of a Vision Transformer [50]

2.1.2 Vision Transformers

Vision Transformers were proposed in 2021 by Dosovitskiy et al. [50]. The architecture of basic ViT is presented in Figure 3. First, an input image is transformed into a sequence of 2D patches. The patches are flattened and mapped to D dimensions through a trainable linear projection. A classifier embedding is added to the resulting patch embeddings. The embeddings are then passed through a sequence of Transformer Encoder blocks. Each encoder block includes Multi-Head Attention (MHA) and a Multi-Layer Perceptron (MLP), consisting of two fully connected (FC) layers. Transformers were originally designed for natural language processing tasks [198]. They utilize a self-attention mechanism that enables the model to effectively capture relationships between different components of the input sequence, especially long-term dependencies.

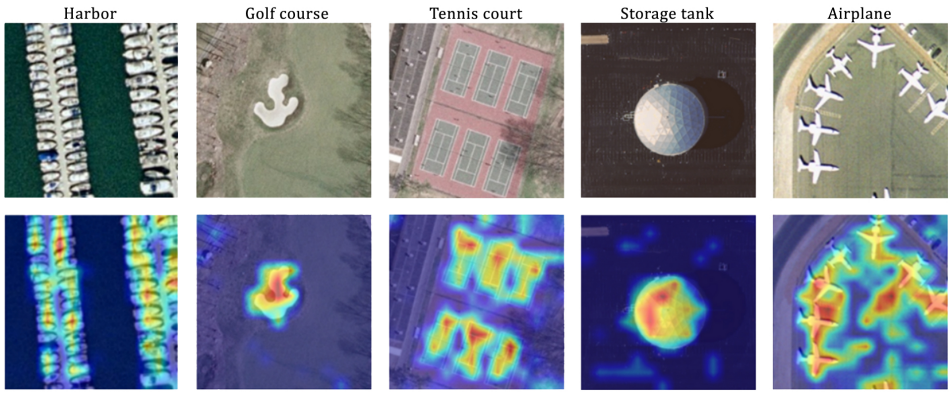


Figure 4: Attention map visualization (adapted from [67])

Unlike convolutional layers in CNNs, which extract local features, the attention mechanism in ViTs directly captures global features and relationships among different regions of the input [16]. In other words, the MHA block defines where to put focus in the image and draws connections between patches. Visualization of the attention map is given in Figure 4. It starts with the calculation of **query** (Q), **key** (K) and **value** (V) vectors for each patch:

$$Q = XW_Q, K = XW_K, V = XW_V, \tag{2}$$

where X is the embedding matrix, W_Q, W_K, W_V are corresponding weights. The query vector represents what is important in the other patches for the particular patch. The key vector represents the features of each patch that can be "matched" against queries. The value vector contains the actual information (features) of the patch.

Then, the query and transposed key matrices are multiplied to get attention weight or **attention score** (S_{attn}) that defines the relevance between the patches:

$$S_{attn} = QK^T. \tag{3}$$

Attention value or a **context** (Z) is then calculated by multiplying value V by the scaled attention score:

$$S_{sim} = \text{softmax}\left(\frac{S_{attn}}{\sqrt{d_k}}\right) \tag{4}$$

$$Z = S_{sim}V, \tag{5}$$

where S_{sim} is a similarity score, and d_k is the dimension of the Q, K and V vectors.

The idea behind MHA is that each head learns different relationships and provides a unique perspective on the input image. The attentions from all heads are concatenated and passed through a linear layer to get the **MHA output**:

$$O = ZW_Z. \tag{6}$$

The MHA output (O) is then passed to the MLP block, as mentioned earlier, consisting of two FC layers. Unlike CNNs, where FC layers are represented by vector-matrix multiplication, in ViTs, FC layers are represented by matrix-matrix multiplication.

The output of the final encoder layer corresponding to the added classifier embedding is passed to the MLP Head for classification. The FC layer of the head is a vector-matrix multiplication.

Specific architectures of Vision Transformers are defined by the following parameters: patch size, embedding size, number of encoder layers, number of heads in MHA, and MLP dimension.

2.2 Systolic Arrays

Systolic arrays were first introduced in the 1980s [101] and recently became popular for accelerating DNNs. Many DNN accelerators are based on systolic array architecture, including Google TPU [93], MIT Eyeriss [30], and Berkeley Gemmini [62].

2.2.1 Systolic array architecture

Systolic array has a regular structure consisting of interconnected Processing Elements (PEs). In this thesis, systolic arrays used to accelerate matrix multiplication for DNN applications are considered, with PEs organized in a 2D grid. Each PE consists of a Multiply-Accumulate (MAC) unit and registers for holding intermediate values. Data is fed from the

top and left sides of the systolic array and propagates through rows and columns during the computation. The name "systolic" was selected because the data movement in the array resembles the function of the heart, as each PE constantly "pumps" data inside and outside [12]. Each clock cycle, PE in row i and column j performs calculations with values coming from PE on the left (row i, column $j-1$) and PE above (row $i-1$, column j); the data is then passed to the PE on the right (row i, column $j+1$) and PE below (row $i+1$, column j). This way, systolic arrays exploit both parallelism and pipelining in the matrix multiplication algorithm, enabling high throughput and avoiding the memory-access bottleneck. Commonly, memory buffers for activations and weights are located at the top and left sides of the systolic array, respectively. In the case of DNNs, weights are fed vertically from top to bottom; thus, each column is assigned to one output channel. Activations (inputs) are fed from left to right; hence, each row processes values from the same sliding window. There are different types of systolic arrays based on the dataflow. Dataflow type defines the internal structure of PEs and the movement of data through the array, i.e., which values are stationary and which values are propagating through the array during calculation. Dataflow types include weight-stationary (WS), input-stationary (IS), and output-stationary (OS) [29]. The architecture of the systolic array and an individual PE for output-stationary and weight-stationary dataflows are presented in Figure 5.

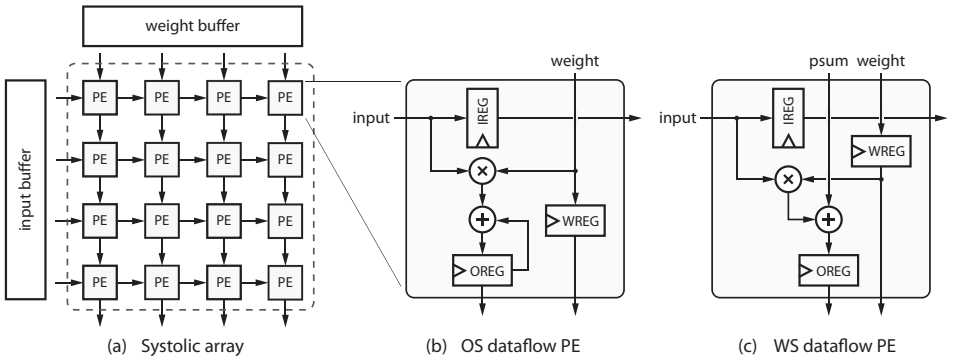


Figure 5: Systolic array and individual PE architecture

In the **weight-stationary** (WS) dataflow, the weight matrix is first pre-loaded into a systolic array by loading one row per cycle, then the input matrix propagates through the array from left to right. The outputs (results of the MAC operation) propagate from top to bottom. In this scenario, weights are stationary and remain in the systolic array during matrix multiplication. This dataflow maximizes weight reuse.

In the **input-stationary** (IS) dataflow, activations are pre-loaded into the systolic array similarly to WS and kept in the systolic array during the matrix multiplication process. The data movement and structure of the systolic array and individual PEs are similar to WS dataflow.

In the **output-stationary** (OS) dataflow, outputs are kept in the PEs of the systolic array, while weights and activations are streaming through the systolic array. Weights are streaming from top to bottom, and activations (inputs) are streaming from left to right. Outputs are accumulated inside PEs. This dataflow type does not need a pre-loading stage.

2.2.2 Convolution on systolic array

The convolution operation is mapped to the systolic array by transforming it to matrix multiplication using the im2col (image-to-column) algorithm [25]. An input tensor is trans-

formed into a $(H_{out} \cdot W_{out}) \times (H_k \cdot W_k \cdot C_{in})$ matrix and weights are transformed into a $(H_k \cdot W_k \cdot C_{in}) \times C_{out}$ matrix, where H_{out} and W_{out} are the height and width of the output tensor, H_k and W_k are the height and width of a kernel, C_{in} and C_{out} are the number of channels in the input and output tensor correspondingly. When inputs and weights are represented this way, each column of the systolic array calculates values for one output channel, and each row calculates values of the same pixel across all output channels.

If the input matrices exceed the size of the systolic array, matrix multiplication is performed in tiles corresponding to the size of the systolic array grid.

The latency L of a matrix multiplication on a systolic array is calculated as

$$L_{SA} = M + N - 1 + N - 1 = M + 2N - 2\,, \tag{7}$$

where $N \times N$ is the size of the systolic array and $N \times M$ is the size of the matrices [161].

For the OS systolic array, the number of tiles is

$$T_a^{os} = \left\lceil \frac{H_{out} \cdot W_{out}}{N} \right\rceil, \quad T_w^{os} = \left\lceil \frac{C_{out}}{N} \right\rceil, \tag{8}$$

where T_a and T_w are the number of tiles in activations and weights. For the WS systolic array, the number of tiles is

$$T_a^{ws} = \left\lceil \frac{H_k \cdot W_k \cdot C_{in}}{N} \right\rceil, \quad T_w^{ws} = \left\lceil \frac{C_{out}}{N} \right\rceil \cdot T_a^{ws} = T_w^C \cdot T_a^{ws}. \tag{9}$$

For the IS systolic array, the number of tiles is

$$T_a^{is} = \left\lceil \frac{H_{out} \cdot W_{out}}{N} \right\rceil \cdot T_w^{is} = T_a^S \cdot T_w^{is}, \quad T_w^{is} = \left\lceil \frac{H_k \cdot W_k \cdot C_{in}}{N} \right\rceil. \tag{10}$$

2.3 Approximate Computing

As energy consumption and complexity in modern computing systems grow rapidly, one promising energy-efficient design technique is approximate computing [26, 91]. Many operations do not require a perfect result; an approximation with tolerable error would suffice. For example, to compare 2 and $\sqrt{3}$, there is no need to compute an accurate square root of 3; it is enough to know that the answer is between 1 and 2. The basic building blocks for approximate computing are approximate arithmetic circuits, such as adders, multipliers, and dividers, as well as larger modules like the fast Fourier transform and the multiply-accumulate (MAC) unit [26]. Commonly, approximate arithmetic circuits are derived from accurate ones by modifying, removing, or adding some components [91].

Multipliers are one of the primary arithmetic building blocks in DNN accelerators. Since DNNs have intrinsic error resilience [3, 134], approximate computing can be used to optimize area and power consumption while achieving higher operating frequency. A typical integer multiplier consists of three stages: partial products generation, accumulation, and addition. Approximate multipliers are based on various techniques. A common way to approximate multiplication is to truncate the inputs, for instance, by dynamically identifying the most important bits and performing exact multiplication of the smaller segments, e.g., the Dynamic Segment Method (DSM) [132] and the Dynamic Range Unbiased Multiplier (DRUM) [71]. Inputs can also be rounded to the nearest power of two value to substitute multiplication with addition and shifting operations, as in RoBA multiplier [220]. Other methods include logic simplification for the addition of partial products, such as inexact adders and compressors [165], simplification of encoders in Booth

multipliers [69], and the utilization of innate characteristics of emerging technologies, e.g., memristor-based multipliers [134]. To compensate for introduced inaccuracies, many approximate multipliers include error recovery; e.g., DRUM [71] ensures that errors are centered around zero, so they cancel each other out during repeated calculations rather than accumulating. Another approximation technique is to switch to a logarithmic number system and compute addition instead of multiplication. The general idea is to take the logarithms of the inputs, sum them, and convert the output to the final result via an antilogarithm [55, 125]. The complexity and accuracy of this method come from the logarithm and antilogarithm steps. Truncating operands and approximating logarithm steps can further optimize area, speed, and power consumption. The first approximate logarithmic multiplier was proposed by Mitchell, who used binary logarithms to approximate multiplication [125]. Several studies have been conducted to improve the performance and accuracy of this method for DNN applications [7]. TOSAM is a scalable approximate multiplier that reduces computational complexity by truncating and rounding each input operand based on its leading one-bit position [196]. ScaleTRIM exploits curve fitting and linearization to fit the product of truncated inputs and approximate a correct result, and employs an error compensation method using lookup tables [55].

2.4 Hardware Reliability

Reliability of an electronic system is the ability to perform its intended function over the lifetime of the targeted application. It is the probability that no failure or user-visible error occurs during system operation in the time period $(0,t]$. This way, reliability can be expressed as $R(t) = P(T > t)$, where T is the lifetime of the system [129]. A faulty system can still be reliable if the fault is masked or mitigated.

Reliability is tightly connected to the availability of the system. **Availability** is the probability that the electronic system is functioning properly at any given time t. While reliability is defined over a period of time, availability is defined at a particular time instance. Availability considers the possible downtime of the system due to repairs and thus can be expressed as the ratio of system uptime to total system operating time [129].

Reliability and availability are components of the broader concept of dependability.

2.4.1 Faults, Errors, and Failures

A **fault** is a physical defect in the circuit. Faults can be caused by manufacturing defects or appear during the system's lifetime as a result of aging or environmental stress.

An **error** occurs when a fault propagates to the interface of the system component. However, not all faults cause errors. Faults can be masked or mitigated using fault tolerance mechanisms. Fault masking includes logical, electrical, and timing masking mechanisms [204]. In the case of logical masking, the fault is masked by the design structure and other signal values. Electrical masking occurs when a faulty transient pulse weakens as it propagates through combinational logic gates, so it is not latched by the sequential logic. Timing masking occurs when a faulty value reaches the sequential element outside its latching window.

A **failure** is a deviation of a system functionality observable at the top-level interface. A failure occurs when an error propagates to the top-level system's interface. Errors can be masked at higher system levels as well. Failures observed at the top-level system interface can be classified into two categories: Silent Data Corruption (SDC) and Detected Unrecoverable Errors (DUE). DUEs are usually caused by faults in control signals, causing functional interrupts and system crashes. SDC, on the other hand, means that the sys-

tem's output data differs from the expected output, without any indication that a fault has occurred.

Based on their duration, faults are categorized into transient, permanent, and intermittent. Permanent faults remain indefinitely until they are removed by some intervention or correction mechanism. Permanent faults are primarily caused by manufacturing defects and aging mechanisms. Transient faults appear briefly and disappear without intervention. The primary cause of transient faults is terrestrial and cosmic radiation. Intermittent faults are recurring transient faults that reappear after usually irregular intervals. Intermittent faults can, over time, become permanent faults [129].

A **fault model** is the way a logical fault models a physical defect. Common fault models include stuck-at-0 and stuck-at-1 for modeling the effect of memory cells and logic gates permanently stuck at the logic value zero or one, delay faults for modeling operating frequency degradation due to aging, and bit-flips to model transient faults that cause the transition of one logical value to another [204].

Chains fault $\rightarrow$ error $\rightarrow$ failure show how a fault in a single component propagates through the system layers. A cross-layer approach to reliability requires an understanding of the fault's source, fault propagation, and manifestation mechanisms.

2.4.2 Soft errors

Hardware faults can be caused by various reasons: radiation, aging, manufacturing process variations, manufacturing defects, and early-life failures. As stated in the Introduction, radiation-induced soft errors are considered dominant for today's digital systems.

Faults caused by radiation are divided into two groups: hard errors, which lead to the destruction of the component, and soft errors, which cause the device to behave incorrectly. While soft errors are temporary and do not permanently damage the device, they can also lead to system failure and catastrophic consequences.

The most common type of soft error is a Single Event Upset (SEU), a change of the logic state of the sequential element, i.e., a single bit flip. Less common but more critical is Single Event Functional Interrupt (SEFI)—a soft error that causes a functional interrupt in circuits, interfaces, or entire chips. Another common type of soft error is a Single Event Transient (SET), a current or voltage spike in a signal. SET is transitory in nature; however, if captured by a sequential element, it can become an SEU, which is more likely in modern technologies as clock frequencies increase. Other types include Multi-Cell Upset (MCU), a corruption of adjacent memory cells or flip-flops; and Multi-Bit Upset (MBU), a corruption of different bits in the same data word [89].

2.4.3 Reliability evaluation

Reliability of hardware architectures can be evaluated using various methods, including fault injection, irradiation, analytical evaluation, and hybrid approaches that combine multiple methods. **Fault injection** is the most commonly used method for reliability evaluation and involves introducing faults into system components to analyze its behavior in the presence of errors [54]. It can be performed using software-based simulation or hardware-based emulation. Software-based fault injection offers high control over the process, while emulation leverages hardware speed. Fault injection can be performed on different levels of abstraction. Low-level simulations, e.g., at the gate and register-transfer level (RTL), are more accurate but more time-consuming. High-level simulations, e.g., at the model and application levels, are faster but less accurate [139].

Irradiation is a special case of fault injection that uses a particle beam to irradiate the hardware platform, simulating a real faulty environment [54]. While radiation experi-

ments are used as a realistic error-rate radiation source, they do not allow tracking of fault propagation and lack control over injected faults. Moreover, irradiation is an expensive method and cannot be performed during the design stage, as it requires a manufactured device.

Analytical methods model system functionality and reliability using mathematical equations. Analytical methods enable fast and scalable evaluation by eliminating the complexity of fault simulation. However, they often lack accuracy [6].

Hybrid methods benefit from the strengths of combined approaches and allow for compensation of the drawbacks of one approach by the other, as the balance between performance and accuracy is an important aspect of reliability evaluation methods [6].

3 Understanding Fault-tolerance Vulnerabilities in Advanced SoC FPGAs for Critical Applications

This chapter presents a survey on fault-tolerance vulnerabilities in modern heterogeneous systems, such as advanced SoC FPGAs. It includes a taxonomy of an advanced SoC FPGA structure; an analysis of fault effects caused by faults in the different components, identifying the most critical and vulnerable components; a review of vendor-provided fault-tolerance mechanisms and an analysis of fault-tolerance vulnerabilities and gaps; as well as a review of complementary user-space fault-tolerance solutions. The chapter is based on publication [1]:

> N. Cherezova, K. Shibin, M. Jenihhin, and A. Jutman. Understanding fault-tolerance vulnerabilities in advanced SoC FPGAs for critical applications. *Microelectronics Reliability*, 146:115010, 2023

3.1 Introduction

Modern heterogeneous SoC FPGAs, which provide software programmability through scalar and vector processors and hardware programmability through FPGA logic, are getting more complex and advanced, integrating more components on a single chip. However, their dense structure is also increasingly susceptible to various hardware faults. Their reliability assessment presents certain challenges: simulation and testing are difficult due to the design's complexity, and faults in one core can propagate to other cores [58].

The heterogeneous nature of SoC FPGAs means that faults in different components manifest differently and require tailored fault-tolerance solutions, making reliability assurance for the chip challenging.

3.2 Faults and fault effects in SoC FPGAs

This section presents a typical structure of a modern FPGA-based SoC, along with faults and fault effects characterizing each component.

3.2.1 Typical structure of modern FPGA-based SoC

The structure is based on the latest series from AMD-Xilinx, Intel, and Microchip: Versal ACAP (Adaptive Compute Acceleration Platform) [206] and Zynq UltraScale+ [211] (AMD-Xilinx); Agilex [82] and Stratix 10 [85] SoC families (Intel), and PolarFire SoC [124] (Microchip). A modern FPGA-based SoC includes multi-core processors (ARM Cortex-A and Cortex-R cores in AMD-Xilinx and Intel devices, and RISC-V cores in Microchip SoC), FPGA logic, hard peripheral controllers, embedded on-chip memories, and system infrastructure represented by non-functional utility resources, such as chip monitors, ECC controllers, test facilities, etc. The most notable among them is a separate processor that acts as a device manager. Device manager, called Platform Management Unit (PMU) in AMD-Xilinx devices, Secure Device Manager (SDM) in Intel, and monitor core in Microchip, sets the root-of-trust, configures the device after the power-up, and monitors its state during the lifetime. Some platforms have accelerators, like embedded GPU for video rendering or custom engines for AI and ML tasks processing (Zynq UltraScale+ and Versal), and specialized modules, like radio frequency (RF) data converters and video processing units (Zynq UltraScale+). Versal ACAP represents the new generation of programmable SoCs from AMD-Xilinx and has the most complex structure. The processing system, FPGA logic, AI Engines (custom blocks for fast data processing), and peripheral controllers are con-

nected through Programmable Network-on-Chip (PNoC). One can assume that the next FPGA-based SoCs, generally, follow the Versal model.

The typical structure described is presented in Figure 6. The presented structure can be considered as a super-set. More complex system-on-chips include more elements from the set, and less complex ones include fewer, but they still stay within the given super-set.

3.2.2 Faults and fault effects in different components

Below is the analysis based on the literature survey of the most common types of errors caused by faults in different components and the most critical and vulnerable elements in each module. Table 1 summarizes fault effects at the module and system levels. System failures are divided into three classes: system crash (SC), application crash (AC), and application silent data corruption (ASDC). The most common system-level failure is ASDC. AC can be caused by control flow errors, and SC might be caused by errors related to the SoC configuration or control circuitry.

Processing system (PS). Structurally, faults for a processor were organized into three large groups: faults in registers (including general-purpose and special-purpose ones), caches, and data processing units (ALU / FPU) [52, 87, 182, 183, 191, 216]. It can be seen that faults in different components affect processors in two ways: silent data corruption (SDC) and control flow errors that can lead to functional interrupts or crashes. Functional interrupts can be caused by errors in control or special-purpose registers, errors in addressing, and errors in instruction (wrong instruction, wrong operand, or wrong immediate) [52]. While functional interrupts require a reset and result in the system being unavailable during the restart, SDC does not disable the system, but it might be of great concern for a user, as they corrupt results without giving any indications that something went wrong. The most critical elements are registers (particularly special-purpose ones) and caches [52, 87, 139, 182, 183, 191, 216] since faults in those components directly affect the data being processed and can introduce control flow errors. Additionally, studies show that processor cores integrated into SoC platforms have higher rates of crashes and functional interrupts compared to standalone processors [19].

FPGA. Structurally, faults in FPGA can be organized into three groups as well: faults in configuration memory, user memory, and control-related logic [147]. Unlike processors, the functionality of FPGA is not hardened and is defined by the values loaded into configuration memory (called CRAM). Therefore, the main fault effect for them is the change in circuit functionality, which might result in silent data corruption as well as application or even system crash.

Configuration memory consists of routing elements and logic resources, such as LUTs (Look Up Tables) and control bits.

Routing resources can be local (inside logic blocks) or global (between logic blocks). Local routing is represented by multiplexers (MUX) controlling inputs and outputs of the logic block and LUTs. Logic blocks are connected through switch boxes and connection boxes by means of Programmable Interconnect Points (PIPs). PIP may be represented by a simple pass transistor controlled by one configuration bit or a complex structure created using several multiplexers controlled by a group of configuration bits. For PIP, the following fault models are defined: Open, Conflict, Input Antenna, Output Antenna, and Bridge [15]. Open fault happens when a previously used connection is disabled. Conflict is characterized by a new connection being added between a used input node and a used output node. Input Antenna is a new connection between an unused input node and a used output node. Output Antenna is a new connection between an unused input node

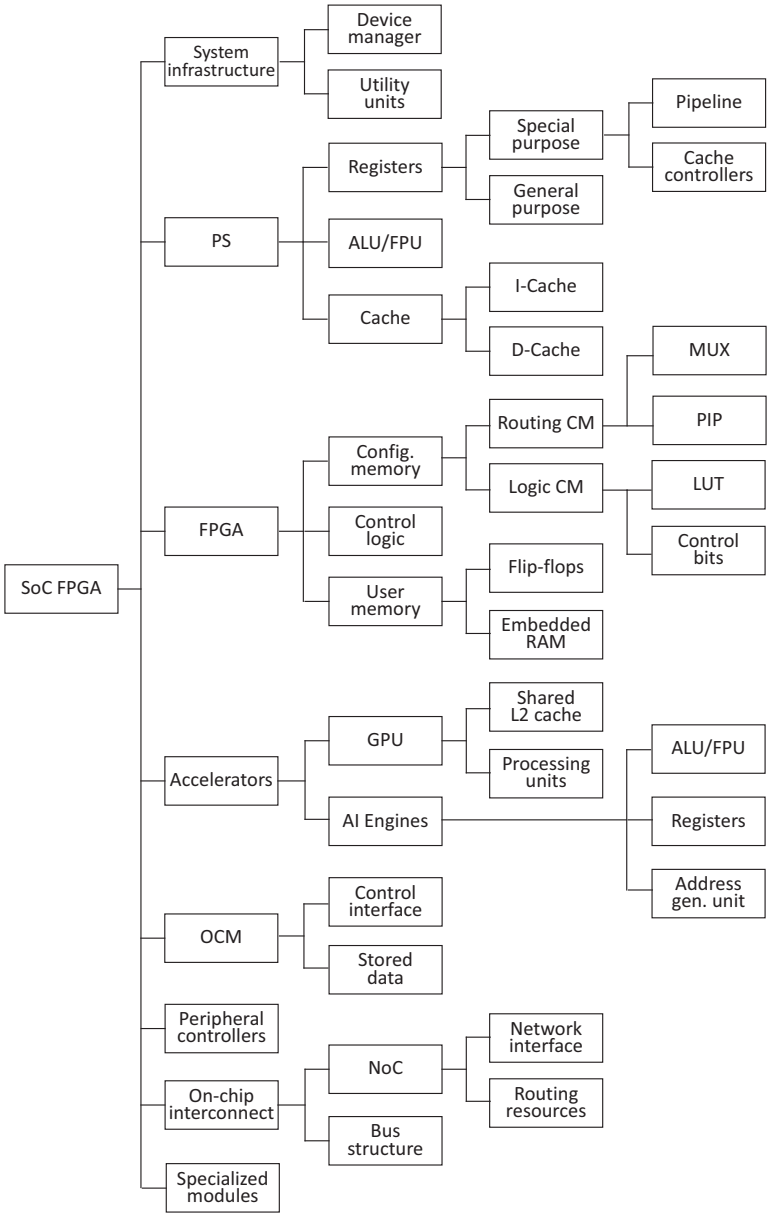


Figure 6: Typical structure of modern FPGA-based SoC

and a used output node. Bridge happens when a previously used connection is disabled, and an unrelated input is connected to the output from the disabled connection.

Along with the change in circuit functionality, SEU in routing configuration bits that create an undesired bridge between a signal and an unused wire adds an extra parasitic capacitance (and thus an extra delay) to the signal path. The impact is cumulative and can lead to delays that are significant enough to cause delay faults [187].

Routing faults are the most common and the most critical for FPGAs since the routing structure occupies a large portion of the FPGA area and configuration bits [133]. Moreover, the most failure-sensitive routing elements are local MUXes [66, 92].

Faults in logic include LUT values change and control bits changes. LUTs are used to hold the data (when used as distributed RAM or shift register) and implement data processing or Finite State Machine (FSM) to control the workflow. Therefore, faults in LUT values can cause data corruption as well as control flow errors. However, whether it produces an error or not depends on the inputs to the circuit and which part of the logic function is impacted.

On the contrary, changes in control bits generally cause errors for all, or almost all, possible circuit inputs. Control bits define LUT mode (whether it is used as a LUT, distributed RAM, or a shift register), configure storage elements as flip-flops or latches, and define Input/Output Block (IOB) parameters, such as electrical standard, mode (input, output, bi-directional), termination (pull-up or pull-down resistors), signal drive strength, optional storage element, input delay, and a slew rate. Faults in IOB parameters can create unintended outputs [152] and additional delays [145, 185, 186].

User memory includes embedded RAM and flip-flops in the logic blocks. It stores the data produced by the application. Faults in user memory are straightforward and cause silent data corruption. While faults in embedded RAM data remain until a new value is stored there or the fault is corrected, original flip-flop values might be restored in the next clock cycle. Faults in user memory are considered less critical than faults in the configuration memory. The former ones do not affect the circuit functionality and can be fixed without device reconfiguration.

Faults in control logic include faults in reset circuitry and configuration ports, such as JTAG TAP (Test Access Port) controller, ICAP (Internal Configuration Access Port), PCAP (Processor Configuration Access Port), and SelectMAP interface. SEU in reset circuitry can clear out the device functionality or all of the flip-flops. Faults in the JTAG TAP controller can damage the FPGA or other system devices as well as cause a loss of control [215]. Faults within the programmable SelectMAP interface configuration pins result in a malfunctioning interface: the corrupted interface can return bad values during a read operation and corrupt the configuration during a write operation [147]. Faults in control logic have serious consequences and require device reconfiguration, though, in practice, their rates are very low [147].

Moreover, in SoC, the FPGA part is proved to be more prone to soft errors [9, 52, 79], while processors and peripherals have more vanished errors.

Accelerators. Two types of accelerators are considered in this work: embedded GPU and custom AI Engines suggested by AMD-Xilinx. Versal AI Engines [208] consist of two processors, for scalar and vector arithmetic, and thus their structural faults are similar to the processor faults. Faults in instructions can cause data and control flow errors, and faults in registers and in data processing units can cause silent data corruption. AI Engines do not have cache memory, though they do have shared memory for storing the data.

Table 1: Faults and fault effects in SoC FPGA

Fault location		Error within module	System level failures		
			SC	AC	ASDC
Processor					
Register file	General-purpose registers	Silent data corruption, control flow errors	×	●	●
	Special purpose registers	Control flow errors	×	●	○
Cache	Instruction cache	Control flow errors	×	●	○
	Data cache	Silent data corruption	×	○	●
Data processing units (ALU/FPU)		Silent data corruption	×	○	●
FPGA					
Configuration memory	Routing elements	Functional errors (change in circuit functionality)	●	●	●
	Logic resources				
User memory	Flip-flops	Silent data corruption	×	○	●
	Embedded RAM				
Control logic	Reset	Configuration loss, functional interrupts	●	●	●
	JTAG TAP controller				
	SelectMAP interface				
Accelerators					
GPU	L2 shared cache	Silent data corruption, control flow errors	×	×	○
	Processor units				
AI Engine	Register file	Silent data corruption, control flow errors	×	×	●
	Data processing units				
	Address generation unit				
OCM and peripherals					
OCM	Control interface	Silent data corruption, control flow errors	○	●	●
	Stored data		×	○	●
Peripheral controllers		Control flow errors	●	○	○
On-chip interconnect					
NoC	Processing units	Silent data corruption, configuration loss	●	●	●
	Connections				
Bus structure		Silent data corruption, configuration loss	●	●	●

SC — system crash. AC — application crash. ASDC — application silent data corruption.
× — not expected. ○ — possible. ● — highly likely.

For GPU, faults in global buffers (shared L2 cache) can be considered the most critical, since they may spread in multiple locations and can cause many erroneous outputs [111, 136, 162].

Faults in accelerators are not expected to cause crashes but rather produce wrong outputs. While data errors in machine learning applications might have serious consequences, erroneous pixels in a video frame just degrade the user experience.

Peripherals and memory controllers. Experiments show that faults in address and control registers cause many data errors and interrupts [51, 52]. However, there are only a few works that explore faults in so-called "uncore" components, such as peripherals, on-chip memories, etc. [39, 40, 113]. However, loss of configuration in peripheral controllers might result in a system crash and require device reconfiguration.

On-chip interconnect. Different components in heterogeneous SoCs are usually connected through a bus-based communication architecture. However, Versal ACAP represents an interesting case that additionally utilizes network-on-chip (NoC). Versal NoC architecture is similar to a typical NoC structure: network interface units connect processing elements to the NoC, and routers perform the transportation of the data through the NoC. Network interface units are represented by NoC Master Units (NMU) and NoC Slave Units (NSU) that denote ingress and egress blocks accordingly. Routers are represented by NoC Packet Switch (NPS). It is a crossbar switch that determines the direction of the packet based on the special routing tables. Though Versal NoC is programmable, its configuration is statically defined during the design time, therefore, it includes an additional unit called NoC Peripheral Interconnect (NPI). NPI is an internal register programming interconnect of the NoC that controls and configures NMU, NSU, and NPS. The NPI programs NoC registers that define the aforementioned routing tables [209].

In the classic NoCs, faults in interconnects are considered dominant [140, 146]. Additionally, since the configuration of Versal PNoC is programmable, faults in routing components and NPI registers can have serious consequences. They can lead to a loss of configuration and, therefore, to an application or a system crash.

For the AMBA AHB system bus, experiments in [27] show that SDC is the most common error, however, faults in the address bus have a high probability to cause an application or system crash.

3.3 Vendor-provided fault tolerance mechanisms

The heterogeneous nature of modern SoC FPGAs, combining different components, requires a complex fault tolerance solution that covers the whole chip.

The main fault detection and fault tolerance mechanisms employed by vendors are the following:

- Device manager
- Modular redundancy (usually TMR)
- ECC, CRC, parity checking
- Data scrubbing
- Dual lock-step
- Memory protection units
- Built-in tests

- Watchdog timers

A comparison between fault detection and fault tolerance methods on different platforms is presented in Table 2.

Device manager. As already mentioned, all considered platforms have a dedicated processor or two acting as a device manager: Platform Management Unit (PMU) on AMD-Xilinx platforms, Secure Device Manager (SDM) on Intel platforms, and monitor core on Microchip PolarFire SoC. The device manager holds the root-of-trust, configures the device after the power-up, performs error management, and monitors its state during the lifetime to support the functional safety of the platform.

Triple-modular redundancy (TMR). On AMD-Xilinx and Intel platforms TMR is used to protect the device manager and critical state registers. Additionally, Intel and Microchip offer functionality to generate a triplicated design for FPGA logic. AMD-Xilinx offers similar functionality for Virtex family devices only. For modern platforms, it offers TMR MicroBlaze IP (soft processor).

Error correction code (ECC). ECC is the standard protection mechanism for memories. AMD-Xilinx, Intel, and Microchip have ECC (single-error correction, double error detection) on all processor memories, on-chip memories, FPGA embedded memories, and DDR4 controllers. ECC mechanism supports error injection functionality. On Intel platforms, USB, SD/MMC, EMAC, DMA, and NAND flash controllers have integrated ECC-protected memories. While most ECCs provide a single-error correction, double-error detection, for M20K blocks (internal 20 Kbit memories), Intel suggests single-error, double-adjacent-error, and triple-adjacent-error correction in a 32-bit word. However, it is reported that with the ECC feature enabled, M20K runs slower [83]. In order to decrease the possibility of multi-bit errors, all vendors use a memory interleaving technique.

Dual lock-step. AMD-Xilinx platforms include ARM Cortex-R5F dual-core processor that supports the lock-step functionality. In the lock-step mode, two cores perform the same task in parallel.

Data scrubbing. AMD-Xilinx and Intel, both suggest solutions for soft-errors correction in the FPGA configuration memory (CRAM) based on the background scan. Xilinx Soft Error Mitigation (XilSEM) is available as soft IP in the UltraScale+ series [207], however, in the Versal series, it is a part of PMU [210]. Triplicated PMU provides more protection for the module itself. XilSEM uses ECC and CRC for single-error correction and double-error detection. On Versal, XilSEM can additionally detect errors in NPI (NoC Peripheral Interconnect) registers that connect NoC with some modules using SHA techniques. XilSEM supports error injection functionality.

Intel Error Detection Circuit (EDC) with integrated ECC corrects single- and double-bit errors and detects up to 8 bits that fit in a rectangular box (8×1, 4×2, 1×8, 2×4). EDC supports error injection and priority scrubbing. Some portions of the design can be marked as high-priority sectors, those sectors are scanned more frequently. Additionally, there is a sensitivity processing and hierarchy tagging functionality that defines the criticality of the CRAM bit and suggests corrective actions based on the classification [84, 86].

Since PolarFire SoC incorporates flash-based FPGA, which is more robust towards soft errors, there is no scrubbing solution suggested.

Memory protection units (MPU). While memory protection units are not explicitly designed for soft error protection, they can help with them as well. One of the main purposes of MPU is to restrict memory access of a process to some defined region. If there is an error in the address due to the SEU so that it no longer belongs to the allowed region, access is blocked, which might prevent SDC or control flow error.

Table 2: Vendor-provided fault tolerance mechanisms in different platforms

Method	AMD-Xilinx		Intel		Microchip	
	Realization	**What is protected**	**Realization**	**What is protected**	**Realization**	**What is protected**
Device manager	Platform Management Unit (PMU)	System	Secure Device Manager (SDM)	System	RISC-V Monitor core	System
Modular redundancy	Hard TMR	Device manager + critical state registers	Hard TMR	Device manager + critical on-chip state machines	—	
	—		Wizard for TRM generation on FPGA logic	Design on FPGA logic	Wizard for TRM generation on FPGA logic	Design on FPGA logic
ECC	SECDED	Caches, OCM, FPGA embedded memories, DDR4 controller	SECDED / multi-bit correction	Caches, OCM, FPGA embedded memories, DDR4 controller, NAND flash controller	SECDED	Caches, FPGA embedded RAM, DDR3/4 controller
Lock-step	Dual lock-step	Cortex-R5F core	—		—	
Data scrubbing	XilSEM IP	FPGA CRAM	Intel EDC feature	FPGA CRAM	—	
Memory protection units	ARM TrustZone, XMPU, XPPU	Memories, peripherals	ARM TrustZone, SMMU	Memories	MMU, PMP	Memories
Built-in tests	LBIST	XMPU, lock-step, and ECC checkers	*(No info found)*		MBIST	L2 cache
	MBIST	Cortex-R5F core memories				
	STL	memories, PMU, XMPU, XPPU, clocks, voltages, temperature				
Watchdog timers	Hardware watchdogs	Processing system	Hardware watchdogs	Processing system	Hardware watchdogs	Processing system

AMD-Xilinx platforms have a Memory Protection Unit (XMPU) and Peripheral Protection Unit (XPPU) to protect on-chip memory blocks and peripheral programming registers [210]. Intel platforms have System Memory Management Unit (SMMU), and one of its functions is memory access permissions. PolarFire has Memory Protection Unit as well. Additionally, since AMD-Xilinx and Intel SoC include ARM processor cores, they are protected by ARM TrustZone.

Built-in tests. AMD-Xilinx provides LBIST for check logic such as XMPU, lock-step, and ECC checkers, and MBIST for Cortex-R5F core memories. Additionally, memories, lock-step functionality, device manager, XMPU, XPPU, clocks, voltages, and temperature can be checked by the dedicated software test library (STL) [210]. Microchip provides MBIST for L2 cache [123].

Watchdog timers. Used for detection of the hanged or stuck in an infinite loop process, watchdog timers are provided by all three considered vendors.

3.4 Analysis and conclusions about the gaps

3.4.1 Vendor-provided fault tolerance methods analysis

Fault tolerance methods described in the previous section have certain overheads, such as area, power consumption, or execution time. They also have functional limitations on what kind of errors they can detect or correct and how fast they can do it. Therefore, for this study, it is important to see how vendor-provided methods protect the system from soft errors and what parts of SoC FPGAs are left unprotected.

Device manager. Implementation of the device manager requires one or two additional processors, which results in power consumption overhead. Moreover, the device manager does not protect the system from soft errors.

Triple-modular redundancy. TMR is based on utilizing three copies of the same module working in parallel, which introduces more than 200% overhead in the area and power consumption. This might be critical for energy-restricted applications. Therefore, TMR is only used to protect the device manager which acts as a safety island.

ECC. When enabled, ECC requires additional bits to store the codes, which results in 13 and 22% overhead for 32-bit and 64-bit data accordingly. In most cases, it can only correct single-bit errors and detect double-bit errors. Multi-bit correction and detection versions are possible but might slow down the memory. Since ECC is a memory protection technique, it mostly prevents SDC errors, however, in the case of Instruction Cache, it might prevent control flow errors as well.

Data scrubbing. The scrubbing solution that scans FPGA configuration memory in the background brings power consumption overhead. It has a latency of several milliseconds, during which the device might operate incorrectly. Studies show that scrubbing does not protect the device against immediate failures since it cannot fix errors propagated from faults in configuration memory [76]. Its main functionality is to prevent the accumulation of faults. Besides, it is not intended for use in a high-irradiation environment, such as space. An experiment in [202] shows the significant decrease in neural network accuracy deployed on the AMD-Xilinx UltraScale+ platform under beam radiation testing even with XilSEM enabled.

Dual lock-step. When two cores are running in the dual lock-step mode, they perform the same tasks with one clock cycle shift, which means the system can perform fewer tasks than in the split mode. Furthermore, since lock-step is a fault detection technique, it requires additional time to rerun the task in case of an error. Additionally, lock-step functionality is not supported by every processor.

MPU. As it was mentioned before, the memory protection unit is not designed to handle faults, though it can detect faults in some cases and prevent them from propagating further.

Watchdog timers. As a symptom-based fault detection method, the watchdog timer can neither isolate the fault nor correct it. The watchdog timer can either create an interrupt or restart the processor core.

3.4.2 Gaps in vendor-provided protection

Based on the information from previous sections, the fault tolerance taxonomy for modern SoC FPGA was created (Figure 7). Critical and vulnerable components pointed out by research papers are marked by a red square, and components that have some vendor-provided fault detection or fault tolerance solutions are marked using corresponding colors.

As can be seen from the figure, the most protected module in the modern SoC FPGA is the processing system that has watchdog timers, lock-step capabilities, ECC for caches, and memory protection units. However, it should be noted that not every processor supports lock-step functionality, watchdog timers cannot isolate and recover the fault, and memory protection units are not designed to handle faults but can only detect errors in some cases. ECC prevents errors originating from caches, however, the lack of protection in the other components of the processor, including registers that are considered critical as well, means that faults stemming from the other components are not covered. The lock-step mechanism protects the whole processor, however, in the considered SoC families, lock-step functionality is only supported in the ARM R-cores, leaving A-cores vulnerable.

The next protected module is FPGA logic which has scrubbing solutions for configuration memory and ECC for embedded memories. User memory flip-flops, though, are not covered by any means. The least protected modules are peripherals, interconnects, accelerators, and specialized modules, the ones that are less explored in the research articles as well.

The following gaps can be formulated:

1. Most of the vendor-provided fault-tolerance methods protect the processing system and memory components. ECC effectively covers single-bit faults in memories. Since memories usually occupy the major part of the chip, a high percentage of fault tolerance can be achieved by covering only memories using a considerably inexpensive and easy-to-implement mechanism like ECC. However, that leaves computational and control units less protected. They might occupy smaller area than memories, but faults in those components might lead to a crash.

2. The least protected modules are uncore components that are less considered in the literature as well. However, due to the complexity of the SoC, it is not enough to protect only the processing system. In fact, all components may have an impact on the reliability of the whole system [135].

3. The vendor-provided fault-tolerance methods for cost-efficient COTS devices are not designed for harsh environments or critical applications.

4. Different fault effects and used fault-tolerance methods require a systematic and hierarchical approach for efficient device protection. However, the current approach seems fragmented. The main reason to use the SoC FPGA platform is to use as many of its components as possible, not only the processor or the FPGA. In such cases,

a communication system, for example, plays an important role. For the application to run correctly, it is important to protect the whole system.

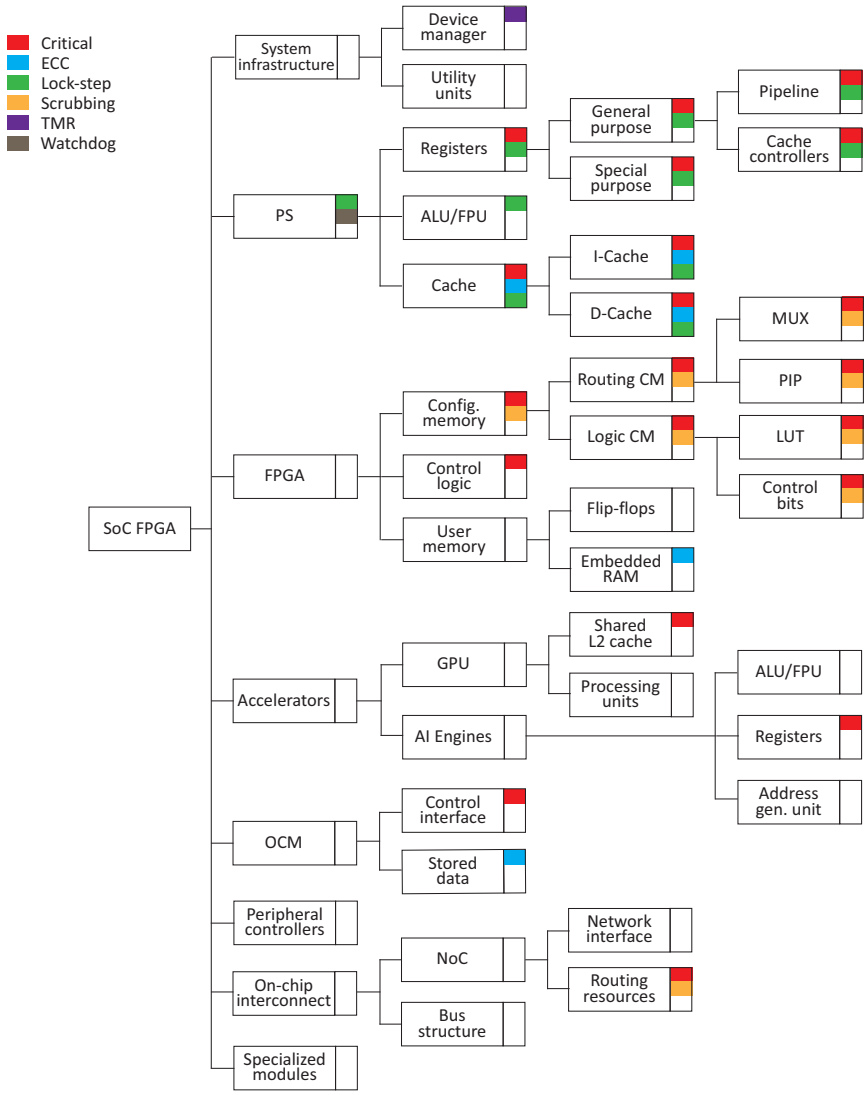


Figure 7: Fault tolerance taxonomy for SoC FPGA based on vendor-provided methods

3.5 Mapping known user-space methods

Clearly, all major FPGA vendors are seeking the most cost-efficient methods to offer basic protection against in-field failures. Advanced or full protection becomes too expensive as the main volume of FPGA devices goes into commercial applications such as industrial, telecom, or video processing, where fault tolerance mechanisms would only introduce additional cost and overhead and, hence, reduce the vendor's competitive advantage. Therefore, gaps in protection are inevitable and have to be addressed carefully by the

users targeting commercial off-the-shelf (COTS) devices for mission or safety-critical applications such as aerospace.

The pursuit of improving fault tolerance is not limited to the features that the manufacturer embeds into their devices. There is ongoing research on methods to analyze and improve the fault tolerance [119] using the controls available to the user of an SoC, such as programming of the processor cores or the FPGA logic in a way to make the operation more reliable. Several of those methods typically exploit redundancy that already exists or can be created by the user.

Since the modern SoC FPGAs inevitably contain heterogeneous modules, which are also commonly found separately, it makes sense to look at the methods that are already employed for such modules.

General-purpose processing units (PS). The availability of multi-core CPUs in modern SoCs allows for exploiting the inherent redundancy for improving transient fault tolerance in software. Where available in hardware, dual-core or triple-core lock-step redundant core configurations can provide significant protection against transient errors [43, 87]. A hybrid lock-step approach with hard CPU cores in combination with redundant soft cores is also possible and demonstrated in [94]. Fault tolerance methods described in [144] allow using temporal redundancy and target non-protected COTS CPUs. They enable protection by trading higher performance and a larger quantity of these CPUs (compared to fault-tolerant alternatives, if any are present) for fault tolerance by employing redundant code execution with some variable but predictable performance penalty. These methods use structural or temporal *active* redundancy where the reaction to fault detection as well as recovery from an error can be flexible and can depend on the executed application. Another software-centric approach proposed in [168] uses process-level redundancy (PLR), where a set of redundant copies of an application process is created and the results are systematically compared to guarantee correct execution. Redundant multithreaded processes are proposed for fault tolerance in user-space programs running on multi-core CPUs [131] where the results of two threads are compared for error detection while another thread is used as a watchdog. There are other similar methods from other groups which aim at increasing fault tolerance in software [117, 173]. Several methods aim at soft-core CPUs. One of them uses a combination of redundancy in multi-core processors and extra hardware measures for managing redundancy configurations in asymmetric multiprocessing mode (AMP) [112]. In [59], the authors implement fault detection in software through thread-replication and coarse-grain lock-step for weakly coupled processing cores.

FPGA fabric is configurable and, from the functional point of view, can work like any other digital hardware and, therefore, can implement hardware methods like TMR, ECC, and others in so-called soft cores. The underlying physical structure of FPGA, however, is very different from usual digital hardware because the function depends on the configuration memory as described earlier. FPGA works properly as long as the configuration memory is intact. In the case of SRAM-based FPGA, the configuration memory is susceptible to SEUs, and these may result in a change of the function. Therefore, usual hardware fault-tolerance methods need to be applied carefully. For example, TMR can be combined with scrubbing [14] in order to detect and correct the errors in the configuration memory, while TMR protects from soft errors in user logic memory. SEUs in configuration memory can also affect routing, therefore partitioning and placement of the logic in an FPGA can be exploited to increase fault tolerance [57, 100]. For example, it can change the probability of errors affecting TMR [116]. Other methods also aim at repairing broken configuration by partial dynamic reconfiguration [102].

Accelerators. Various hardware accelerators are used to increase the performance and/or efficiency of the processing on SoCs, and they, too, can benefit from added fault tolerance. For example, neural network accelerators are susceptible to hardware-induced faults, but there are methods to mitigate the effects of faults [70]. For fixed-hardware neural network accelerators, there are software methods like result range restriction [74] and algorithm-based fault tolerance [222]. Soft-core accelerators can be augmented with an extra layer for improved fault tolerance [77].

3.6 Chapter conclusions

This chapter reviewed the state of the art in fault-tolerance mechanisms offered by SoC FPGA vendors with a focus on the most popular COTS devices of the latest SoC families. There has been substantial progress in such protection mechanisms during the last decade as opposed to simple technology-level hardening. As a result, there is a clear industrial trend to build mission- and safety-critical systems around COTS devices [23]. For instance, in the space segment, this trend is further escalated by the need for in-orbit high-performance computing [109].

Still, as the vast majority of SoC FPGAs are employed in non-critical commercial applications, there are gaps in the protection offered off the shelf by vendors. Advanced SoC FPGAs comprise a diversity of tightly integrated components that should all be protected to ensure the system's reliability and availability. However, as was shown, the most protected components are the processing system and memories, leaving the rest of the system vulnerable. Uncore components are not protected and are less considered in the literature. The current approach to fault tolerance is fragmented and not systematic. Finally, vendor-provided fault-tolerance solutions for COTS devices are not intended for harsh environments and critical applications.

Hence, it is necessary to implement complementary user-space fault-tolerance mechanisms, tailoring them to target mission and application profiles, from simple custom TMR [63] to advanced fault handling schemes [56]. For better protection, user-space methods should interact with vendor-provided ones, resulting in an integrated fault-tolerance solution. For example, the scrubbing procedure can be triggered by a TMR voter in the FPGA logic or lower-cost fault detection checkers or monitors in the FPGA logic. Also, in some scenarios, fault detection may be supported by employing higher-level techniques such as time-domain redundancy for unprotected processors, software-based checking, and external performance/process monitoring. Here, the user-space fault-tolerance mechanisms would both protect the operation and initiate the recovery procedure offered by the vendor, thus improving the overall protection efficiency. More advanced fault-handling mechanisms that involve cross-layer interactions and data exchange are also possible [167].

However, implementing fault-tolerance solutions on different layers without considering the whole picture might result in an overprotected system or even a degradation in the overall reliability. A prominent solution to cover the aforementioned gaps efficiently is a cross-layer reliability approach.

A cross-layer approach to reliability has been gaining popularity in recent years since it can help to develop a globally optimized fault management system. Several frameworks have been developed by the academic community for the reliability-aware design space exploration to find the best combination of fault-tolerance methods [32, 164] and for the cross-layer reliability assessment of the system [97]. Those frameworks primarily target processors at the early design stage, considering the HW/SW co-design process. However, as already mentioned, COTS devices are used more and more for critical applications.

4 Reliability Assessment of DNNs on Systolic Arrays

This chapter presents an analytical fault-propagation-based method, SAFPA, that enables fast evaluation and supports multiple systolic array dataflows. The method is built on cross-layer analysis that examines how faults in the hardware microarchitecture propagate through system layers and manifest in the application, namely DNNs. Experimental results on representative convolutional neural networks demonstrate that the method provides several orders of magnitude speedup over microarchitecture-level fault injection and highlights dataflow-dependent vulnerability differences. The method is used to assess and analyze the reliability of two modern DNN architectures for computer vision tasks, namely Convolutional Neural Networks and Vision Transformers, highlighting their differences. The chapter is based on publications [VII], [IV], [V], and [VI]:

> N. Cherezova, A. Jutman, and M. Jenihhin. FORTALESA: Fault-tolerant reconfigurable systolic array for DNN inference. *Microprocessors and Microsystems*, 119:105222, 12 2025
>
> N. Cherezova, S. Pappalardo, A. Ruospo, B. Deveautour, L. Fezza, A. Jutman, E. Sanchez, A. Bosio, M. S. Reorda, and M. Jenihhin. Special session: Reliability assessment of DNN models and inference on systolic arrays. In *IEEE VLSI Test Symposium (VTS)*, 2026
>
> N. Cherezova, M. Jenihhin, and A. Jutman. Scalable reliability assessment of vision transformers on systolic arrays via fault propagation analysis. In *IEEE 32nd International Symposium on On-Line Testing and Robust System Design (IOLTS)*, pages 1–7, 2026
>
> M. H. Ahmadilivani, N. Cherezova, M. Glaß, J.-D. Guerrero-Balaguera, M. Jenihhin, A. Kritikakou, R. L. Sierra, L. B. Poelhs, J. Raik, L. Roquet, F. F. dos Santos, F. A. da Silva, M. S. Reorda, A. Veronesi, E. C. V. Castillo, and J. E. R. Condia. Reliability assessment of deep neural networks and accelerators across design stages. In *IEEE 27th Latin American Test Symposium (LATS)*, pages 1–10, 2026

4.1 SAFPA: Fast Analytical Reliability Assessment for DNN Inference on Systolic Arrays

4.1.1 Introduction

Systolic arrays were first introduced in the 1980s [101] and recently became popular for accelerating DNNs. As the reliability of DNN accelerators comes to the front, recent works have proposed methods for the assessment and enhancement of systolic arrays. [194] analyzed the effects of transient faults in the combinational logic of systolic arrays. [184] presented Saca-FI, a microarchitecture-level fault injection framework for systolic array-based CNN accelerators, for the analysis of transient faults in registers. Saca-FI is based on SCALE-Sim, a cycle-accurate simulator for DNN inference on systolic arrays [161]. [2] proposed an RTL-level fault injection framework for the analysis of stuck-at faults within multiply and accumulate (MAC) units. [179] presented a fault injection framework that models systolic array dataflow using Uniform Recurrent Equations. Unlike previous works, the proposed reliability assessment method is based on fault propagation analysis. Instead of injecting faults in the microarchitectural or RTL model of a systolic array, which is a very time-consuming task, errors are added directly to the DNN layer output using fault propagation analysis, which significantly speeds up the fault injection process.

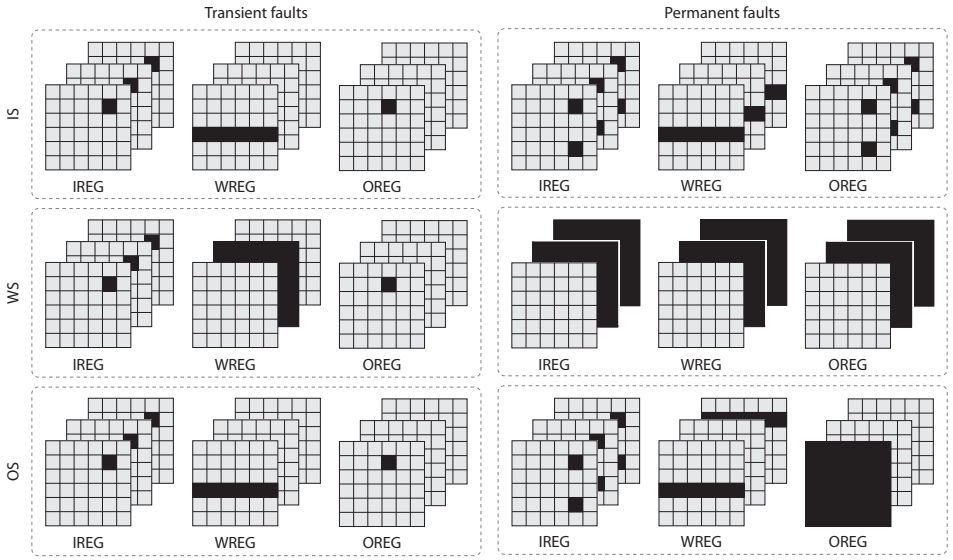


Figure 8: Error patterns in the output tensor for different systolic array dataflows

4.1.2 Methodology

This subsection presents the proposed reliability assessment method for systolic arrays (SAFPA). It combines fault injection with fault propagation analysis, which is used to calculate the resulting error in the layer output. This allows a noticeable speedup in the reliability assessment of DNN inference on a systolic array. The method allows analysis of both transient and permanent faults.

4.1.2.1 Fault propagation in systolic arrays

Faults in PE registers are considered for the analysis. Due to the regular structure of a systolic array and data movement, faults in different components produce distinct error patterns in the output tensor. Figure 8 presents error patterns for the convolution operation on different dataflows. Patterns are derived analytically and experimentally, using microarchitecture-level simulation of the systolic array, following examples from previous works for similar cases [21, 114].

Transient faults

Input register (IREG): For both OS and WS systolic array, a bit flip in an input register (IREG) results in the propagation of faulty activation through the whole row, affecting one value in several output channels. In the IS systolic array, the fault persists until a new activation tile is loaded, affecting one value in all output channels. This ends up in the *bullet pattern*.

Weight register (WREG): A bit flip in WREG, however, produces different patterns. In WS, fault persists until new weights are loaded, resulting in the *partial or whole channel pattern*. In OS, faulty weight propagates through the whole column, affecting several output values in one channel. In IS, faulty weight propagates through the whole row, still affecting several output values in one channel. This ends up in the *line pattern*.

Output register (OREG): A bit flip in OREG affects one value in one output channel, resulting in a *point pattern* for all dataflows.

Permanent faults

For WS systolic array, a stuck-at fault in all cases results in several whole channel patterns. For OS, a fault in IREG results in several bullet patterns; a fault in WREG results in several line patterns; and a fault in OREG results in several whole-channel patterns, since the results are offloaded by propagating from top to bottom. For IS, faults in IREG and OREG result in several bullet patterns; a fault in WREG results in a line bullet pattern.

4.1.2.2 Fault propagation analysis for transient faults

Each fault is defined by parameters given in Table 3.

Table 3: Transient fault parameters

Symbol	Description
f_{type}	Type of fault (IREG, WREG, OREG)
ts	Cycle of a systolic array execution
t_w	Weight tile
t_a	Activation tile
p_{row}	Row of the PE
p_{col}	Column of the PE
β_f	Bit index

An output of the convolutional layer can be expressed as

$$y_{(u,v)} = \sum_{c}^{C_{in}} \sum_{i}^{H_k} \sum_{j}^{W_k} w_{(c,i,j)} \cdot x_{(c,u+i,v+j)} + b\,, \tag{11}$$

where $y_{(u,v)}$ is the output value with index (u,v), C_{in} is the number of input channels, H_k and W_k are the height and width of a kernel, $w_{(c,i,j)}$ is an individual weight, $x_{(c,u+i,v+j)}$ is an activation (input), and b is a bias.

A bit flip in the numerical value can be expressed as an addition of an error term ε. For a signed integer, the error term is defined as

$$\varepsilon = 2^{\beta_f} \cdot \gamma, \tag{12}$$

where β_f is the index of a faulty bit and γ is a sign coefficient, which is derived by the following equation

$$\gamma = \begin{cases} -1 & \text{if} \quad x_f^{(\beta_f)} = 1 \wedge \beta_f \neq \beta_{sign} \\ 1 & \text{if} \quad x_f^{(\beta_f)} = 1 \wedge \beta_f = \beta_{sign} \\ -1 & \text{if} \quad x_f^{(\beta_f)} = 0 \wedge \beta_f = \beta_{sign} \\ 1 & \text{if} \quad x_f^{(\beta_f)} = 0 \wedge \beta_f \neq \beta_{sign} \end{cases} \tag{13}$$

where $x_f^{(\beta_f)}$ is the faulty bit of the affected value and β_{sign} is the index of a sign bit.

In the case of a floating point, the error term is different for the sign bit, exponent, and mantissa bits [53]:

$$\varepsilon = \begin{cases} (-1)^{b_f} 2^E 2^{\beta_f - 24} & \text{if} \quad 0 \leq \beta_f \leq 23 & \text{(mantissa)} \\ (2^{(-1)^{b_f} 2^{\beta_f - 24}} - 1)x & \text{if} \quad 24 \leq \beta_f \leq 31 & \text{(exponent)} \\ -2x & \text{if} \quad \beta_f = 32 & \text{(sign)} \end{cases} \tag{14}$$

where β_f is an index of a faulty bit, b_f is the value of the β_f-th bit, and E is an exponent of numerical value x, expressed as

$$E = \sum_{i=24}^{31} b_i 2^{i-23}. \tag{15}$$

A fault in a register holding activation, weight, or partial sum results in adding an error e to the output of the convolutional layer. This error e can be calculated for each type of fault from the parameters given in Table 3.

Transient faults in OS dataflow

Fault in IREG. If the fault happens in the input register, it corrupts an activation, then an error added to the output of the convolutional layer equals

$$e_{ireg} = w_{(c_f, i_f, j_f)} \cdot \varepsilon_x, \tag{16}$$

where ε_x is the error term added to the faulty activation and $w_{(c_f, i_f, j_f)}$ is the weight multiplied with that activation. This weight is found by its position (c_f, i_f, j_f):

$$c_f = \left\lceil \frac{ts - p_{col} - p_{row}}{H_k \cdot W_k} \right\rceil \tag{17}$$

$$k_f = (ts - p_{col} - p_{row}) \bmod (H_k \cdot W_k) \tag{18}$$

$$i_f = \left\lfloor \frac{k_f}{W_k} \right\rfloor \tag{19}$$

$$j_f = k_f \bmod W_k \tag{20}$$

where ts is the targeted cycle of a systolic array execution, p_{row} and p_{col} are the position of the targeted PE.

Fault results in the bullet error pattern. Therefore, the following parameters define affected output values: output channels $c_{out,f}$ and position in the feature map (u_f, v_f). Affected output channels are calculated as follows:

$$c_{out,f} = \left[c_{out,f}^{start}, c_{out,f}^{end} \right] \tag{21}$$

$$c_{out,f}^{start} = (t_w - 1) \cdot N + p_{col} + 1 \tag{22}$$

$$c_{out,f}^{end} = \begin{cases} t_w \cdot N & \text{if} \quad C_{out} \geqslant t_w \cdot N \\ C_{out} & \text{if} \quad C_{out} < t_w \cdot N \end{cases} \tag{23}$$

where $c_{out,f}$ is a range of affected output channels, starting with $c_{out,f}^{start}$ and ending with $c_{out,f}^{end}$, t_w is the targeted weight tile, p_{row} is the row of the targeted PE, N is the size of the systolic array, and C_{out} is the total number of output channels.

The position (u_f, v_f) is found in the following way:

$$\square_f = (t_a - 1) \cdot N + p_{row} \tag{24}$$

$$u_f = \left\lfloor \frac{\square_f}{W_{out}} \right\rfloor \tag{25}$$

$$v_f = \square_f \bmod W_{out} \tag{26}$$

where $\square_f$ is the affected sliding window and W_{out} is the width of the output feature map.

An error e_{ireg} calculated using Eq. (16) is added to the affected output values.

Fault in WREG. An error added to the output of the convolutional layer in this case equals

$$e_{wreg} = x_{(c_f, i_f, j_f)} \cdot \varepsilon_w \,, \tag{27}$$

where ε_w is the error term added to the faulty weight and $x_{(c_f, i_f, j_f)}$ represents the activations multiplied by that weight.

Fault in the weight register results in the line error pattern, therefore, the following parameters define affected output values: output channel $c_{out,f}$ and a range of positions starting with $(u_f^{start}, v_f^{start})$ and ending with (u_f^{end}, v_f^{end}). The affected output channel is calculated as follows:

$$c_{out,f} = (t_w - 1) \cdot N + p_{col} \,. \tag{28}$$

Positions are calculated from the affected sliding windows following Eq. (25) and (26):

$$\square_f^{start} = (t_a - 1) \cdot N + p_{row} \,, \tag{29}$$

$$\square_f^{end} = \begin{cases} t_a \cdot N & \text{if} \quad W_{out} \cdot H_{out} \geqslant t_a \cdot N \,, \\ W_{out} \cdot H_{out} & \text{if} \quad W_{out} \cdot H_{out} < t_a \cdot N \,, \end{cases} \tag{30}$$

where W_{out} and H_{out} are the width and height of the output feature map.

Faults in OREG. Since those faults result in a single point error pattern, an error added to the output equals

$$e_{oreg} = \varepsilon_y \,. \tag{31}$$

The output channel $c_{out,f}$ is found using Eq. (28), and the position is found using Eq. (24)–(26).

Transient faults in WS dataflow

Fault in IREG. The fault corrupts an activation, hence an error added to the output of the convolutional layer equals

$$e_{ireg} = w_{(c_f, i_f, j_f)} \cdot \varepsilon_x \,, \tag{32}$$

where $w_{(c_f, i_f, j_f)}$ is the weight multiplied with the faulty activation. This weight is found by its position (c_f, i_f, j_f):

$$c_f = \left\lceil \frac{(t_a - 1) \cdot N + p_{row}}{H_k \cdot W_k} \right\rceil \,, \tag{33}$$

$$k_f = \left((t_a - 1) \cdot N + p_{row}\right) \bmod \left(H_k \cdot W_k\right) \,, \tag{34}$$

$$i_f = \left\lfloor \frac{k_f}{W_k} \right\rfloor \,, \tag{35}$$

$$j_f = k_f \bmod W_k \,. \tag{36}$$

Fault results in the bullet error pattern. Therefore, the following parameters define affected output values: output channels $c_{out,f}$ and position in the feature map (u_f, v_f). Affected output channels are calculated as follows:

$$c_{out,f} = \left[c_{out,f}^{start}, c_{out,f}^{end} \right] \,, \tag{37}$$

$$c_{out,f}^{start} = (t_w - 1) \cdot N + p_{col} + 1 \,, \tag{38}$$

$$c_{out,f}^{end} = \begin{cases} t_w \cdot N & \text{if} \quad C_{out} \geqslant t_w \cdot N \\ C_{out} & \text{if} \quad C_{out} < t_w \cdot N \,, \end{cases} \tag{39}$$

where $c_{out,f}$ is a range of affected output channels, starting with $c_{out,f}^{start}$ and ending with $c_{out,f}^{end}$.

The position (u_f, v_f) is found from the sliding window $\square_f$ using Eq. (25) and (26):

$$\square_f = ts - p_{col} - p_{row}\,, \tag{40}$$

where $\square_f$ is the affected sliding window. An error e_{ireg} calculated using Eq. (32) is added to the affected output values.

Fault in WREG. An error added to the output of the convolutional layer in this case equals

$$e_{wreg} = x_{(c_f, i_f, j_f)} \cdot \varepsilon_w\,. \tag{41}$$

Fault results in the partial or whole channel error pattern; therefore, the following parameters define affected output values: output channel $c_{out,f}$ and a range of positions starting with $(u_f^{start}, v_f^{start})$ and ending with (u_f^{end}, v_f^{end}). However, (u_f^{end}, v_f^{end}) is constant; it is the last pixel in the output. The affected output channel is calculated as follows:

$$c_{out,f} = (t_w - 1) \cdot N + p_{col}\,. \tag{42}$$

Positions are calculated from the affected sliding windows following Eq. (25) and (26):

$$\square_f^{start} = (t_a - 1) \cdot N + p_{row}\,, \tag{43}$$

$$\square_f^{end} = W_{out} \cdot H_{out}\,. \tag{44}$$

Fault in OREG. Since those faults result in a point error pattern, an error $e_{oreg} = \varepsilon_y$ is added to a single pixel in the output feature map. The output channel $c_{out,f}$ is found using Eq. (42), and the position is found using Eq. (40), (25), and (26).

Transient faults in IS dataflow

Fault in IREG. Fault results in the bullet error pattern. The weight $w_{(c_f, i_f, j_f)}$ multiplied with the faulty activation is found by its position (c_f, i_f, j_f):

$$c_f = \left\lceil \frac{(t_w - 1) \cdot N + p_{row} + 1}{H_k \cdot W_k} \right\rceil\,, \tag{45}$$

$$k_f = ((t_w - 1) \cdot N + p_{row}) \bmod (H_k \cdot W_k)\,, \tag{46}$$

$$i_f = \left\lfloor \frac{k_f}{W_k} \right\rfloor\,, \tag{47}$$

$$j_f = k_f \bmod W_k\,. \tag{48}$$

Affected output channels are calculated as:

$$c_{out,f} = \left[c_{out,f}^{start}, c_{out,f}^{end} \right]\,, \tag{49}$$

$$c_{out,f}^{start} = ts - p_{col} - p_{row}\,, \tag{50}$$

$$c_{out,f}^{end} = C_{out}\,. \tag{51}$$

The position of the affected output value (u_f, v_f) is found from the sliding window using Eq. (25) and (26):

$$\square_f = (t_a - 1) \cdot N + p_{col}\,. \tag{52}$$

Fault in WREG. Fault results in the line error pattern; therefore, the following parameters define affected output values: output channel $c_{out,f}$ and a range of positions starting

with $(u_f^{start}, v_f^{start})$ and ending with (u_f^{end}, v_f^{end}). The affected output channel is calculated as follows:

$$c_{out,f} = ts - p_{col} - p_{row}\,. \tag{53}$$

Positions are calculated from the affected sliding windows following Eq. (25) and (26):

$$\square_f^{start} = (t_a - 1)\cdot N + p_{col}\,, \tag{54}$$

$$\square_f^{end} = \begin{cases} t_a \cdot N & \text{if} \quad W_{out}\cdot H_{out} \geqslant t_a \cdot N\,, \\ W_{out}\cdot H_{out} & \text{if} \quad W_{out}\cdot H_{out} < t_a \cdot N\,. \end{cases} \tag{55}$$

Fault in OREG. Fault results in a single-point error pattern as well. An error $e_{oreg} = \varepsilon_y$ is added to a single pixel in the output feature map. The output channel $c_{out,f}$ is found using Eq. (53), and the position is found using Eq. (52), (25), and (26).

4.1.2.3 Fault propagation analysis for permanent faults

Permanent faults are present during the whole execution and are therefore defined by four parameters (Table 4).

Table 4: Permanent fault parameters

Symbol	Description
f_{type}	Type of fault (IREG, WREG, OREG)
p_{row}	Row of the PE
p_{col}	Column of the PE
β_f	Bit index

Permanent faults result in several error patterns, one for each step. Therefore, the affected output values are defined by sets of parameters. E.g., in the case of permanent faults in the OS IREG, the affected output values are defined by a set of output channels

$$\left\{ (c_{out,f}^{start}, c_{out,f}^{end})_i \mid i \in \{1,...,T_w^{os}\} \right\} \tag{56}$$

and a set of positions

$$\left\{ (u_f, v_f)_i \mid i \in \{1,...,T_a^{os}\} \right\}. \tag{57}$$

A set of output channels is found by substituting t_w with an iterator in Eq. (22) and (23):

$$c_{out,f(i)}^{start} = (i-1)\cdot N + p_{col} + 1 \quad \forall\, i \in \{1,...,T_w^{os}\} \tag{58}$$

$$c_{out,f(i)}^{end} = \begin{cases} i\cdot N & \text{if} \quad C_{out} \geqslant i\cdot N \\ C_{out} & \text{if} \quad C_{out} < i\cdot N \end{cases} \qquad \forall\, i \in \{1,...,T_w^{os}\} \tag{59}$$

which gives a pair $(c_{out,f}^{start}, c_{out,f}^{end})$ for every weight tile.

A set of positions is found by substituting t_a with an iterator in Eq. (24) for the affected sliding windows and then using Eq. (25) and (26) to calculate position from the sliding window:

$$\square_{f(i)} = (i-1)\cdot N + p_{row} \quad \forall\, i \in \{1,...,T_a^{os}\} \tag{60}$$

$$u_{f(i)} = \left\lfloor \frac{\square_{f(i)}}{W_{out}} \right\rfloor \tag{61}$$

$$v_{f(i)} = \square_{f(i)} \bmod W_{out} \tag{62}$$

which gives a position for every input tile. This way, every bullet pattern is characterized by a Cartesian product of those two sets.

An error added to the output for each bullet pattern is the cumulative value

$$e^{p}_{ireg} = \sum_{c}^{C_{in}} \sum_{i}^{H_k} \sum_{j}^{W_k} w_{(c,i,j)} \cdot \varepsilon_{(c,u_f+i,v_f+j)} \tag{63}$$

where $\varepsilon_{(c,u_f+i,v_f+j)}$ is the error term added to input $x_{(c,u_f+i,v_f+j)}$. Since permanent faults are modeled as stuck-at-0 or stuck-at-1 faults, an error term (for the signed integer) is expressed as

$$\varepsilon = \begin{cases} 2^{\beta_f} \cdot \gamma & \text{if} \quad x_f^{(\beta_f)} \neq s \\ 0 & \text{if} \quad x_f^{(\beta_f)} = s \end{cases} \tag{64}$$

where s is the stuck-at state (0 or 1).

4.1.2.4 Implementation

The workflow of the proposed reliability assessment method, combining fault injection with fault propagation analysis, is presented in Figure 9. SAFPA is implemented in Python, and PyTorch is used to run the inference of DNNs, leveraging GPU speedup. To speed up permanent fault injection, error calculations are implemented in C. The fault injection module gets the output of the targeted convolutional layer and modifies only affected output values according to the presented equations. The inference continues with the erroneous values. SAFPA is available open-source at `https://github.com/ntlchr/safpa`.

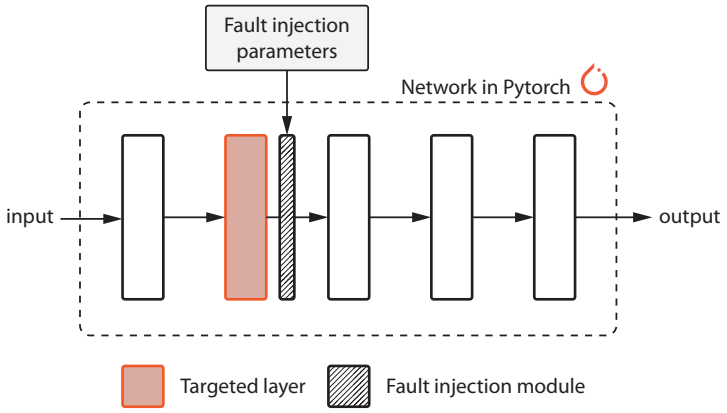


Figure 9: Workflow of the fault injection process

4.1.3 Results

4.1.3.1 Evaluation

SAFPA is compared with the representative fault-injection framework that models the systolic array microarchitecture, Saca-FI [184]. While previous works rely on cycle-accurate simulation of a systolic array, which is a very time-consuming task, the proposed method is based on fault propagation analysis. Table 5 presents the comparison of two methods. Runtime denotes the processing time of one input with fault injection for transient and permanent faults. For runtime comparison, the selected model is VGG-16 evaluated on ImageNet (Saca-FI supports only three DNNs, the largest one was selected), the SA

size is 256×256, and a fault is injected in the first convolutional layer. Experiments were performed on an NVIDIA A100 GPU. Both methods give Architecture Vulnerability Factor (AVF) as an output, the probability that a fault in the hardware structure causes an application output error [130]. Numerically, it is the percentage of the hardware faults that propagate to the software output, and is calculated as the ratio of output errors to the number of injected faults [148]. While the functionality of the methods is similar, SAFPA is several orders of magnitude faster than microarchitecture-level fault injection.

Table 5: Comparison of reliability assessment methods

Work	Fault model	Output	Dataflow	Runtime*
Saca-FI	Transient and permanent faults in registers	AVF	IS, WS, OS	37 s / 88 s
SAFPA (ours)	Transient and permanent faults in registers	AVF	IS, WS, OS	0.0006 s / 0.001 s

* Transient fault injection / permanent fault injection.

4.1.3.2 Fault injection

ResNet-18 trained on ImageNet was selected for an example of reliability evaluation. Figure 10 presents the results of the reliability assessment of ResNet-18 considering different dataflows and systolic array sizes (32×32 and 128×128) and transient faults. Results show that, for the selected model, WS dataflow has, on average, higher vulnerability with a smaller SA size, while OS dataflow has higher vulnerability with a larger SA size. Figure 11 presents error rates of PE registers for each convolutional layer (128×128 systolic array). It can be seen that, on average, in IS and WS systolic arrays, registers holding stationary values lead to a higher percentage of output errors. For the OS systolic array, IREG is the most vulnerable register.

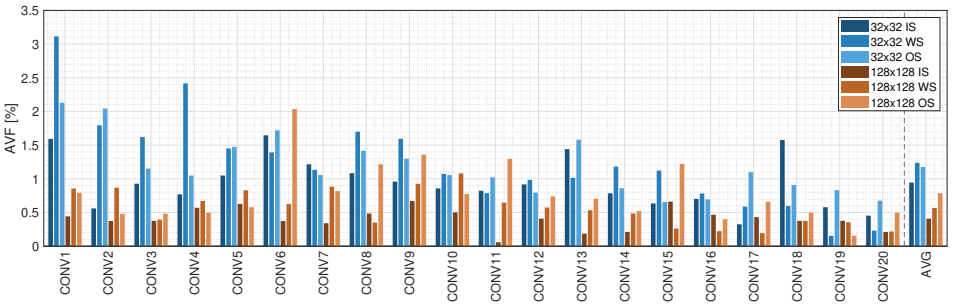


Figure 10: Reliability assessment of ResNet-18 on different systolic arrays considering transient faults

4.2 Reliability-Aware Reconfiguration for Systolic Arrays

4.2.1 Introduction

This section presents a design space exploration of reliability-performance trade-offs in CNNs deployed on Reconfigurable Systolic Array (RSA) accelerators. With a comprehensive reliability evaluation enabled by the SAFPA framework, the section proposes a layer-wise reconfiguration strategy for RSAs to balance reliability and performance. RSAs are proposed to optimize system performance and energy consumption [105, 142, 217]. How-

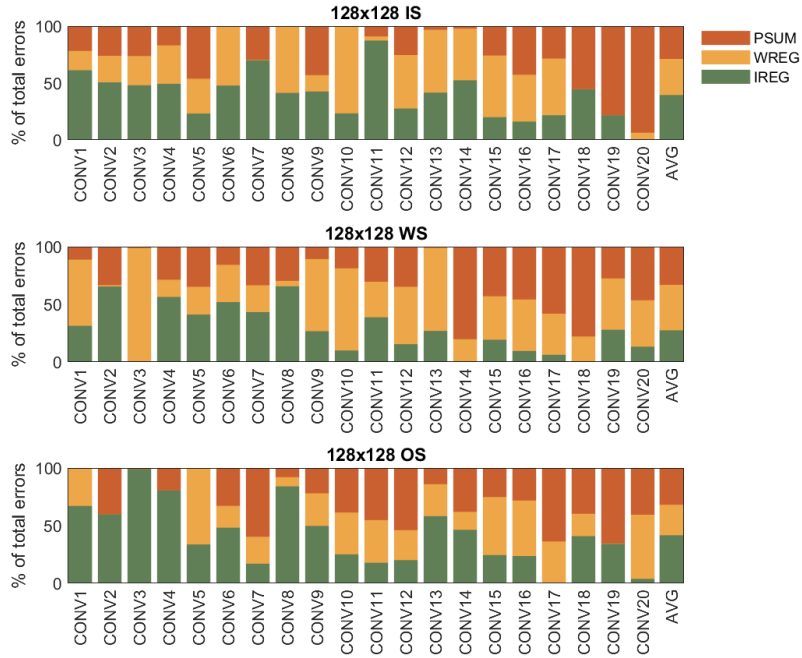


Figure 11: Layer-wise fault distribution for ResNet-18

ever, the reliability of reconfigurable systolic arrays has not been previously studied in the literature. This section studies CNNs deployed on a layer-wise reconfigurable systolic array. The objective is to evaluate the reliability of CNN layers under transient faults in RSA registers across different systolic array sizes, along with their performance and utilization. Furthermore, a resizing strategy is proposed to identify the optimal systolic array size for each layer, considering both reliability and performance.

4.2.2 Methodology

4.2.2.1 RSA Architecture

The architecture of the RSA considered in this work is presented in Figure 12. The RSA is based on the output-stationary dataflow, in which weights propagate through the SA from top to bottom, activations propagate from left to right, and partial sums (outputs) are kept stationary in the PEs. Before the execution of a layer, a set of PEs is activated by the controller, and only they perform the operations of that layer of a CNN. The unused PEs are clock-gated to optimize energy consumption. The outputs of the layer are captured from the determined subset of PEs. In this work, the RSA consists of 256×256 PEs, and it can be reconfigured into four sizes: 32×32, 64×64, 128×128, and 256×256. Before inference, the system's configuration for each layer is determined based on offline profiling, and the RSA is reconfigured at runtime with negligible performance overhead.

4.2.2.2 Reliability and Performance Analysis for the RSA

To analyze the reliability of the RSA, an extensive fault-injection experiment is conducted to obtain the AVF for each CNN layer across each systolic array configuration. Fault injection is done using the SAFPA framework presented earlier (Section 4.1). Statistical layer-

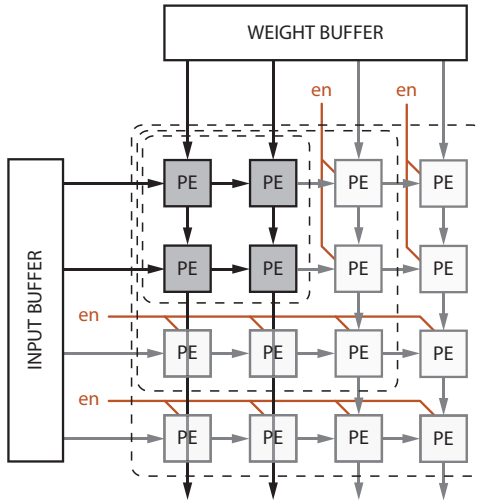


Figure 12: Architecture of the reconfigurable systolic array

wise fault injection is performed, with over 1000 injected faults per convolutional layer, ensuring 95% confidence and 5% error margin [110]. Faults are injected only in used PEs.

To analyze the performance, the execution time T of each layer i on each systolic array configuration j is profiled, as well as the utilization U of the PEs in each configuration by the CNN layer. To evaluate the performance, the efficiency metric E is considered:

$$E_{size_j}^{layer_i} = \frac{U_{size_j}^{layer_i}}{T_{size_j}^{layer_i}} \tag{65}$$

The efficiency metric combines the utilization and execution time of the systolic array configuration for a single layer. A higher value indicates that a systolic array configuration can achieve faster execution while using as many active PEs as possible. To calculate efficiency, the execution time of all layers of a CNN is normalized to the longest execution time across all configurations. Also, the obtained E value is normalized to the maximum value across the CNN.

4.2.2.3 Resizing Strategies

To determine the RSA configuration, first, an extensive layer-wise reliability and performance analysis of all RSA configurations for each layer of the target CNNs is conducted. Based on the obtained AVF and efficiency E values, the following strategies for layer-wise reconfiguration of the RSA are considered:

- **Constant Configuration**: Two constant configurations for the RSA, based on the minimum (RSA32) and maximum (RSA256) available configurations, are considered to demonstrate the advantages of the reconfigurability of RSA in terms of reliability and performance.
- **Best Reliability (BR)**: The RSA size for each layer is determined by the configuration, which implies minimum AVF.
- **Best Performance (BP)**: The RSA size for each layer is determined by the configuration, yielding the maximum performance efficiency according to Eq. (65).

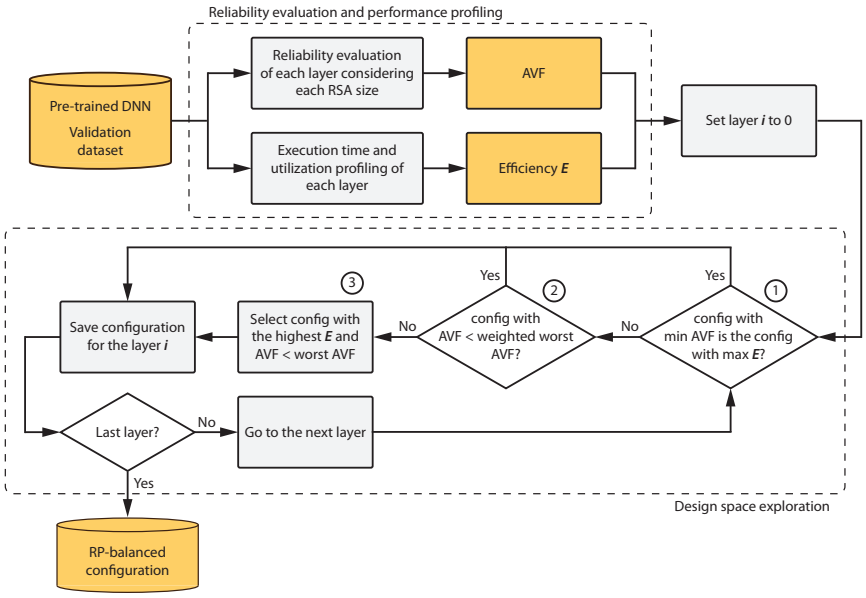


Figure 13: Proposed algorithm to determine the RSA configuration for each layer by balancing reliability and performance (RP strategy)

- **Reliability-Performance Balance (RP)**: The RSA size for each layer is determined based on the proposed algorithm in Figure 13, to balance the reliability and performance of the systolic array. The AVF and efficiency E for all configurations of the RSA for each layer are explored, and the RSA size for a layer is selected, which: 1) offers the best AVF and E, or 2) holds an AVF lower than the weighted worst AVF, or 3) have the highest E as long as it does not have the worst AVF.

In the experiments, the weight coefficient in the *RP* strategy is set to 0.6, based on an exploration of AVF values across different layers and RSA configurations.

4.2.3 Experimental Results

Three representative CNNs are selected for analysis: VGG-19, ResNet-18, and ResNet-50, all trained on the ImageNet dataset. All networks use a 32-bit floating-point data type.

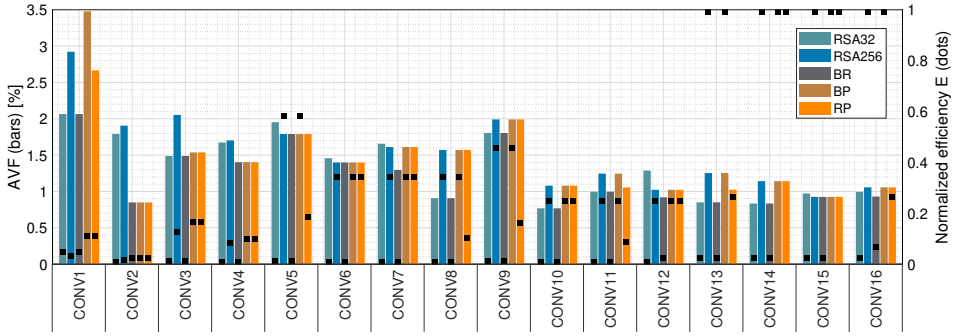


Figure 14: Layer-wise AVF and efficiency E comparison for VGG-19, with different RSA configuration strategies

Table 6: Systolic array size selected for each layer by each strategy for VGG-19

Layer	RSA32	RSA256	BR	BP	RP
1	32×32	256×256	32×32	64×64	128×128
2	32×32	256×256	64×64	64×64	64×64
3	32×32	256×256	32×32	128×128	128×128
4	32×32	256×256	128×128	128×128	128×128
5	32×32	256×256	256×256	256×256	256×256
6	32×32	256×256	256×256	256×256	256×256
7	32×32	256×256	128×128	256×256	256×256
8	32×32	256×256	32×32	256×256	256×256
9	32×32	256×256	32×32	256×256	256×256
10	32×32	256×256	32×32	256×256	256×256
11	32×32	256×256	32×32	256×256	128×128
12	32×32	256×256	64×64	256×256	256×256
13	32×32	256×256	32×32	256×256	128×128
14	32×32	256×256	32×32	256×256	256×256
15	32×32	256×256	256×256	256×256	256×256
16	32×32	256×256	64×64	256×256	256×256
Average	32×32	256×256	92×92	216×216	204×204

Figure 14 presents a layer-wise comparison of reliability and performance between different RSA reconfiguration strategies for VGG-19. Table 6 shows the RSA size selected for each layer by different strategies. The following observations can be highlighted:

- Reconfiguring the RSA for each layer provides an opportunity to optimize the reliability or performance of the CNNs deployment on systolic arrays. Comparing the results for the constant configurations (*RSA32* and *RSA256*) with those for the reconfigured configurations (*BR*, *BP*, and *PR*) shows gains in either reliability or performance.

- Based on the reconfiguration strategy, the RSA can provide the best reliability (*BR* strategy) or performance (*BP* strategy) in all layers. As observed, the AVF across all layers in the *BR* strategy is the lowest; however, its efficiency E is very low in many cases. Instead, in the *BP* strategy, efficiency is maximized across all layers; however, the AVF is high in many cases. This observation underscores the need to present the *RP* strategy.

- The proposed *RP* strategy yields configurations that balance reliability and performance. As observed, the AVF of *RP* in all layers is always equal to or less than that in the *BP* strategy, and the efficiency E in all layers is always equal to or more than that in the *BR* strategy.

- In terms of RSA size, it is observed that the *RP* strategy utilizes an average SA size of 204×204 PEs to balance the inference reliability and performance, which is smaller than that of *BP* and larger than that of *BR*.

Furthermore, the reliability and performance of the selected CNN models are presented to compare the impact of the strategies. Figure 15 shows average AVF and efficiency E for three CNN models deployed on RSA using five strategies. As observed, layer-wise reconfiguration strategies can effectively lead to a more reliable or performant deployment. On the other hand, in all cases, the *RP* strategy achieves a lower AVF (5.5% to

8.3% more reliable) than the *BP* strategy and higher performance efficiency than the *BR* strategy (1.6× up to 14.7× more performant). These results highlight that *RP* can successfully balance reliability and performance of CNN deployment on RSAs.

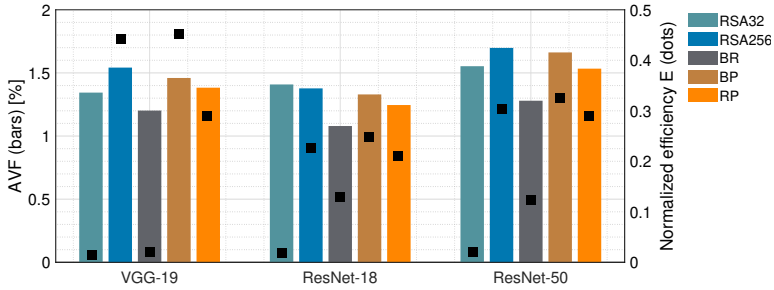


Figure 15: Model-wise AVF and efficiency E comparison with different RSA configuration strategies

4.3 Reliability Assessment of Vision Transformers on Systolic Arrays via Fault Propagation Analysis

4.3.1 Introduction

Vision Transformers, which outperform traditional Convolutional Neural Networks in computer vision tasks, have gained significant popularity recently. Since CNNs were the predominant solutions for computer vision tasks before, their reliability has been thoroughly explored. The reliability aspects of ViTs, in comparison, are less explored.

Several works have been published on the reliability of Vision Transformers. The vulnerability of ViTs to bit-flips in the outputs of the major arithmetic operations, i.e., multiplication and addition, is explored in [214]. ViT sensitivity to faults in weights is analyzed in [72]. The vulnerability of the Swin Transformer model to stuck-at faults in weights and bit-flips in the output neurons is evaluated in [60]. The work also proposes a fault-tolerance solution based on the ranging of the outputs, inputs, and weights of the fully connected layers. However, those works perform analysis at the model level, without considering the hardware platform.

A reliability analysis of ViTs considering faults in the GPU is performed in [153]. The evaluation is based on data collected through neutron beam experiments and software fault simulations. [49] and [11] also explore the reliability of ViTs on a GPU under neutron radiation and compare their reliability with that of CNNs. [11] additionally explores how runtime optimization techniques, data precision reduction, and quantization affect the reliability of the selected models. [121] presents a reliability evaluation of ViTs on Coral TPU under neutron radiation.

Another work [163] presents a reliability analysis of the text embeddings considering different tasks and datasets. The analysis is performed at the model level as well.

Compared to the existing works, a novel reliability assessment method for ViT inference on systolic arrays based on fault propagation analysis is proposed. This method is used to perform fine-grained reliability analysis of three common ViT models for image classification. Their vulnerability to soft errors in systolic arrays is investigated, focusing on different models, layers, and operations. The reliability of ViTs is compared with a representative CNN model, namely ResNet-50.

4.3.2 Methodology

The proposed reliability assessment method is an extension of the method for CNNs presented in Subsection 4.1 for Vision Transformers. *This method combines fault injection with fault propagation analysis, which is used to calculate the resulting error in the operation output*, instead of performing time-consuming cycle-accurate simulations using a microarchitectural or RTL model of a systolic array. This allows for fast and scalable assessment. As was shown previously, the method is several orders of magnitude faster than microarchitecture-level fault injection.

The method described beforehand was developed for convolutional layers. However, all major arithmetic operations in ViTs are represented by matrix-matrix multiplication, whether it involves multiplying two previously calculated values or multiplying a previously calculated value by a weight. Fault propagation in matrix multiplication is different compared to convolution. Therefore, an extension to support ViTs was developed based on new error patterns and new equations for their calculation. It is available open-source at `https://github.com/ntlchr/safpa-vit`.

4.3.2.1 Fault Propagation in Systolic Arrays for Matrix Multiplication

The accelerators for ViTs proposed in the literature [28, 174] are based on output-stationary systolic arrays, since they allow high input and output reuse. In the output-stationary systolic arrays, weights and inputs propagate through the array, and outputs are accumulated inside the PEs. For this reason, the analysis in this work is performed for the output-stationary systolic arrays.

Single transient faults (bit flips) in PE registers holding inputs (activations), weights, and partial sums (outputs) are considered for the analysis, following previous works [184]. Due to the regular structure of a systolic array and the propagation of the data, faults in different registers result in distinct error patterns in the output of the matrix multiplication. The error patterns were presented in [205]. A fault in a weight register (WREG) results in the *vertical line* error pattern, since a faulty weight propagates through the entire column and corrupts several pixels in the output. A fault in the input register (IREG) results in the *horizontal line* pattern, since faulty activation propagates through the entire row. Finally, a fault in the output register (OREG) holding partial sum results in the *point pattern*, since output values are stationary.

4.3.2.2 Fault Propagation Analysis

Single transient faults in registers are considered for the analysis. Therefore, each fault is defined by seven parameters given in Table 3: fault type f_{type}, execution cycle ts, weight tile t_w, activation tile t_a, PE row and column p_{row} and p_{col}, and bit index β_f.

A bit flip in the numerical value x can be expressed as an addition of the error term ε. For this study, a 32-bit floating-point arithmetic is considered. Therefore, an error term is defined following Eq. (14).

The result of the matrix multiplication is the sum of the element-wise multiplication between rows of the first input and columns of the second input. *A fault in one of the components results in the addition of error e to one or several values in the output.* Multiplying input matrix $X \in \mathbb{R}^{P \times S}$ and weight matrix $W \in \mathbb{R}^{S \times R}$, we get output matrix $Y \in \mathbb{R}^{P \times R}$.

Fault in input register (IREG). If the fault happens in the input register, it corrupts an activation value coming from the side and results in the horizontal line error pattern. Therefore, an error e_{ireg} is added to several output values in one row. The affected output values are defined by the row of the output matrix and the range of columns.

An error added to the output equals $e_{ireg} = w_{(u_f,v_f)} \cdot \varepsilon_x$, where $w_{(u_f,v_f)}$ is the weight multiplied with the faulty activation, and ε_x is an error term. The faulty activation is found by its position (i_f, j_f):

$$i_f = (t_a - 1) \cdot N + p_{row} \tag{66}$$

$$j_f = ts - p_{col} - p_{row} \tag{67}$$

where t_a is the targeted activation tile, N is the size of the systolic array, ts is the cycle of a systolic array execution, p_{row} and p_{col} denote the position of the targeted PE.

The weight is found by its position (u_f, v_f):

$$u_f = j_f \tag{68}$$

$$v_f = \left[v_f^{start}, v_f^{end} \right] \tag{69}$$

$$v_f^{start} = (t_w - 1) \cdot N + p_{col} + 1 \tag{70}$$

$$v_f^{end} = \begin{cases} t_w \cdot N & \text{if} \quad R \geqslant t_w \cdot N \\ R & \text{if} \quad R < t_w \cdot N \end{cases} \tag{71}$$

where $v_{out,f}$ is a range of affected columns, starting with v_f^{start} and ending with v_f^{end}, t_w is the targeted weight tile, and R is the width of the output matrix.

The affected output row is found using Eq. (66) and the range of columns is found using Eq. (69)–(71).

Fault in weight register (WREG). In this case, a weight value coming from the top is corrupted, resulting in the vertical line error pattern. Therefore, an error e_{wreg} is added to the column of the output matrix. The affected output values are defined by the range of rows and one column.

An error added to the output in this case equals

$$e_{wreg} = x_{(i_f,j_f)} \cdot \varepsilon_w \,, \tag{72}$$

where $x_{(i_f,j_f)}$ is the activation multiplied by the faulty weight. The activation is found by its position (i_f, j_f):

$$i_f = \left[i_f^{start}, i_f^{end} \right] \tag{73}$$

$$i_f^{start} = (t_a - 1) \cdot N + p_{row} \tag{74}$$

$$i_f^{end} = \begin{cases} t_a \cdot N & \text{if} \quad P \geqslant t_a \cdot N \\ P & \text{if} \quad P < t_a \cdot N \end{cases} \tag{75}$$

$$j_f = ts - p_{col} - p_{row} \tag{76}$$

where P is the height of the output matrix.

The faulty weight is found by its position (u_f, v_f):

$$u_f = (t_w - 1) \cdot N + p_{col} \tag{77}$$

$$v_f = j_f \tag{78}$$

The affected output rows are found using Eq. (73)–(75) and the column is found using Eq. (77).

Fault in the output register (OREG). A fault in the output register results in the point error pattern. An error e_{oreg} is added to a single value in the output matrix. Its position is defined by the row and column. An error added to the output equals

$$e_{oreg} = \varepsilon_y \,, \tag{79}$$

Table 7: Parameters of the tested vision transformers

Model	Dataset	SA size	Input size	Patch size	Embedding size	# of layers	# of heads	MLP dim
ViT-B16	CIFAR-100	32×32	224×224*	16×16	768	12	12	3072
ViT-B16	Imagenet	128×128	224×224	16×16	768	12	12	3072
DeiT-S	Imagenet	128×128	224×224	16×16	384	12	6	1536
Swin-T	Imagenet	128×128	224×224	4×4	96	(2,2,6,2)	(3,6,12,24)	384

* CIFAR-100 images were upscaled to 224×224.

where ε_y is an error term added to the faulty partial sum. The row is found using Eq. (66), the column is found using Eq. (77).

4.3.3 Experimental setup

Three common vision transformer models were selected for the experiments, including the original ViT-B16 [50], DeiT-S [188], and Swin-T [118]. ResNet-50 was selected as an example of CNNs. The parameters of the tested ViT models are presented in Table 7. ResNet-50 is trained on Imagenet1K and tested considering a 128×128 systolic array. The fault injection framework is implemented in Python using PyTorch. Experiments were performed on an NVIDIA A100 GPU.

AVF was used as a reliability measure. As was stated earlier, AVF is the ratio of output errors to the number of injected faults, and since the output of the selected models is the probability score of each class, the following output errors of the DNNs were considered [184]:

- Top1-class: the top-ranked class is different from the golden run.
- Top1-acc: the probability score of the top-ranked class is different. The top-ranked class may be different. Top1-acc includes the Top1-class errors.
- Top5-class: at least one class in the top-5 is different, including different order of the top-5 classes.
- Top5-acc: the probability score of at least one class in the top-5 is different. The top-5 classes may be different. Top5-acc includes Top1-class, Top1-acc and Top5-class errors.

Layer-wise and operation-wise statistical fault injection was performed using the equation introduced in [110] to achieve 95% confidence and 5% error margin. For each fault injection, fault parameters (Table 3) were set randomly, and a test dataset of 10 000 inputs was fed to the network. Faults were injected in each layer and operation that can be mapped to a systolic array, following the presented methodology (Subsection 4.3.2.2).

4.3.4 Empirical analysis

4.3.4.1 Operation-dependent vulnerability pattern

Figures 16 and 17 present AVF values for each layer and operation of selected transformer models. Figure 18 presents AVF values for each convolutional layer in ResNet-50. For the sake of space, all four output error categories are presented only for the ViT-B16 model on a 128×128 systolic array. For the other models, only Top1-class AVF values are presented.

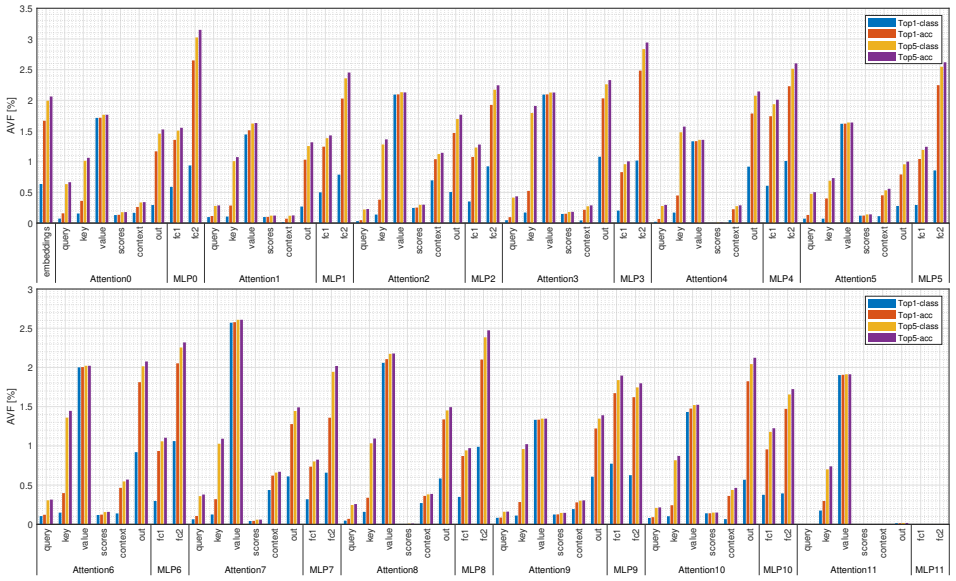


Figure 16: Layer- and operation-wise vulnerability assessment of ViT-B16 considering transient faults (128×128 systolic array)

ResNet-50 (Figure 18) shows high variations in vulnerability across the layers, common for CNNs. In ViTs, on the other hand, certain operations, namely value, context, MHA output, and both FC layers, consistently show high vulnerability. It can also be observed that the variations in the operations AVFs across layers (encoder blocks) are low, except for the Swin-T model. *It shows that while the reliability of CNNs is defined by the position of the layer, the reliability of ViTs is defined by the type of operation*. This means that, unlike reliability solutions for CNNs that proposed different levels of protection for different layers based on their vulnerability, the fault tolerance solution for ViTs should propose different levels of protection for different operations. Potentially by deploying several systolic arrays, one for performance, one for fault tolerance, since several operations in the MHA block can be performed in parallel, thus absorbing the timing overhead that can be introduced by fault tolerance.

4.3.4.2 Significant criticality of the value matrix

Figure 16 shows that, while for most operations the vulnerability factors for Top1-acc, Top5-class, and Top5-acc errors are noticeably higher than those for Top1-class errors, for value calculations, they are always similar. It means that if a fault in value calculation affects the top-5 predictions, it almost always results in misclassification. For other operations, faults more often affect the probability score of the top-5 classes. The exceptional criticality of the value is confirmed by Figure 19 showing the average Top1-class AVF for each operation in an encoder block. *The vulnerability of value is significantly higher than the vulnerability of other operations*. This can be explained by the fact that the faults in query and key computations propagate into the attention score (see Eq. (3) on page 23) and then get attenuated by the softmax normalization (see Eq. (4) on page 23), while faults in the value matrix pass through linearly (see Eq. (5) on page 23). Moreover, while systolic array size and model size affect the vulnerability of other operations, the high criticality of the value matrix persists.



Figure 17: Layer- and operation-wise vulnerability assessment of tested vision transformers considering transient faults

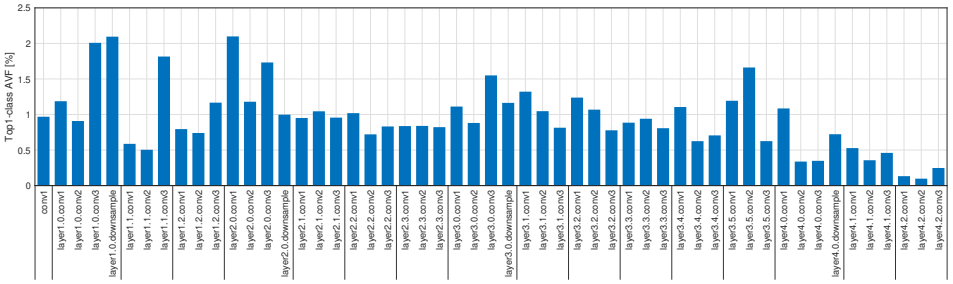


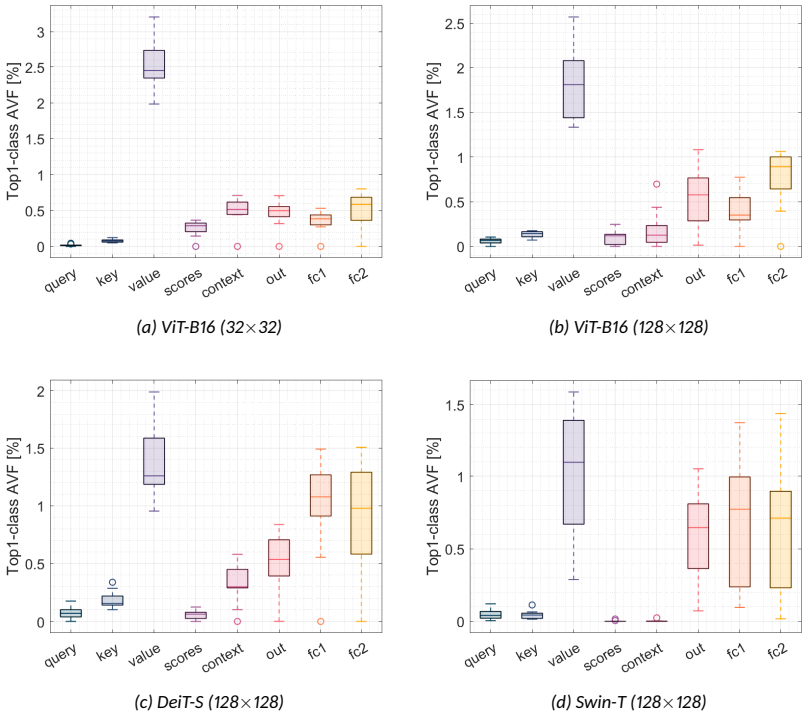
Figure 18: Layer-wise vulnerability assessment of ResNet-50 considering transient faults (128×128 systolic array)

4.3.4.3 Reliability of embeddings

Figure 20 presents the average Top1-class AVF for embeddings and each encoder block in the networks. Contrary to some previous works [60, 121] that report patch embedding as the most critical component of vision transformers, it can be seen that embedding vulnerability is comparable to the average AVF of encoder blocks. Detailed results (Figures 16 and 17) show that certain encoder operations are more critical.

4.3.4.4 Effect of architectural regularity on reliability

From Figures 17 and 20, it can be seen that for the Swin-T model, the variation in AVF values across layers (encoder blocks) is high, resembling CNN behavior (e.g., ResNet-50 on Figure 18). Models like ViT and DeiT have the same dimensions across the layers. Conversely, in Swin transformers, due to the smaller patches merging into bigger ones in the deeper layers and to the increase in the number of heads in MHA, dimensions change across the layers, similar to CNNs. This change in size might explain higher variations in



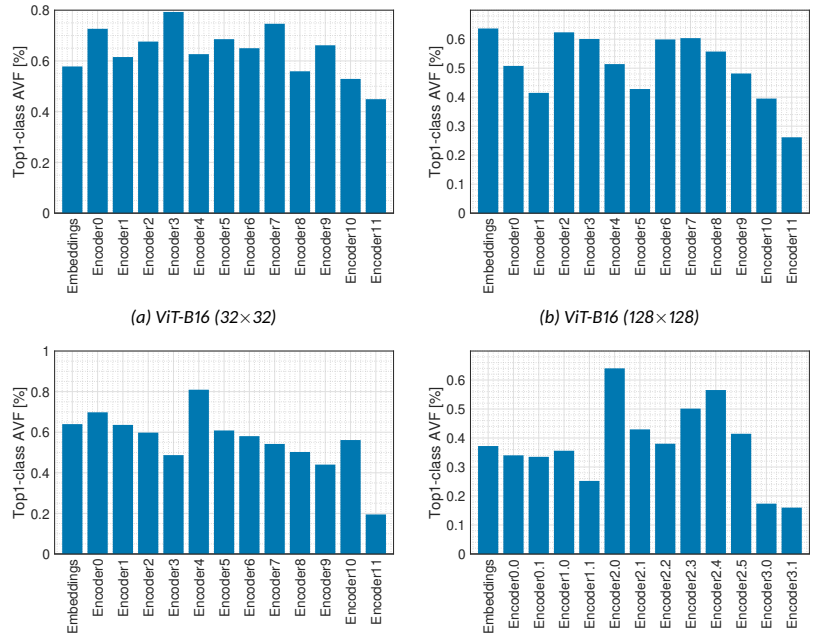
(a) ViT-B16 (32×32)

(b) ViT-B16 (128×128)

(c) DeiT-S (128×128)

(d) Swin-T (128×128)

Figure 19: Layer- and operation-wise vulnerability comparison



(a) ViT-B16 (32×32)

(b) ViT-B16 (128×128)

(c) DeiT-S (128×128)

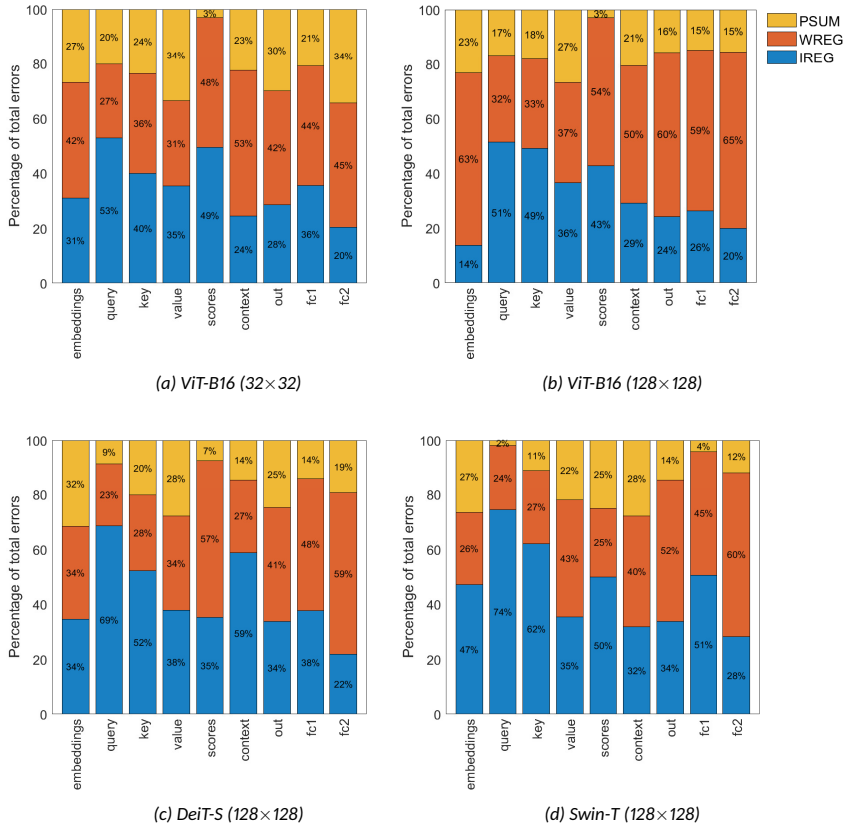
(d) Swin-T (128×128)

Figure 20: Embeddings and encoder blocks vulnerability

AVF values. *This shows that architectural regularity has an impact on model vulnerability across the layers*. Single-resolution models like ViT and DeiT show more uniform reliability across the layers. Hierarchical models like Swin show high variation across the layers, similar to CNNs.

4.3.4.5 Fault distribution

Figure 21 presents average error rates of PE registers for each operation in an encoder block. For the attention score, the query matrix goes through the input pipeline, and the transposed key matrix goes through the weight pipeline. For the context, the similarity score is considered as input, and the value matrix as weights. It can be noticed that for query and key, faults in the input register, corrupting several values in one patch embedding, have a higher error rate. For the other operations, predominantly faults in the weight register, corrupting the same value in several patch embeddings, have a higher error rate. Noticeably, value has similar error rates for all registers.



(a) ViT-B16 (32×32) *(b) ViT-B16 (128×128)*

(c) DeiT-S (128×128) *(d) Swin-T (128×128)*

Figure 21: Layer- and operation-wise fault distribution

4.4 Chapter conclusions

This chapter presents the fast analytical reliability assessment method for DNN inference on systolic arrays (SAFPA). The method effectively combines fault injection with fault propagation analysis. Instead of injecting faults into the microarchitectural or RTL model of a

systolic array, which is very time-consuming, errors are added directly to the DNN layer output using fault propagation analysis. SAFPA has been shown to be several orders of magnitude faster than a microarchitecture-level fault-injection framework. The method allows an analysis of both transient and permanent faults.

The method is used to analyze the reliability of modern DNN architectures for computer vision tasks, including Convolutional Neural Networks and Vision Transformers. Specifically, the chapter presents a fine-grained reliability analysis of three representative ViT models for image classification, namely ViT-B16, DeiT-S, and Swin-T. Their vulnerability to soft errors in systolic array registers is investigated, with a focus on different models, layers, and operations. We also compare the reliability of ViTs to that of the ResNet-50 CNN. An empirical analysis shows that ViTs exhibit an operation-dependent vulnerability pattern, with significant criticality of value matrix calculation. It also shows the effect of architectural regularity on reliability, specifically that hierarchical transformer models like Swin exhibit high vulnerability variation across the layers, resembling the behavior of CNNs.

Moreover, SAFPA enables a design space exploration of reliability-performance tradeoffs in CNNs deployed on systolic arrays with reconfigurable size. Based on the extensive reliability evaluation of selected CNN models, a layer-wise reconfiguration strategy for RSAs is proposed to balance reliability and performance. The proposed algorithm allows identifying the optimal systolic array size for each layer.

5 FORTALESA: Fault-Tolerant Reconfigurable Systolic Array for DNN Inference

This chapter presents a run-time reconfigurable systolic array architecture, FORTALESA, that tackles the balance between reliability and performance through different execution modes. The fault-tolerance features at the hardware architecture layer are intended to enhance application reliability while balancing performance by considering application characteristics, thereby enabling a cross-layer approach for efficient fault tolerance. DNN inference is considered as a target application. Since different DNN components have different vulnerability levels, they might benefit from heterogeneous mapping to different execution modes of the proposed systolic array. The proposed architecture efficiently protects both sequential (registers) and combinational components of systolic array processing elements. While the main objective is to address transient faults (soft errors), the proposed architecture can also handle permanent faults, enabling graceful degradation. The reconfigurability feature of FORTALESA enables a speedup of up to 3×, depending on layer vulnerability, and requires 6× fewer resources compared to static redundancy and 2.5× fewer resources than the previously proposed solution for transient faults. The chapter is based on publication [VII]:

> N. Cherezova, A. Jutman, and M. Jenihhin. FORTALESA: Fault-tolerant reconfigurable systolic array for DNN inference. *Microprocessors and Microsystems*, 119:105222, 12 2025

5.1 Introduction

A reconfigurable redundancy approach is common in multicore processors, e.g., ARM's Triple Core Lock-Step (TCLS) system [88] and RISC-V-based multicore processing cluster with on-demand redundancy for an efficient performance vs. reliability trade-off [151]. Such solutions allow the use of individual cores separately to run different tasks for high performance or as a group to run the same task for high reliability. Execution mode can be reconfigured during run-time, thus allowing the computing system to adapt to changing conditions and task requirements.

While there are several ways to ensure the reliability of DNN inference, the most common approaches imply redundancy, fault-aware training, and algorithm-based fault tolerance (ABFT). Redundancy indicates replication of the components working in parallel (spatial redundancy) or re-execution of the same operation (temporal redundancy). Since redundancy incurs high overhead, the goal is to find a trade-off between reliability and performance by identifying and protecting more vulnerable components. To tackle this challenge, prior works have proposed selective replication of the layers [20, 115], of the channels within a layer [17], of the weights [95], or neurons [5]. Another common approach involves retraining the DNN with added faults to improve the network's resilience. Fault-aware training was used to address faults in FPGAs [219], timing [44], and computational [212] errors in accelerators. ABFT methods consider certain features of the application for fault detection and correction. Proposed solutions include checksums [137, 138], bounded activation functions [74], restricted output ranges [31] and quantile shifts [61], and AN codes [64].

Systolic arrays were first introduced in the 1980s [101] and recently became popular for accelerating DNNs. As the reliability of DNN accelerators comes to the front, recent works have proposed methods for the assessment and enhancement of systolic arrays. Reliability enhancement methods for systolic arrays, similarly to DNNs, include ABFT and

network retraining. [224] presented a permanent fault correction technique by introducing a re-computing unit that recalculates the computations assigned to faulty processing elements. [108] addressed permanent faults through bypassing faulty units and assigning their computations to the redundant ones. Similarly, [106] used pruning of faulty processing elements for permanent fault correction. [221] proposed a fault-aware pruning plus retraining (FAP+T) technique to address manufacturing defects in the systolic array-based accelerators. It requires modification of the target DNN architecture through pruning and retraining to adapt it to a specific defect in the chip. [114] presented an ABFT technique for permanent fault detection in systolic arrays on FPGAs. Permanent faults in configuration memory are considered in that work. Another permanent fault detection approach assuming faults in registers and MAC units is presented in [193]. The approach is based on checksum calculations and was implemented on an FPGA as well. [41] presented a fault tolerance solution for an FPGA-based systolic array accelerator. Faults in configuration memory that affect the systolic array datapath are detected using checksums and corrected via dynamic partial reconfiguration of the FPGA. [141, 143] presented an ABFT methodology specifically developed for sparse systolic arrays, considering their unique characteristics. While most works address permanent faults, [159] proposed an ABFT method for detecting timing faults due to reduced voltage. Several works proposed test architectures for permanent and transition-delay fault detection in low-power accelerators based on systolic arrays [81, 107]. Compared to the aforementioned works that mostly address the detection of permanent faults, the proposed reconfigurable systolic array architecture addresses both permanent and transient faults, enables a reliability-performance trade-off, provides fault correction in both fault-tolerant execution modes, and does not require retraining of a DNN for each particular chip.

Reconfigurable DNN accelerators based on a systolic array have been proposed before: to improve energy efficiency [217], to decrease inference latency [105], and power consumption [142], to boost transformer models performance [223]. In contrast to the previous works, the proposed reconfigurable systolic array architecture improves the reliability of the DNN inference while balancing performance.

5.2 Fault-tolerant reconfigurable systolic array

We propose FORTALESA, a systolic array with reconfigurable redundancy that provides a flexible reliability–performance trade-off for critical tasks requiring fault tolerance and non-critical ones that benefit from increased parallelism.

An **output-stationary systolic array** is considered for this work. It is commonly used and allows high inputs and outputs reuse [45, 213] and, on average, has higher performance [105]. Previous works have shown that it is more fault-tolerant to permanent faults in arithmetic units [24].

The proposed architecture supports three **execution modes**:

- no redundancy for increased performance;
- dual-redundancy grouping (DRG) for reliability–performance trade-off;
- triple-redundancy grouping (TRG) for full protection.

The selected execution mode is configured through a control signal that controls multiplexers. The control signal comes from the host processor based on the task requirement. Since different layers of DNNs have different vulnerability, it is possible to execute different layers in different modes, thus achieving an efficient trade-off between performance and reliability.

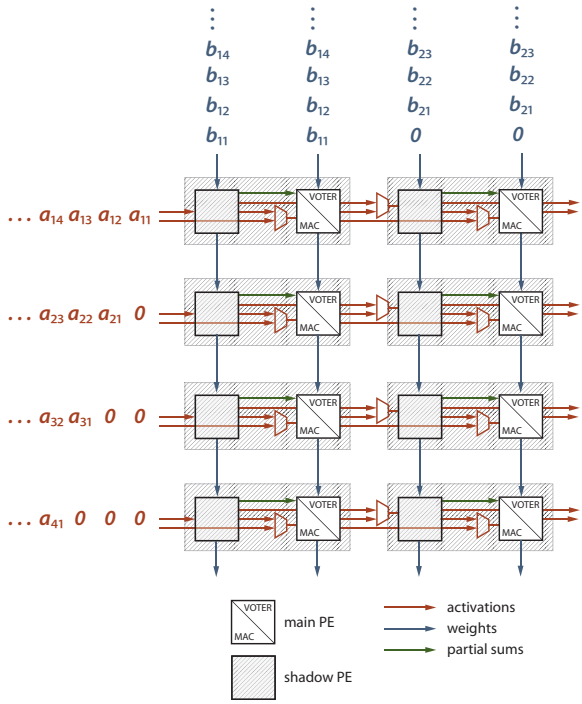


Figure 22: Arrangement of the systolic array in the dual-redundancy grouping execution mode

The proposed architecture addresses faults in registers and arithmetic units of systolic array PEs, since they lead to silent data corruption without indication of an error, unlike faults in control logic that usually result in functional interrupts that signal that there was a fault and require a reset.

DRG execution mode. In the DRG execution mode, two neighboring PEs form a group (Figure 22). They receive the same inputs; one of them acts as the main PE, the other as a shadow. Main PE compares two partial sums (OREG) and adjusts its partial sum value without re-execution using one of the possible fault correction techniques: (a) averaging of two values (DRGA) or (b) setting faulty bits to zero (DRG0). While dual redundancy does not provide full protection, these correction techniques reduce the effect of the fault on the output values, thereby enabling a reliability–performance trade-off. The choice of fault-correction technique is a design-time parameter that determines two implementation options for the DRG execution mode.

Additional connections are introduced to support redundant execution modes. To ensure that faulty IREG or WREG values do not propagate through the array, each PE passes the activation to the next PE of the same type, skipping one in between. While faults in buffers and interconnects are not in the focus of this work, such interleaved connections can help protect against those faults as well. During fault-tolerant execution modes, there are two copies of the input data traversing the systolic array. The shadow PE also passes the value of the OREG (partial sum) to the main PE for comparison. Main PE has additional functionality for fault correction that works in parallel with the MAC unit. Since the updated partial sum is available on the next clock cycle, the latency of the systolic array working in DRG execution mode equals the latency of the $N \times N/2$ systolic array (see Eq. (7) on page 25 for reference) plus 1:

$$L_{SA}^{drg} = M + N - 1 + \frac{N}{2} - 1 + 1 = M + \frac{3N}{2} - 1\,. \tag{80}$$

The total latency for the convolution operation is the latency of one matrix multiplication multiplied by the total number of steps needed to calculate output. For the OS systolic array, the total number of steps equals $S = T_a^{os} \cdot T_w^{os}$ (see Eq. (8) on page 25 for reference), and the total latency for DRG execution mode is then

$$L_{total}^{drg} = \left\lceil \frac{H_{out} \cdot W_{out}}{N} \right\rceil \cdot \left\lceil \frac{2 \cdot C_{out}}{N} \right\rceil \cdot \left(M + \frac{3N}{2} - 1 \right). \tag{81}$$

Since in the redundancy execution modes, several PEs calculate the same values, the term **effective size** of the systolic array is introduced. It defines the number of groups calculating unique values and the size of the output matrix. The effective size of the systolic array in the DRG execution mode is $N \times \frac{N}{2}$.

TRG execution mode. For the TRG execution mode, two implementations are possible: with groups of three or four PEs. The implementation option with groups of three PEs (TRG3) is shown in Figure 23 and the implementation option with groups of four PEs (TRG4) in Figure 24. The group size is the design-time parameter.

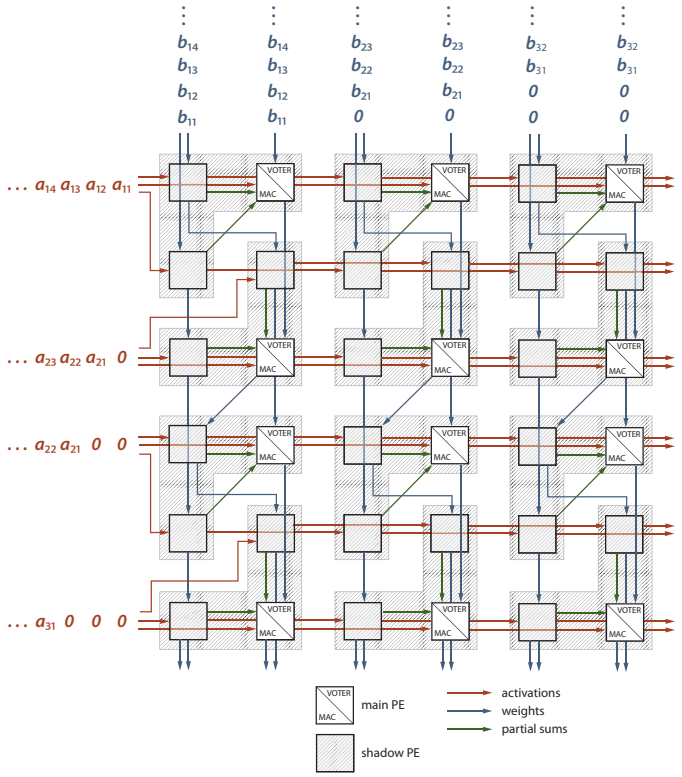


Figure 23: Arrangement of the systolic array in the triple-redundancy grouping execution mode, implementation option with groups of three PEs (TRG3)

In the TRG3 implementation, the voter in the main PE works in parallel with the MAC unit. In the TRG4 implementation, the MAC unit in the main PE is not used in the TRG execution mode, the main PE only compares the partial sums calculated by three shadow PEs.

The effective size of the systolic array, i.e., the size of the output matrix, for TRG3 implementation is $\frac{2N}{3} \times \frac{N}{2}$. The effective sizes of the matrices that can be multiplied are

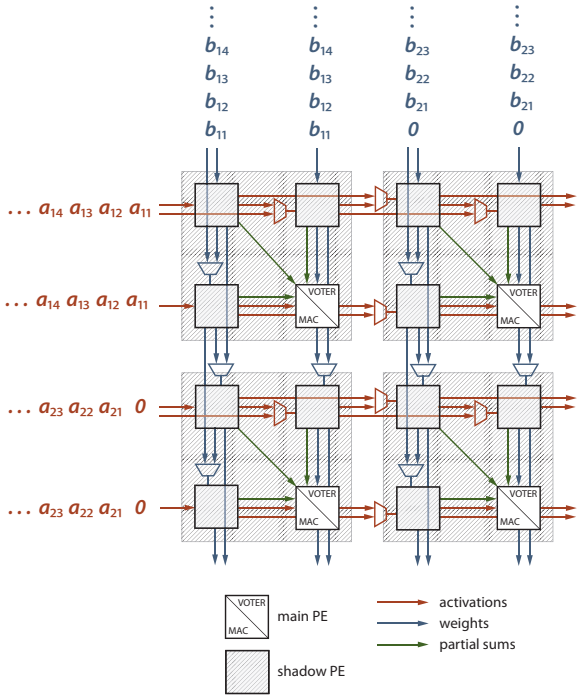


Figure 24: Arrangement of the systolic array in the triple-redundancy grouping execution mode, implementation option with groups of four PEs (TRG4)

$\frac{2N}{3} \times M$ and $M \times \frac{N}{2}$. The latency of the TRG3 implementation is

$$L_{SA}^{trg3} = M + \frac{2N}{3} - 1 + \frac{N}{2} - 1 + 1 = M + \frac{7N}{6} - 1\,. \tag{82}$$

The total latency then

$$L_{total}^{trg3} = \left\lceil \frac{3 \cdot H_{out} \cdot W_{out}}{2N} \right\rceil \cdot \left\lceil \frac{2 \cdot C_{out}}{N} \right\rceil \cdot \left(M + \frac{7N}{6} - 1 \right). \tag{83}$$

The effective size of the systolic array for the TRG4 implementation is $\frac{N}{2} \times \frac{N}{2}$. The effective sizes of the matrices that can be multiplied are $\frac{N}{2} \times M$ and $M \times \frac{N}{2}$. The latency of the TRG4 implementation is, therefore,

$$L_{SA}^{trg4} = M + \frac{N}{2} - 1 + \frac{N}{2} - 1 + 1 = M + N - 1\,. \tag{84}$$

The total latency then

$$L_{total}^{trg4} = \left\lceil \frac{2 \cdot H_{out} \cdot W_{out}}{N} \right\rceil \cdot \left\lceil \frac{2 \cdot C_{out}}{N} \right\rceil \cdot (M + N - 1). \tag{85}$$

Since both execution modes have design-time parameters, four implementation options are possible: PM-DRG0-TRG3, PM-DRG0-TRG4, PM-DRGA-TRG3, and PM-DRGA-TRG4. Their efficiency is evaluated in Section 5.3. The overview of the execution modes and implementation options is given in Table 8.

The architecture of FORTALESA is based on the output-stationary dataflow; however, other dataflows can be used to build reconfigurable systolic arrays following FORTALESA methodology. Processing elements can be grouped in a similar way, with similar interleaved connections. During the pre-loading stage, PEs from one group should receive the same input or weight. The faults would be corrected by applying a selected correction technique to the output value.

Table 8: Execution modes (run-time) and implementation options (design-time)

Execution mode	Implementation options	Effective size
Performance (PM)	Baseline SA	$N \times N$
DRG	DRGA DRG0	$\frac{N}{2} \times N$
TRG	TRG3	$\frac{2N}{3} \times \frac{N}{2}$
	TRG4	$\frac{N}{2} \times \frac{N}{2}$

5.3 Evaluation

5.3.1 FORTALESA hardware implementation parameters

The proposed architecture of the systolic array is described in SystemVerilog and synthesized using Cadence Genus 2021 and 45nm Nangate PDK. A comparison of the baseline systolic array without reconfiguration ability and four implementation options of FORTALESA is presented in Tables 9 and 10. Two sizes of the systolic array were selected: 48×48 and 132×132 to demonstrate the scalability of the design. Input and weight registers are 8-bit long, and the output register is 32-bit long.

Table 9: Hardware implementation parameters for 48×48 systolic array

Implementations	Area (mm^2)	Power (W)	Max frequency (MHz)
Baseline SA	1.726	0.158	402
PM-DRG0-TRG3	1.937	0.177	357
PM-DRG0-TRG4	1.929	0.176	372
PM-DRGA-TRG3	2.129	0.193	303
PM-DRGA-TRG4	2.091	0.190	302

Table 10: Hardware implementation parameters for 132×132 systolic array

Implementations	Area (mm^2)	Power (W)	Max frequency (MHz)
Baseline SA	12.533	3.625	285
PM-DRG0-TRG3	14.243	4.053	252
PM-DRG0-TRG4	14.284	4.029	249
PM-DRGA-TRG3	15.826	4.552	230
PM-DRGA-TRG4	15.605	4.489	262

Due to the regular structure of the proposed architecture, the design scales almost linearly proportionate to the increase in the number of processing elements. All imple-

mentations introduce around 5% overhead due to additional interconnects and MUXes, around 6-7% due to correction mechanisms for DRG0 versions, and around 17-18% for DRGA versions.

It can be noticed that the frequency might drop considerably for the implementations with the averaging correction scheme for the DRG execution mode (DRGA). Critical path of these implementation variants includes an additional adder in the main PEs, thus increasing the datapath and lowering the frequency.

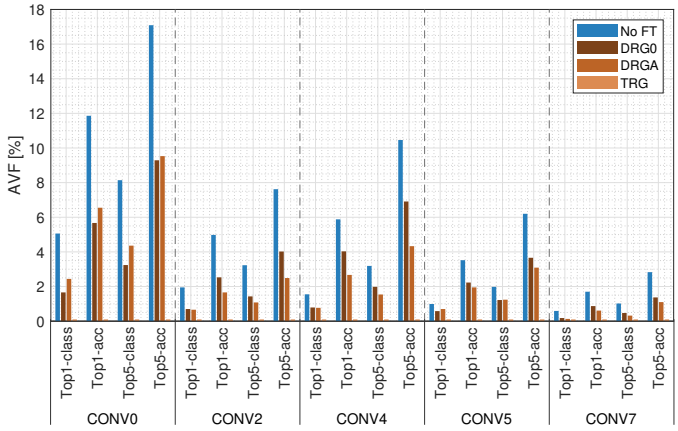


Figure 25: Layer-wise reliability assessment of AlexNet considering transient faults

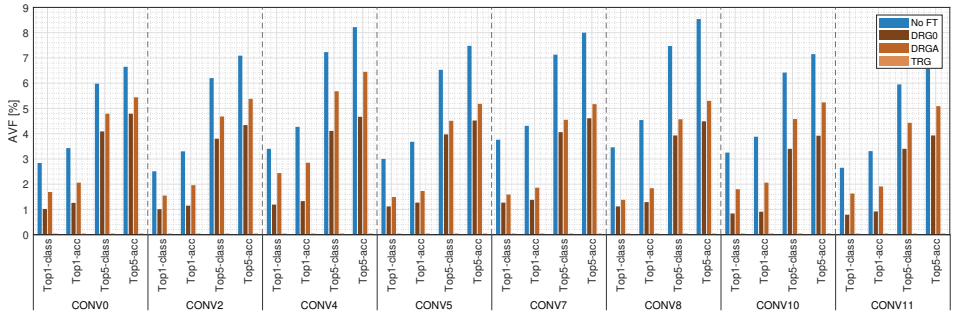


Figure 26: Layer-wise reliability assessment of VGG-11 considering transient faults

5.3.2 Reliability analysis

Experimental setup. Three common CNN models were used for reliability evaluation, including AlexNet, trained on the CIFAR-10 dataset, and VGG-11 and ResNet-18, trained on the ImageNet dataset. Models and datasets were selected to represent different network scales. For the fault injection experiments, networks were quantized to an 8-bit integer format. Experiments were performed on NVIDIA GeForce RTX 3090 24G GPU.

AVF (Architectural vulnerability factor) was used as a reliability measure. Four output errors of the DNNs were considered for the analysis: Top1-class, Top1-acc, Top5-class, and Top5-acc (see definition in Chapter 4, page 58).

For transient fault analysis, layer-wise statistical fault injection was performed using the equation introduced in [110] to achieve 95% confidence and a 5% error margin. Faults were injected in each convolution layer using the SAFPA framework (Chapter 4). For each fault injection, fault parameters (Table 3, page 44) were set randomly, and a test dataset of 10 000 inputs was fed to the network. For the TRG execution mode, it is assumed that

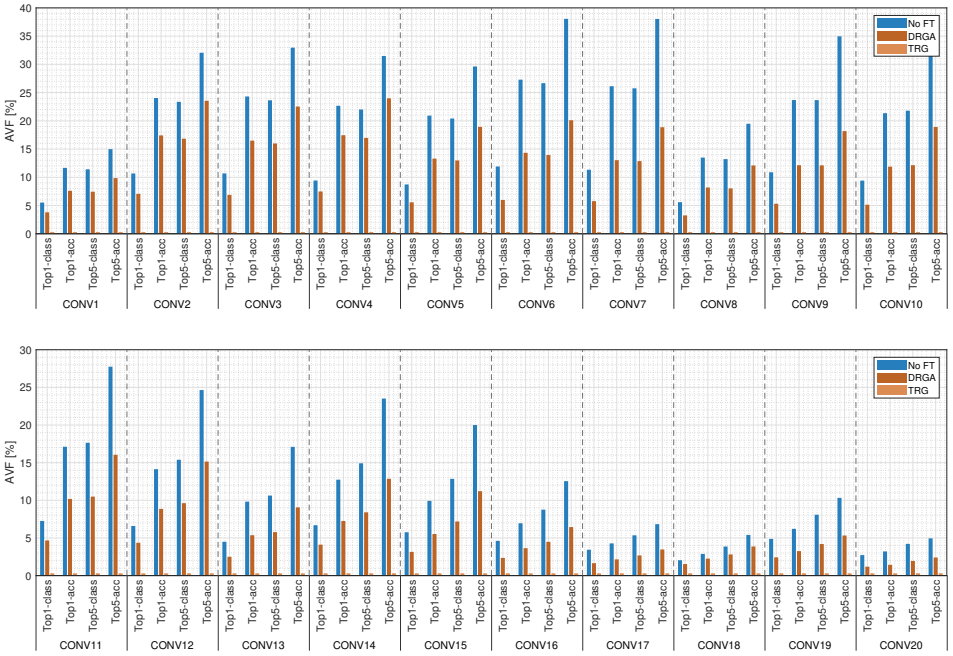


Figure 27: Layer-wise reliability assessment of ResNet-18 considering transient faults

all faults are corrected, and there are no output errors. Fully connected layers were not considered for fault injection, as they occupy only one row of the systolic array and their vulnerability factor, therefore, is very low [184].

Experimental results. Figures 25, 26, and 27 show AVF values of each layer for selected networks considering different execution modes and configurations of the 48×48 systolic array and transient faults. For ResNet-18, only DRGA is considered since the DRG0 correction scheme was not very efficient for this model.

Results show the varying vulnerability of different layers and the masking effect of DRG execution mode implementations. It can be seen that the DRG execution mode decreases the vulnerability of certain layers almost twice. Those results can be used to decide how to assign different layers of DNNs to different execution modes of the systolic array.

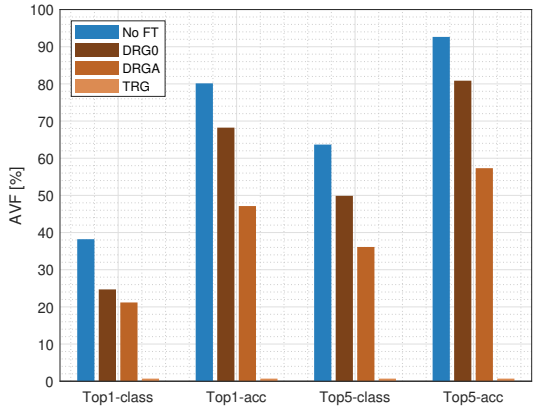
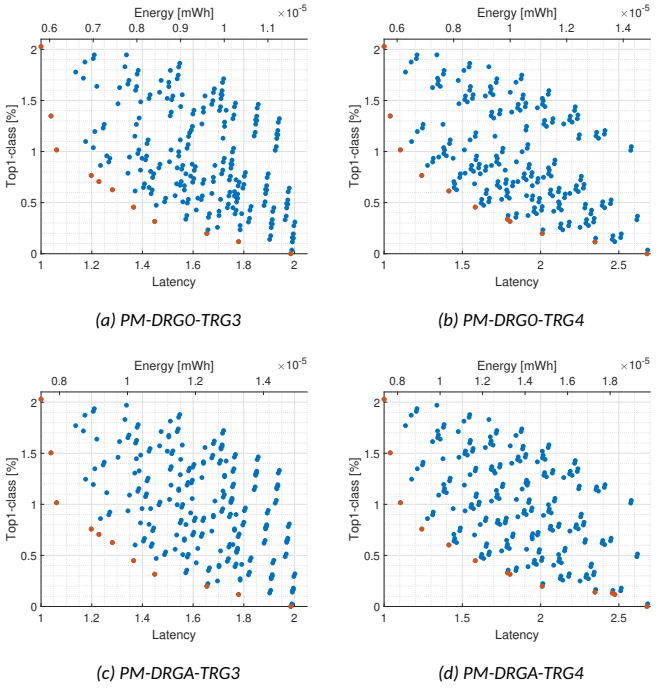


Figure 28: Reliability assessment of AlexNet considering permanent faults (stuck-at-1)

Figure 28 shows AVF values for the reliability assessment of AlexNet considering permanent faults. The analysis is done for stuck-at-1 faults since they are proven to be more critical [184]. Unlike transient faults, analysis is done for the whole network (all convolutional layers) as permanent faults persist throughout the network execution.

5.3.3 Performance vs. reliability trade-off

A flexible performance vs. reliability trade-off can be achieved by executing different layers of the DNN using different modes of the reconfigurable systolic array. Figures 29, 30 and 31 plot reliability versus execution latency of the networks and energy consumption for all the possible combinations of modes. Latency is normalized to the execution of the network using performance mode for every layer (no fault tolerance). Energy is given in milliwatt-hour (mWh). For reliability, Top1-class errors are considered. The Pareto front is marked with red color. Four implementations of FORTALESA are considered for each network since different implementations of DRG execution mode give slightly different reliability results, and different implementations of TRG execution mode affect the latency.



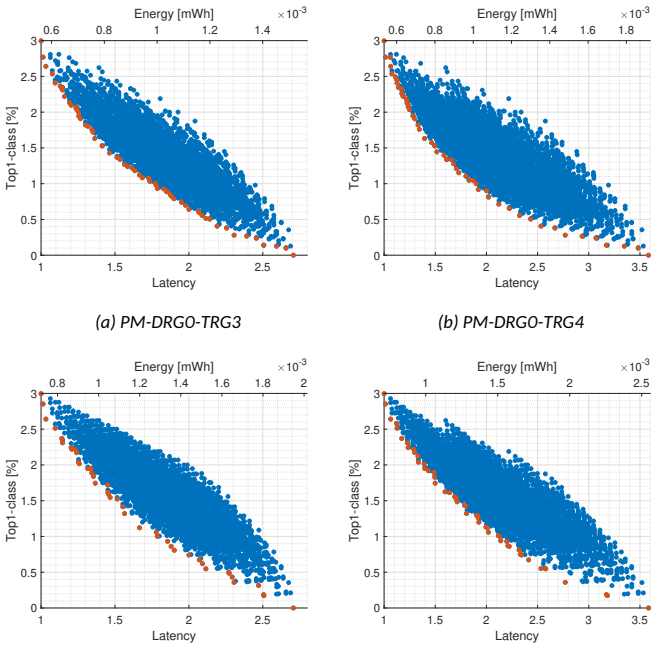
(a) PM-DRG0-TRG3 *(b) PM-DRG0-TRG4*

(c) PM-DRGA-TRG3 *(d) PM-DRGA-TRG4*

Figure 29: Reliability vs. latency and energy for AlexNet

The left-most point of the Pareto front denotes the configuration in which all layers are executed using performance mode. The execution latency is small, but there is no fault tolerance. The right-most point denotes the configuration in which all layers are executed using TRG mode. The execution latency is high, but all faults are corrected. The points in between denote configurations with a combination of execution modes, including DRG mode. The figures show that DRG execution mode allows for a flexible reliability–performance trade-off.

To compare different implementations, the AVF of the Pareto front points was plotted against a product of latency, power, area, and delay (Figure 32). It can be seen that, while



(a) PM-DRGO-TRG3 *(b) PM-DRGO-TRG4*

(c) PM-DRGA-TRG3 *(d) PM-DRGA-TRG4*

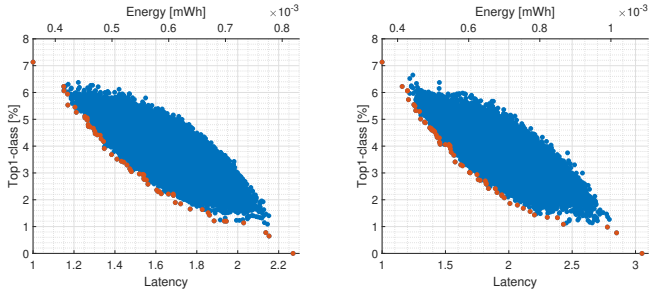
Figure 30: Reliability vs. latency and energy for VGG-11

systolic arrays with the TRG4 implementation have smaller area and power and higher frequency parameters, the version with the TRG3 implementation shows better overall results when execution latency is taken into consideration.

These figures present a design space exploration with the highlighted Pareto front, allowing designers to select a trade-off that fits the needs and requirements of their system.

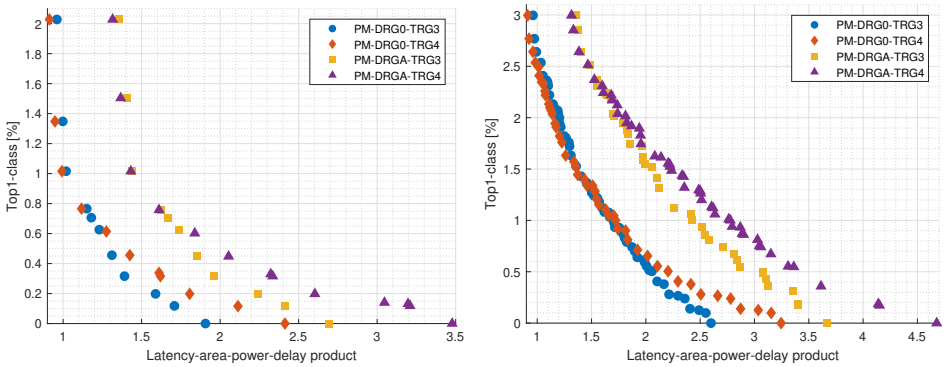
5.3.4 Comparison with static redundancy

The comparison of the proposed run-time reconfigurable systolic array architecture with the static TMR approaches is presented in Figure 33. A power-area product is plotted against the maximum possible throughput of the architecture (since the throughput of



(a) PM-DRGA-TRG3 *(b) PM-DRGA-TRG4*

Figure 31: Reliability vs. latency and energy for ResNet-18



(a) AlexNet *(b) VGG-11*

Figure 32: Comparison of FORTALESA implementation options considering reliability

FORTALESA depends on the execution mode). Throughput is calculated as the number of MAC operations performed by the systolic array per clock cycle, multiplied by the frequency. Different cases are considered for static TMR: triplication of registers only, triplication of registers and MAC units, and triplication of the whole array. As well as different sizes: 48×48 and 24×32, since this is the effective size for the 48×48 systolic array operating in TRG execution mode (TRG3 implementation option). As shown in the figure, the 24×32 systolic array with static TMR has a lower power-area product than the proposed architecture but maintains a fixed low throughput. On the other hand, the 48×48 systolic array with static TMR has high throughput, but the power-area product is significantly higher than that of the proposed architecture. FORTALESA, on average, requires 6× fewer resources than static TMR.

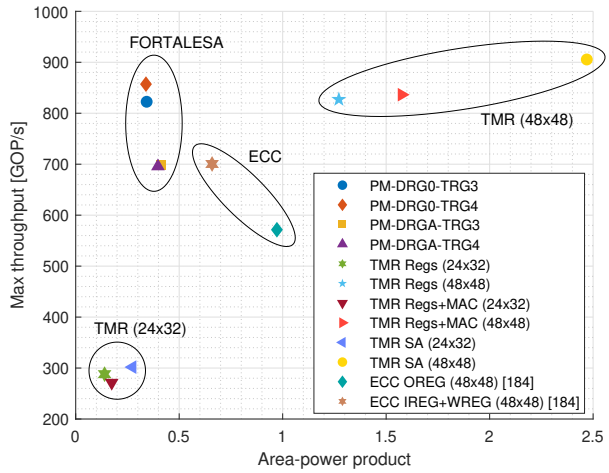


Figure 33: Comparison of FORTALESA with static TMR and selective ECC [184]

The proposed architecture is also compared with the selective ECC proposed in [184]. While it protects only selected registers, it requires, on average, 2.5× more resources than FORTALESA, which protects all registers and MAC units.

5.3.5 Comparison with the state-of-the-art

The comparison of the proposed architecture with other methods to enhance fault tolerance of systolic arrays is presented in Table 11. The proposed architecture provides a comprehensive solution that protects both registers and MAC units in the PEs, and since it utilizes a form of spatial redundancy, it covers both transient and permanent faults. Faults are corrected in both fault-tolerant modes without interrupting inference execution for recovery. Additionally, the proposed architecture provides support for reconfigurability, addressing the dynamic requirements of AI applications.

Table 11: Comparison with the state-of-the-art

Work	Covered faults		Protected parts		Fault correction
	Permanent	Transient	Registers	MAC	
[224]	✓	✗	✗	✓	✓
[221]	✓	✗	✗	✓	✓[1]
[114]	✓	✗	✗	✓[2]	✗
[184]	✗	✓	✓	✗	✓[3]
[193]	✓	✗	✓	✓	✗
[159]	✗	✓[4]	✗	✓	✗
Our	✓	✓	✓	✓	✓

[1] Requires retraining of the DNN for each faulty chip.
[2] Faults in configurational memory of FPGA.
[3] Only single-bit fault correction (ECC).
[4] Timing faults from low voltage.

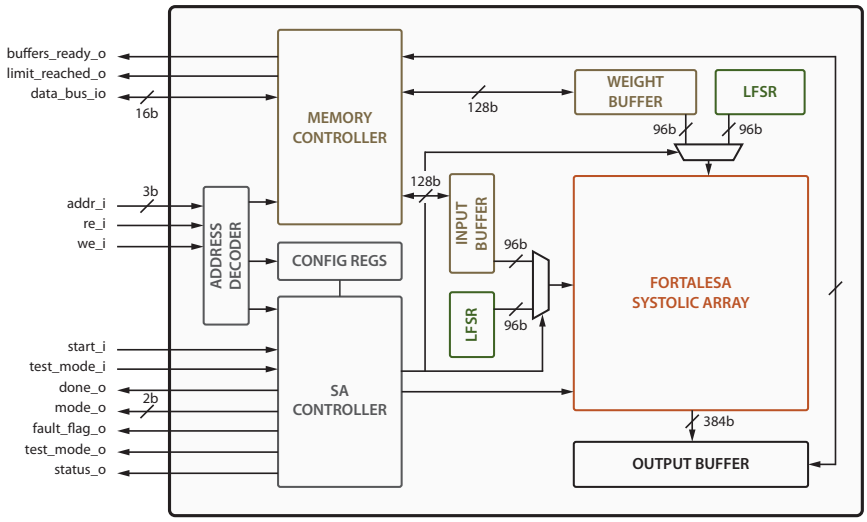


Figure 34: Block diagram of the FORTALESA-based ASIC

5.4 FORTALESA-based ASIC design

An ASIC based on FORTALESA was designed for further testing, e.g., at radiation facilities or aging chambers. The chip features a 12×12 systolic array with a controller; memory buffers for inputs, weights, and outputs (6 256×64 SRAM blocks); a memory controller;

configuration registers; and LFSRs (Linear-Feedback Shift Registers) for generating random input data. The final size of the chip is 3500×3500 µm^2. The block diagram of the chip is presented in Figure 34. The chip features a 16-bit bidirectional data bus for reading and writing. FORTALESA systolic array is configured by writing to the configuration registers. There are several status output signals for observability.

The chip was built using open-source EDA tools, including Yosys for synthesis, OpenRoad for floorplanning, physical mapping and routing, and KLayout for sign-off, and an open-source 130 nm BiCMOS PDK (Process Design Kit) from IHP. The objective of adopting an open-source design workflow is to contribute to the advancement of the open-source ecosystem by adopting an open-source tool flow for chip design and extending existing open-source systems with fault-tolerant infrastructure. Moreover, open-source solutions promote collaboration and knowledge sharing. The final layout of the ASIC and a photo of the produced chip are presented in Figure 35. The chip design and the full workflow are available open source at `https://github.com/ntlchr/fortalesa-chip`.

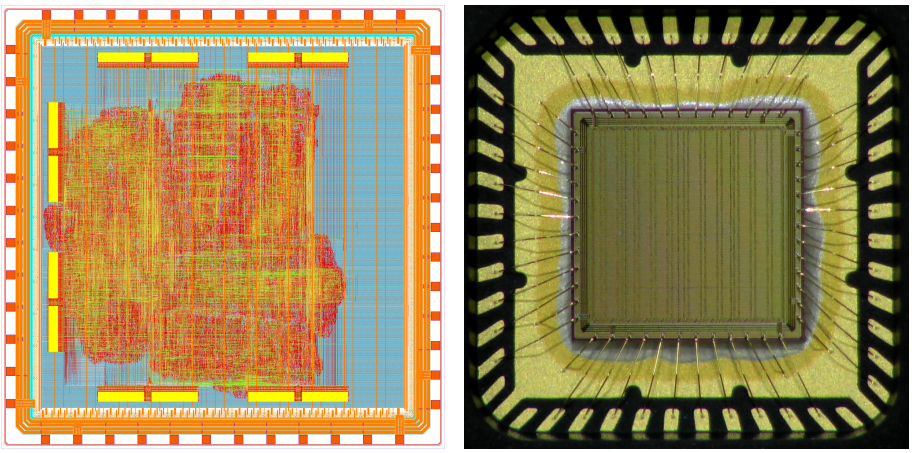


Figure 35: Layout of the FORTALESA-based ASIC and photo of the produced chip

5.5 Chapter conclusions

This chapter presents a run-time reconfigurable fault-tolerant systolic array architecture, FORTALESA, with three execution modes and four implementation options. All four implementation options were evaluated in terms of resource utilization, throughput, and improvements in fault tolerance. The proposed architecture is used to enhance the reliability of DNN inference on a systolic array through heterogeneous mapping of network layers to different execution modes. The appropriate execution mode–layer mapping for DNN inference is selected based on the vulnerability of individual layers. The proposed architecture efficiently protects both the registers and the MAC units of systolic array PEs against transient and permanent faults without interrupting inference execution. The reconfigurability feature of FORTALESA enables a speedup of up to $3\times$, depending on layer vulnerability, and requires $6\times$ fewer resources compared to static redundancy and $2.5\times$ fewer resources than the previously proposed solution for transient faults. An ASIC based on FORTALESA is developed for further testing, e.g., at radiation facilities. The design of the chip is made open source.

6 Approximation-aided Reliability of DNN Accelerators

This chapter presents a solution for approximation-aided reliability of DNN accelerators: a novel adaptive fault-tolerant approximate multiplier (AdAM). The multiplier is based on Mitchell's logarithmic multiplier that substitutes multiplication with the addition of approximated logarithms of the operands. AdAM relies on an unconventional use of the leading-one position in the inputs for fault detection by optimizing unused resources. This multiplier is a negative-overhead fault-tolerant approximate multiplier compared to exact multipliers. The fault-tolerance solution at the individual component layer is developed considering the characteristics of the targeted application, ensuring efficient reliability through a cross-layer approach. The chapter is based on publications [II], and [III]:

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

M. Taheri, N. Cherezova, S. Nazari, A. Azarpeyvand, T. Ghasempouri, M. Daneshtalab, J. Raik, and M. Jenihhin. AdAM: Adaptive approximate multiplier for fault tolerance in DNN accelerators. *IEEE Transactions on Device and Materials Reliability*, 25(1):66–75, 2025

6.1 Introduction

The error introduced by the approximation is deterministic, and its impact on the network's accuracy drop can be studied in detail. However, soft errors are unpredictable in contaminated and harsh environments, which can lead to DNN malfunction and a drastic drop in accuracy [111]. The design of approximate arithmetic circuits and their reliability have been extensively covered in the literature both without considering an application [22, 150, 189] and concerning DNNs [3, 91, 155, 158, 170, 181]. The work presented in [128] introduces a comprehensive library of approximate adders and multipliers, primarily intended for image and video processing circuits. However, as highlighted in [169], using multipliers from this library involves a trade-off between fault resilience and energy efficiency. This indicates the need for specific approximate multiplier (AxM) designs to optimize energy savings and resource utilization while enhancing DNN accuracy. Additionally, the impact of approximate multipliers on DNN accuracy can vary significantly, with some leading to a substantial drop in performance. In contrast to the proposed approximate multipliers, AdAM considers reallocating the resources saved by approximation to improve fault tolerance.

Traditional fault tolerance methods for DNN hardware accelerators include hardware redundancy, correction codes for parameter memories, and algorithm-based fault tolerance. Triple Modular Redundancy (TMR), despite achieving 100% fault coverage, incurs a substantial near 200% area and power overhead [126]. Therefore, numerous algorithms and frameworks have been developed to improve the efficiency of these methods and balance their hardening effects and design costs, such as Approximate TMR (ATMR) [8]. ATMR replaces some TMR modules with approximate ones while ensuring that the majority voter produces the correct output [10]. However, ATMR still requires duplicating the entire combinational circuit, even at the finest level of granularity [47, 190].

To address the high overhead of traditional fault-tolerance methods, this chapter presents an adaptive approximate multiplier that provides high reliability while having a smaller area and power-delay product (PDP) than an exact multiplier.

6.2 AdAM Architecture

We propose an architecture for an adaptive fault-tolerant approximate multiplier tailored for DNN accelerators. This architecture includes an adaptive adder that relies on an unconventional use of input Leading One Detector (LOD) values for fault *detection* and *mitigation* by optimizing unused adder resources. A gate-level-optimized LOD design and a lightweight, triplicated hybrid adder design are used to further enhance the proposed architecture's reliability, resource utilization, and efficiency. The base for the proposed multiplier is the classical Mitchell multiplier [125]. However, the methodology can be applied to all logarithmic approximate multipliers. Another level of approximation is introduced in the adaptive adder (Figure 37), considering the application of this multiplier in DNNs with a proven negligible impact on the network accuracy (see Table 17 in the results section). Mitchell multiplier employs approximate logarithms of the input values. By adding these logarithms, Mitchell's algorithm estimates the product. The final result is obtained by taking the antilogarithm of this sum.

6.2.1 Definitions

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

$$A = \sum_{i=0}^{k_A} 2^i a_i, B = \sum_{i=0}^{k_B} 2^i b_i, \quad a_i = b_i = \{0,1\} \tag{86}$$

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

$$A = 2^{k_A}\left(1 + \sum_{i=0}^{k_A-1} 2^{i-k_A} a_i\right) = 2^{k_A}(1+X_A) \tag{87}$$

$$B = 2^{k_B}\left(1 + \sum_{i=0}^{k_B-1} 2^{i-k_B} b_i\right) = 2^{k_B}(1+X_B) \tag{88}$$

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

$$\log(A) = k_A + \log(1+X_A) \tag{89}$$

$$\log(B) = k_B + \log(1+X_B) \tag{90}$$

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

$$\log(A) \approx k_A + X_A \tag{91}$$

$$\log(B) \approx k_B + X_B \tag{92}$$

The logarithm of $A * B$ can be approximated as

$$\log(A*B) \approx k_A + X_A + k_B + X_B \tag{93}$$

The antilogarithm of the above expression is the final product:

$$\hat{P} = 2^{k_A+k_B}(1+X_A+X_B) \tag{94}$$

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

the truncation parameter, the fractional part X is defined as

$$X_A^t = \sum_{i=0}^{k_A-1} 2^{i-k_A} a_i \cdot \left[|i-k_A| \geq n-t\right] \tag{95}$$

$$X_B^t = \sum_{i=0}^{k_B-1} 2^{i-k_B} b_i \cdot \left[|i-k_B| \geq n-t\right] \tag{96}$$

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

$$\hat{P} = 2^{k_A+k_B}(1+X_A^t+X_B^t) \tag{97}$$

Truncating the fractional parts introduces additional errors into the result. The absolute error ($P-\hat{P}$) behavior of the original 8-bit Mitchell's multiplier and AdAM is compared in Figure 36 over the entire input domain. The maximum errors introduced by each method are marked in the figure. There is no additional error when the truncated bits in both operands are '0'; as the number of '1' in the truncated bits increases, the additional error also increases.

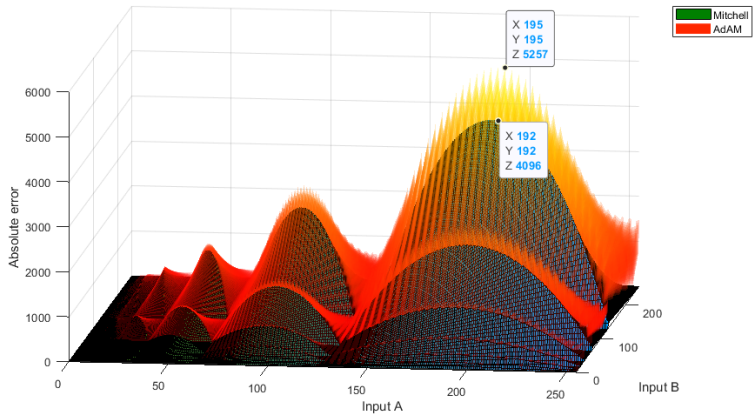


Figure 36: Comparison of absolute error distribution between the original 8-bit Mitchell's multiplier and AdAM: the top point shows the maximum absolute error for AdAM, and the lower point shows the maximum absolute error for Mitchell's multiplier

However, the literature shows that DNN weights do not follow a uniform distribution and are mostly centered around zero [18]. Since the proposed multiplier provides greater protection for values with the smaller LOD value, it can effectively protect most of the network's computations during inference.

The impact of the proposed multiplier on the accuracy of different networks is reported in the results section.

The level of fault tolerance of the proposed multiplier is defined by two parameters: the aforementioned truncation parameter t and duplication level h. The truncation parameter t defines the minimum number of protected bits, and the duplication level h defines the maximum possible number of protected bits. Smaller operands have smaller mantissa values and do not require a full adder. Therefore, more adder resources can be used for fault tolerance by duplicating the addition of more higher-order bits. Duplication level h does not affect the multiplier's accuracy; as a hardware parameter, it affects

area, power, and delay. Duplication level h defines the amount of adder resources that can be used for fault tolerance (see Subsection 6.2.2.1 for a detailed description). A higher h value means a higher level of fault tolerance and higher resource utilization. A smaller value means a lower level of fault tolerance and lower resource utilization. Different configurations of the proposed multiplier are denoted as AdAM(t,h).

6.2.2 Hardware implementation

The proposed architecture of the multiplier is presented in Figure 37 (the contributions and extensions to the logarithmic Mitchell multiplier are marked with red color). First, a novel optimized *Leading One Detector (LOD)* circuit (Subsection 6.2.2.2) is used to find the index of the first '1' bit in each operand. This index, denoted as k, is the characteristic or integer part of the logarithm and has $\log_2(n)$ bits. Then, the operands are shifted left by k bits, aligning the leading one with the Most Significant Bit (MSB). $(n-1)$ bits after the leading one represent the mantissa part, denoted as m. The mantissa is truncated to $(n-1-t)$ bits.

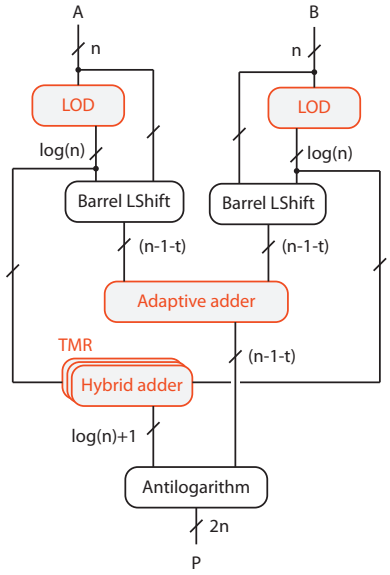


Figure 37: AdAM architecture (the contributions and extensions to the logarithmic Mitchell multiplier are marked with red color)

The truncated operands are passed to the adaptive $(n-1)$-bit adder that adds mantissa together and duplicates the addition of several higher-order bits depending on the k value of the biggest operand for fault *detection* and *mitigation*. As it was already mentioned, the number of duplicated bits depends on the truncation parameter t and duplication level h.

The architecture of the *adaptive adder* is shown in Figure 38 (subsection 6.2.2.1). The adder is based on the carry-lookahead adder, and the carry-generation logic is omitted from the figure to save space. Duplicated results are compared, and if a fault is detected, the faulty bit is set to zero using AND gates (marked in red in the figure). Due to the truncation of the mantissa, up to t lower-order bits are excluded from the calculation, which affects only the bigger numbers with the $k > (n-t)$. This introduces a small error compared to the original Mitchell algorithm (discussed in the following section).

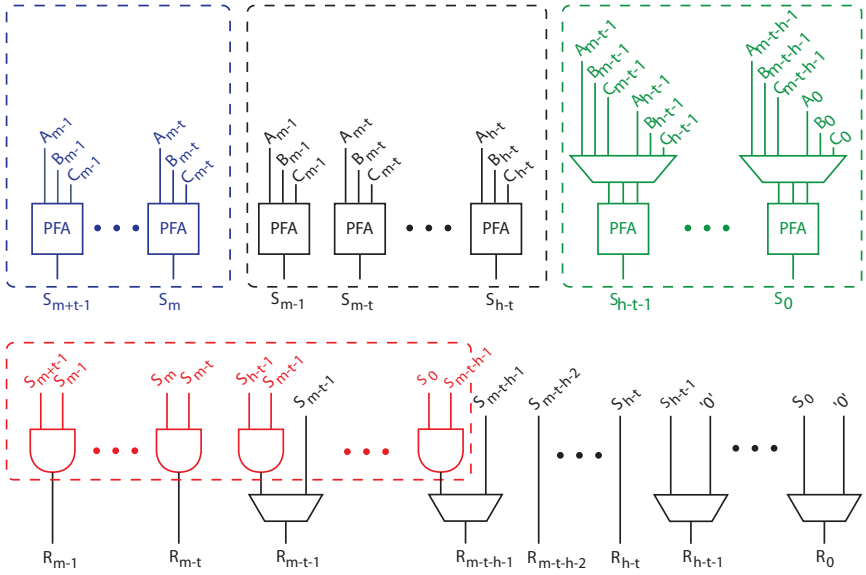


Figure 38: Adaptive adder architecture: A and B are inputs, C is carry values, and PFA stands for partial full adder, m here denotes the size of the truncated mantissa $(n-1-t)$. The truncation parameter t defines the minimum number of protected bits; t PFAs that duplicate the addition of t higher-order bits of the mantissa are colored blue. The duplication level h defines the maximum number of protected bits; $h-t$ PFAs with multiplexers that can duplicate the addition of higher-order bits or add lower-order bits are colored green.

The k values of the operands are added separately using a small *hybrid adder*. A hybrid between a resource-intensive but fast carry look-ahead adder and a lightweight but slow due to the accumulated delay ripple carry (CR) adder is selected for this task. This adder is replicated three times for extra fault tolerance, since the order of the final output depends on the result of this addition. A majority voter selects the final result. Then, the antilogarithm algorithm is used to get the product of multiplication. The sum of k values determines the position of the leading one in the output product, followed by the sum of the mantissa parts using the appropriate shift operation.

6.2.2.1 Adaptive adder

The adaptive adder is designed to detect and mitigate faults based on the multiplier inputs' k values. The adder is divided into three parts: t PFAs (Partial Full Adders) for duplicating the addition of t higher-order bits (colored blue in Figure 38), $(h-t)$ PFAS with multiplexers on inputs that can duplicate the addition of up to $(h-t)$ higher-order bits or add lower-order bits (colored green in Figure 38), and $(n-1-h)$ PFAs for the addition of the mantissa. This way, for smaller operands with smaller mantissas that do not require a whole adder, more resources can be allocated to fault tolerance. The size of the truncated mantissa can be expressed as $W_{X^t} = k \cdot [k < n-t] + (n-1-t) \cdot [k \geq n-t]$ using Iverson brackets notation. The available adaptive adder resources can be expressed as $W_{AR} = (n-1) - W_{X^t}$. Thus, the number of protected bits is $W_{PB} = W_{AR} \cdot [W_{AR} < h] + h \cdot [W_{AR} \geq h]$.

As an example, Figure 39 shows the scheme in which the proposed multiplier introduces fault tolerance considering 8-bit inputs with a truncation parameter $t = 2$ and duplication level $h = 3$. As shown in this figure, five cases are considered. If the maximum LOD of the inputs is 7, two lower-order bits are discarded, and a two-bit adder of the adaptive

adder is dedicated to recomputing the addition of two higher-order bits. These results are compared, and the mismatched bits are set to zero.

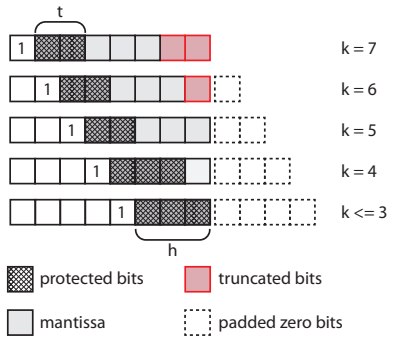


Figure 39: Fault-tolerance and error introduced based on different input values ($n = 8, t = 2, h = 3$)

For $k = 6$, only the Least Significant Bit (LSB) is discarded, and two higher-order bits are protected the same way as in the previous case. When $k = 5$, no bits are discarded, and two higher-order bits are protected. For $k = 4$, three higher-order bits are protected, and only the LSB is not monitored. In the case of $k \leq 3$, all bits are protected, enabling the proposed multiplier to provide comprehensive fault detection and mitigation for all inputs.

6.2.2.2 LOD design

The gate-level structure of the proposed LOD circuit is presented in Figure 40. The circuit is divided into two functional parts: zero flag calculation and leading-one position detection (LOPD). LOPD consists of several stages. During the first stage, the input is divided into nibbles, which are processed in parallel. For each nibble, three signals are calculated except for the first one: a, b, and c. Only two signals are calculated for the first nibble: b and c. Signal a defines whether there is a '1' bit in the nibble; signal b defines whether the output is even or odd; and signal c defines whether there is '1' in the two higher bits of the nibble. During the second stage, those signals form the final result based on the highest nibble with a '1' bit. Since the zero flag is only required at the last stage of the multiplier to determine whether the final product is zero, it is computed from the LOPD output values. The proposed design requires fewer logic gates (as demonstrated in the results section) than designs found in the literature (e.g., ScaleTRIM [55]) by reusing the output values for calculating the zero flag, knowing that it can have a longer delay.

6.3 Experimental Results

6.3.1 Experimental Setup

FreePDK 45 nm Nangate technology library is used in Cadence Genus 2023 to compare the hardware characteristics (area, latency, power consumption) of the proposed methods with the state-of-the-art. The accuracy is reported using Mean Absolute Relative Error (MARE), calculated as

$$\text{MARE} = \frac{1}{N}\sum_{i=1}^{N}\left|\frac{P_i - \hat{P}_i}{P_i}\right|,$$

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

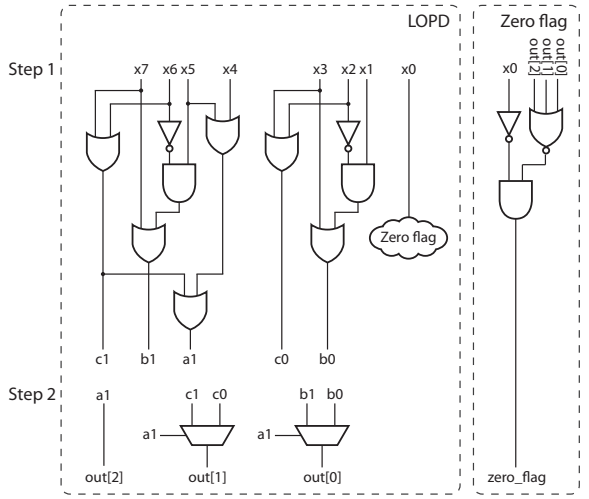


Figure 40: Gate-level structure of the proposed LOD

Additionally, a design of a MAC (multiply-accumulate) unit in a systolic array is synthesized for an ASIC to better illustrate the results in an AI core. The impact on the accuracy of the proposed adaptive multiplier is studied on different networks (i.e., LeNet-5, AlexNet, ResNet-18, VGG-16, DenseNet) trained on MNIST and CIFAR-10 using 8-bit INT with the help of the AdaPT framework [42]. Finally, the impact of the proposed multiplier on DNN reliability is evaluated using the aforementioned benchmarks.

Fault injection. Fault injection is performed, assuming single-bit-flip faults in the network's MAC operation of a systolic array, for reliability assessment. According to the adopted single-bit fault model, a random bit-flip is injected into a random MAC unit of the systolic array core at a random execution time of the network, and the entire test set is fed to the network to obtain the accuracy of the network. Transient faults (bit flips) within subcomponents of the multipliers, such as LOD, adders, and barrel shifters, were considered for the fault injection experiments. A simulation was conducted to determine how those faults will propagate to the multiplier's output. For fault propagation, the RTL model of the multiplier was used, and a random set of potential fault positions in its unprotected components was identified. Faults were injected as bit flips during the simulation using Cadence Genus. For each considered fault position, all input combinations were applied to the multiplier to generate look-up tables with the outputs. Those look-up tables were used during the fault injection in the DNNs. The number of faults injected was chosen based on preliminary experiments and existing literature. Using equations provided in [110], a 95% confidence level and 1% error margin were reached.

6.3.2 Hardware utilization

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

Table 12 shows the area, power, and delay of the proposed 8-bit LOD architecture compared to the state-of-the-art design. The proposed design has fewer gates and a shorter critical path compared to the other methods in the literature.

Tables 13, 14, and 15 report the accuracy, efficiency, and fault tolerance (FT) of 8-bit, 16-bit, and 32-bit approximate multipliers compared with the proposed method. Wal-

Table 12: LOD hardware comparison between the proposed method and the state-of-the-art

LOD Architecture	Delay (ps)	Power (µW)	Area (μm^2)	PDP (aJ)
ScaleTrim [55]	156	1.12	9.84	174.72
Proposed	136	1.09	9.57	148.24

lace, DRUM [71], TOSAM [196], and ScaleTrim [55] are used for this comparison. Logarithmic approximate multipliers such as TOSAM show better performance compared to other state-of-the-art non-logarithmic ones, including DSM [132], RoBA [220], and LETAM [195]. For instance, TOSAM structures exhibit significantly lower energy consumption and area utilization compared to DRUM, DQ4:2C4, and AS-RoBA multipliers, while maintaining comparable accuracy. Specifically, the TOSAM(1,5) configuration achieves 4% lower delay, 78% lower area, and 70% lower energy consumption compared to the RoBA multiplier. The results also show that the overall impact of the TOSAM multiplier, one of the baseline approaches in this work, on delay, power, area, and MARE, is lower than that of other approximate multipliers with similar MARE values. Therefore, for the sake of space, other approximate multipliers are not included in the comparison.

Table 13: Accuracy and efficiency of 8-bit approximate multipliers compared with AdAM

Architecture	Delay (ns)	Power (µW)	Area (μm^2)	PDP (fJ)	MARE (%)	FT
Exact (Wallace)	0.85	360	417	306	0.0	No
DRUM(3)	0.70	104	143	72.8	12.6	No
TOSAM(0,2)	0.58	120	186	69.6	10.1	No
TOSAM(0,3)	0.68	144	198	97.9	7.7	No
DRUM(4)	1.00	172	208	172	6.4	No
TOSAM(1,5)	0.88	231	291	203.2	4.1	No
AdAM(2,3)	1.13	165	152	186.4	4.7	Yes
ScaleTrim(4,8)	1.8	143	216	257.4	3.3	No

The proposed multiplier has hardware parameters similar to the state-of-the-art approximate multipliers with similar accuracy while providing improved reliability through its fault detection and mitigation capability.

Table 16 shows the results of MAC implementation with different multipliers. The MAC unit with the proposed multiplier takes less area and has a lower PDP value.

6.3.3 DNN accuracy

Table 17 compares the accuracy of different CNN architectures using the proposed approximate multiplier with the baseline accuracy using the exact multiplier. The results in Table 17 were obtained by directly substituting the accurate multiplier with the proposed multiplier without any retraining or fine-tuning. The evaluation shows that the accuracy of DNN with the proposed method is very close to the baseline. Hence, the proposed multiplier has a negligible effect on DNN accuracy. It is observed that the accuracy of approximated networks can sometimes exceed that of exact networks. The observed higher accuracy with AdAM compared to the exact multiplier in some cases can be attributed to the probabilistic nature and inherent non-linearity of DNNs, as well as the sensitivity of certain bit positions to errors. This observation aligns with previous studies [156] showing that errors in arithmetic logic can lead to higher inference accuracy; consequently, they are referred to as “good faults”. This behavior arises because faults, particularly those af-

Table 14: Accuracy and efficiency of 16-bit approximate multipliers compared with AdAM

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

Table 15: Accuracy and efficiency of 32-bit approximate multipliers compared with AdAM

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

Table 16: Efficiency of 8-bit MAC unit of a systolic array with different multipliers

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

fecting lower-order bits (which can be interpreted as errors caused by approximate multipliers), do not always degrade accuracy. This phenomenon aligns with the broader understanding that DNNs are not strictly deterministic and can potentially exhibit more accuracy under specific fault conditions. Thus, the accuracy gains with AdAM can be seen conceptually as part of this broader context of DNN fault tolerance and behavior.

Table 17: Accuracy comparison of different CNNs with an exact (baseline) and the proposed approximate multiplier

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

6.3.4 Reliability analysis

To showcase the impact of the AdAM multiplier on reliability and performance trade-offs, the fault injection simulations are performed on a variety of 8-bit convolutional neural network architectures, including AlexNet, DenseNet, Inception, VGG-11, VGG-13, VGG-19, ResNet-18, ResNet-34, and ResNet-50 with four different configurations: unprotected exact multipliers (Unp-Exact), unprotected approximate multipliers (Unp-AxM), protected exact multipliers with TMR (Pro-TMR), and protected approximate multipliers with AdAM (Pro-AdAM).

The DNN reliability is evaluated by comparing the output probability vector of the golden run (i.e., the DNN that behaves as expected, without faults) and the faulty run (i.e., the DNN that includes the fault). The output probability vector represents the list of possible classes with their associated confidence scores predicted by the DNN. The SDC rate is defined as the proportion of faults that cause misclassification in comparison with the golden model. Since DNNs often produce not a single correct output but a ranked list of outputs, each with a confidence score [111], we need new criteria to determine what constitutes an SDC for a DNN application. Therefore, we consider four types of SDCs as follows:

- **SDC-1:** The top-ranked element predicted by the DNN is different from that predicted by its fault-free execution. This is critical to assess because the top-ranked element is what is typically used for downstream processing. This criterion directly compares the class with the highest confidence score in the output probability vector of the faulty run to that of the golden run. If the top-ranked class differs between the two runs, this is counted as an SDC-1.
- **SDC-5%:** More than 5% of variation in the top-ranked output confidence score compared to the golden model. This criterion focuses on the confidence score of the top-ranked class in the output probability vector. If the confidence score deviates by more than 5% from the golden run, it is counted as an SDC-5%.

- **SDC-10%:** More than 10% of variation in the top-ranked output confidence score compared to the golden model. Similarly, this criterion looks for deviations in the confidence score of the top-ranked class that exceed 10% from the golden run.

- **SDC-20%:** More than 20% of variation in the top-ranked output confidence score compared to the golden model. This criterion identifies faults where the confidence score of the top-ranked class in the output probability vector deviates by more than 20% from the golden run.

All four criteria compare the output probability vector of the faulty run with that of the golden run. SDC-1 assesses the correctness of the top-ranked class, while SDC-5%, SDC-10%, and SDC-20% evaluate how much the confidence score of the top-ranked class changes in the faulty run relative to the golden run.

To quantify the effectiveness of each method in mitigating faults, the *fault coverage* metric is used. It is defined as $(100 - \text{SDC value})$, representing the percentage of faults that are correctly handled (i.e., do not result in silent data corruption) by each method.

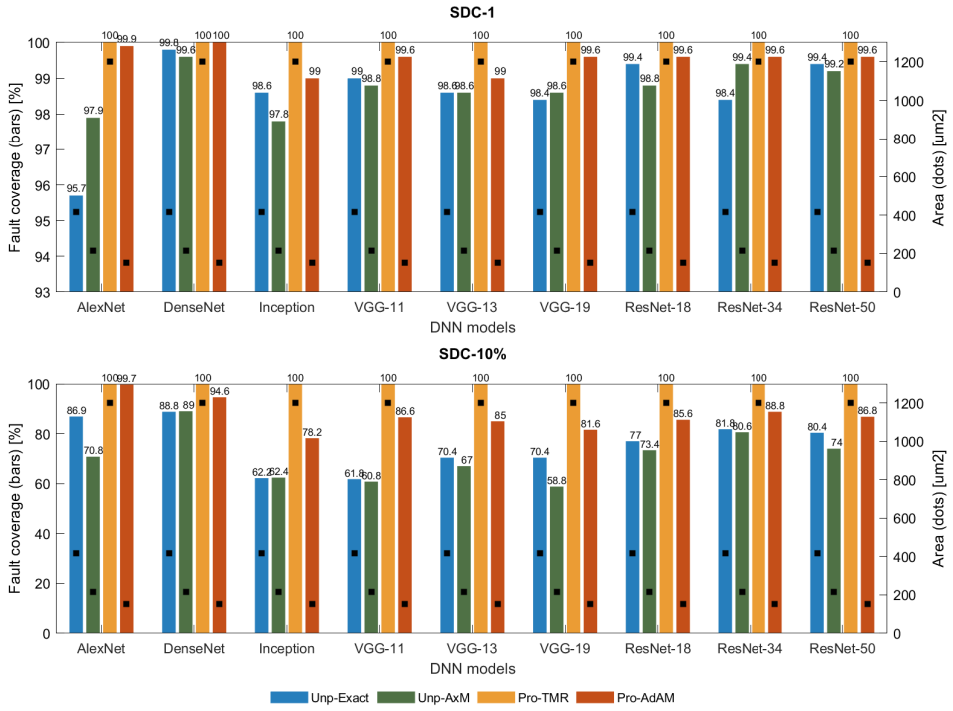


Figure 41: Hardware efficiency (area) and fault resilience (fault coverage considering SDC-1 and SDC-10%) trade-offs in different benchmarks. Unp-Exact: unprotected exact multiplier, Unp-AxM: unprotected approximate multiplier, Pro-TMR: exact multiplier protected with TMR, Pro-AdAM: proposed multiplier

Figure 41 demonstrates the fault tolerance comparison and reliability improvement (for SDC-1 and SDC-10% as two examples) of different networks by using the protected approximate multiplier proposed in this work, compared to the unprotected exact and approximated networks, and protected networks using TMR. As illustrated, TMR provides 100% protection but also requires about 200% area overhead. Unlike TMR, our technique introduces a high-reliability improvement without hardware overhead. The results for

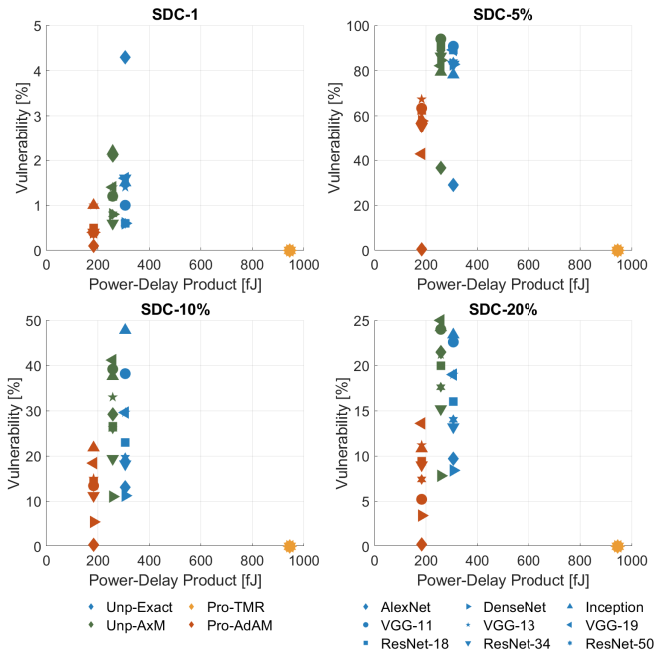


Figure 42: PDP and vulnerability tradeoffs (considering different SDCs) in different benchmarks. Unp-exact: unprotected exact multiplier, Unp-AxM: unprotected approximated multiplier, Pro-TMR: exact multiplier protected with TMR, Pro-AdAM: proposed multiplier

reliability improvement considering SDC-5% and SDC-20% are reported in Tables 18 and 19. Since the main objective of the proposed multiplier is to have the best trade-off in PDP and vulnerability, Figure 42 illustrates these comparisons based on different vulnerability metrics (SDC-10, SDC-5%, SDC-10%, and SDC-20%). In these charts, the closer to the origin (0,0), the higher the cost-efficiency of fault tolerance, i.e., lower vulnerability and lower PDP. As shown, TMR is less efficient for edge AI applications due to its high PDP, whereas the proposed method (AdAM) is closest to the origin.

Table 18: Fault coverage (SDC-5%) in different benchmarks (%)

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

Table 19: Fault coverage (SDC-20%) in different benchmarks (%)

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

6.4 Chapter conclusions

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

The proposed multiplier achieves a reliability level close to that of multipliers protected by Triple Modular Redundancy (TMR), while using 2.74$\times$ less area and 39% lower power-delay product than the unprotected exact multiplier. In fact, it has similar area, delay, and power consumption parameters to state-of-the-art approximate multipliers with comparable accuracy, while providing fault detection and mitigation capabilities.

7 Conclusions

Reliability is the foremost concern for modern nanoelectronic devices, especially hardware accelerators deployed in mission- and safety-critical applications. The objective of this PhD thesis is to evaluate the fault-tolerance vulnerability of advanced computing platforms, with a focus on DNN hardware accelerators, and to develop efficient reliability solutions.

First, Chapter 3 investigates the fault-tolerance vulnerability in heterogeneous SoC FPGA platforms. The analysis shows gaps in the fault-tolerance mechanisms offered off the shelf by vendors. Heterogeneous SoC FPGA platforms comprise a diversity of tightly integrated components that should all be protected to ensure the system's reliability and availability. However, the current approach to fault tolerance is fragmented and not systematic. The most protected components are the processing system and memory, whereas uncore components are not protected and are less considered in the literature, leaving the system vulnerable. Moreover, vendor-provided solutions are not intended for harsh environments and critical applications. This necessitates complementary user-space fault-tolerance mechanisms, tailored to target mission and application profiles. For better protection, user-space methods should interact with vendor-provided ones, resulting in an integrated fault-tolerance solution spanning multiple system layers. A cross-layer approach to reliability enables a globally optimized fault management system and an efficient solution that accounts for the specifics of the application and underlying hardware.

Second, Chapter 4 presents a scalable method for reliability evaluation of DNN hardware accelerators based on systolic arrays, called SAFPA. This method effectively combines fault injection with fault propagation analysis. SAFPA is several orders of magnitude faster than microarchitecture-level fault injection because, instead of time-consuming cycle-accurate simulation of the systolic array, errors are added directly to the DNN layer output via fault propagation analysis. A framework based on the proposed method has been made open source.

SAFPA is used to analyze the reliability of DNN models for computer vision tasks, namely CNNs and ViTs. Evaluation of a representative CNN model using the proposed method shows that WS dataflow has, on average, higher vulnerability with a smaller systolic array size, while OS dataflow has higher vulnerability with a larger systolic array size. Additionally, on average, in IS and WS systolic arrays, registers holding stationary values (activations and weights, respectively) lead to a higher percentage of output errors. For the OS systolic array, registers holding activation values are the most vulnerable. Evaluation of three ViT models shows that transformers exhibit an operation-dependent vulnerability pattern, rather than the layer-dependent pattern observed in CNNs, with significant criticality of value matrix calculation. Analysis shows the effect of architectural regularity on reliability: hierarchical transformer models like Swin exhibit high variation in vulnerability across the layers, resembling the behavior of CNNs.

Additionally, SAFPA is used for a design space exploration of reliability-performance trade-offs in CNNs deployed on systolic arrays with reconfigurable size. Based on the extensive reliability evaluation of selected CNN models, a layer-wise reconfiguration strategy for RSAs is proposed to balance reliability and performance. This strategy identifies the optimal systolic array size for each layer.

Third, Chapter 5 presents a systolic array architecture with runtime reconfigurable fault tolerance, called FORTALESA. FORTALESA supports three execution modes for efficient reliability–performance tradeoff. The proposed architecture is used to enhance the reliability of DNN inference on a systolic array through heterogeneous mapping of network layers to different execution modes. The appropriate execution mode–layer mapping for

DNN inference is selected based on the vulnerability of individual layers. Reliability features of FORTALESA efficiently protect both the registers and arithmetic units of systolic array PEs against transient and permanent faults without interrupting inference execution. The reconfigurability of FORTALESA enables a speedup of up to 3×, depending on layer vulnerability, and requires 6× fewer resources compared to static redundancy and 2.5× fewer resources than the previously proposed solution for transient faults. Furthermore, an ASIC based on FORTALESA was designed for testing, e.g., radiation and aging experiments, using an open-source ecosystem for chip design. The design of the chip and scripts for generating a final layout are made open source to promote collaboration and knowledge sharing and to extend existing open-source systems with fault-tolerant architectures.

Finally, Chapter 6 presents a fault-tolerance solution for an individual component of a DNN accelerator, namely an adaptive fault-tolerant approximate multiplier, called AdAM. The solution shows how approximation can aid reliability. AdAM is based on Mitchell's logarithmic multiplier that substitutes multiplication with the addition of approximated logarithms of the operands. AdAM relies on an unconventional use of the leading-one position in the inputs for fault correction by optimizing unused resources, thereby enabling negative-overhead fault tolerance compared to exact multipliers. This way, it achieves a reliability level close to that of TMR-protected multipliers, while having 2.74× less area and 39% lower power-delay product than the unprotected exact multiplier. Efficient resource utilization and optimization of multiplier components ensure that AdAM has area, delay, and power consumption metrics comparable to state-of-the-art approximate multipliers with comparable accuracy, while providing fault-tolerance capabilities.

As future work, the SAFPA framework can be extended to support faults in control signals, thereby facilitating a more thorough analysis of the reliability of systolic-array-based accelerators. The FORTALESA-based ASIC will be used for testing at radiation facilities and aging chambers. Additionally, a more sophisticated ASIC will be developed, featuring a RISC-V-based SoC integrated with a FORTALESA systolic array.

List of Figures

1 Multi-layer perceptron 20
2 CNN feature visualization (output of the first layer of ResNet-50, the first 16 channels) 21
3 Architecture of a Vision Transformer [50] 22
4 Attention map visualization (adapted from [67]) 22
5 Systolic array and individual PE architecture 24
6 Typical structure of modern FPGA-based SoC 31
7 Fault tolerance taxonomy for SoC FPGA based on vendor-provided methods 39
8 Error patterns in the output tensor for different systolic array dataflows 43
9 Workflow of the fault injection process 49
10 Reliability assessment of ResNet-18 on different systolic arrays considering transient faults 50
11 Layer-wise fault distribution for ResNet-18 51
12 Architecture of the reconfigurable systolic array 52
13 Proposed algorithm to determine the RSA configuration for each layer by balancing reliability and performance (RP strategy) 53
14 Layer-wise AVF and efficiency E comparison for VGG-19, with different RSA configuration strategies 53
15 Model-wise AVF and efficiency E comparison with different RSA configuration strategies 55
16 Layer- and operation-wise vulnerability assessment of ViT-B16 considering transient faults (128×128 systolic array) 59
17 Layer- and operation-wise vulnerability assessment of tested vision transformers considering transient faults 60
18 Layer-wise vulnerability assessment of ResNet-50 considering transient faults (128×128 systolic array) 60
19 Layer- and operation-wise vulnerability comparison 61
20 Embeddings and encoder blocks vulnerability 61
21 Layer- and operation-wise fault distribution 62
22 Arrangement of the systolic array in the dual-redundancy grouping execution mode 66
23 Arrangement of the systolic array in the triple-redundancy grouping execution mode, implementation option with groups of three PEs (TRG3) 67
24 Arrangement of the systolic array in the triple-redundancy grouping execution mode, implementation option with groups of four PEs (TRG4) 68
25 Layer-wise reliability assessment of AlexNet considering transient faults ... 70
26 Layer-wise reliability assessment of VGG-11 considering transient faults 70
27 Layer-wise reliability assessment of ResNet-18 considering transient faults . 71
28 Reliability assessment of AlexNet considering permanent faults (stuck-at-1) 71
29 Reliability vs. latency and energy for AlexNet 72
30 Reliability vs. latency and energy for VGG-11 73
31 Reliability vs. latency and energy for ResNet-18 73
32 Comparison of FORTALESA implementation options considering reliability . 74
33 Comparison of FORTALESA with static TMR and selective ECC [184] 74
34 Block diagram of the FORTALESA-based ASIC 75
35 Layout of the FORTALESA-based ASIC and photo of the produced chip 76

36 Comparison of absolute error distribution between the original 8-bit Mitchell's multiplier and AdAM: the top point shows the maximum absolute error for AdAM, and the lower point shows the maximum absolute error for Mitchell's multiplier 79
37 AdAM architecture (the contributions and extensions to the logarithmic Mitchell multiplier are marked with red color) 80
38 Adaptive adder architecture: A and B are inputs, C is carry values, and PFA stands for partial full adder, m here denotes the size of the truncated mantissa $(n-1-t)$. The truncation parameter t defines the minimum number of protected bits; t PFAs that duplicate the addition of t higher-order bits of the mantissa are colored blue. The duplication level h defines the maximum number of protected bits; $h-t$ PFAs with multiplexers that can duplicate the addition of higher-order bits or add lower-order bits are colored green. 81
39 Fault-tolerance and error introduced based on different input values ($n=8, t=2, h=3$) 82
40 Gate-level structure of the proposed LOD 83
41 Hardware efficiency (area) and fault resilience (fault coverage considering SDC-1 and SDC-10%) trade-offs in different benchmarks. Unp-Exact: unprotected exact multiplier, Unp-AxM: unprotected approximate multiplier, Pro-TMR: exact multiplier protected with TMR, Pro-AdAM: proposed multiplier 87
42 PDP and vulnerability tradeoffs (considering different SDCs) in different benchmarks. Unp-exact: unprotected exact multiplier, Unp-AxM: unprotected approximated multiplier, Pro-TMR: exact multiplier protected with TMR, Pro-AdAM: proposed multiplier 88

List of Tables

1 Faults and fault effects in SoC FPGA 33
2 Vendor-provided fault tolerance mechanisms in different platforms 36
3 Transient fault parameters 44
4 Permanent fault parameters 48
5 Comparison of reliability assessment methods 50
6 Systolic array size selected for each layer by each strategy for VGG-19 54
7 Parameters of the tested vision transformers 58
8 Execution modes (run-time) and implementation options (design-time) 69
9 Hardware implementation parameters for 48×48 systolic array 69
10 Hardware implementation parameters for 132×132 systolic array 69
11 Comparison with the state-of-the-art 75
12 LOD hardware comparison between the proposed method and the state-of-the-art 84
13 Accuracy and efficiency of 8-bit approximate multipliers compared with AdAM 84
14 Accuracy and efficiency of 16-bit approximate multipliers compared with AdAM 85
15 Accuracy and efficiency of 32-bit approximate multipliers compared with AdAM 85
16 Efficiency of 8-bit MAC unit of a systolic array with different multipliers 85
17 Accuracy comparison of different CNNs with an exact (baseline) and the proposed approximate multiplier 86
18 Fault coverage (SDC-5%) in different benchmarks (%) 88
19 Fault coverage (SDC-20%) in different benchmarks (%) 88

References

[1] K. Abdelouahab, M. Pelcat, J. Serot, and F. Berry. Accelerating CNN inference on FPGAs: a survey. *arXiv preprint arXiv:1806.01683*, 2018.

[2] U. K. Agarwal, A. Chan, A. Asgari, and K. Pattabiraman. Towards reliability assessment of systolic arrays against stuck-at faults. In *53rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks - Supplemental Volume, DSN-S*, pages 230–236, 2023.

[3] M. H. Ahmadilivani, M. Barbareschi, S. Barone, A. Bosio, M. Daneshtalab, S. Della Torca, G. Gavarini, M. Jenihhin, J. Raik, A. Ruospo, et al. Special session: Approximation and fault resiliency of DNN accelerators. In *2023 IEEE 41st VLSI Test Symposium (VTS)*, pages 1–10, 2023.

[4] M. H. Ahmadilivani, N. Cherezova, M. Glaß, J.-D. Guerrero-Balaguera, M. Jenihhin, A. Kritikakou, R. L. Sierra, L. B. Poelhs, J. Raik, L. Roquet, F. F. dos Santos, F. A. da Silva, M. S. Reorda, A. Veronesi, E. C. V. Castillo, and J. E. R. Condia. Reliability assessment of deep neural networks and accelerators across design stages. In *IEEE 27th Latin American Test Symposium (LATS)*, pages 1–10, 2026.

[5] 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)*, pages 1–5, 2023.

[6] M. H. Ahmadilivani, M. Taheri, J. Raik, M. Daneshtalab, and M. Jenihhin. A systematic literature review on hardware reliability assessment methods for deep neural networks. *ACM Computing Surveys*, 56(6):1–39, 2024.

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

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

[9] J. D. Anderson, J. C. Leavitt, and M. J. Wirthlin. Neutron radiation beam results for the Xilinx UltraScale+ MPSoC. In *2018 IEEE Nuclear & Space Radiation Effects Conference (NSREC 2018)*, pages 1–7, 7 2018.

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

[11] J. M. Badia, I. Martin-Salinas, G. Leon, A. Amor-Martin, L. Frias-Dominguez, J. A. Belloch, M. Garcia-Valderas, A. Lindoso, C. Cazzaniga, and L. Entrena. Reliability of vision transformers and CNNs on edge AI systems under neutron radiation. *IEEE Transactions on Nuclear Science*, 72(8):2706–2716, 2025.

[12] C. Bagavathi and O. Saraniya. Evolutionary mapping techniques for systolic computing system. In A. K. Sangaiah, editor, *Deep Learning and Parallel Computing Environment for Bioengineering Systems*, pages 207–223. Academic Press, 2019.

[13] R. Baumann. Soft errors in advanced computer systems. *IEEE Design and Test of Computers*, 22:258–266, 5 2005.

[14] L. A. Benites, F. Benevenuti, A. B. De Oliveira, F. L. Kastensmidt, N. Added, V. A. Aguiar, N. H. Medina, and M. A. Guazzelli. Reliability calculation with respect to functional failures induced by radiation in TMR ARM Cortex-M0 soft-core embedded into SRAM-based FPGA. *IEEE Transactions on Nuclear Science*, 66(7):1433–1440, jul 2019.

[15] C. Bernardeschi, L. Cassano, A. Domenici, and L. Sterpone. ASSESS: A simulator of soft errors in the configuration memory of SRAM-based FPGAs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 33:1342–1355, 2014.

[16] A. Berroukham, K. Housni, and M. Lahraichi. Vision transformers: A review of architecture, applications, and future directions. In *2023 7th IEEE Congress on Information Science and Technology (CiSt)*, pages 205–210, 2023.

[17] T. G. Bertoa, G. Gambardella, N. J. Fraser, M. Blott, and J. McAllister. Fault-tolerant neural network accelerators with selective TMR. *IEEE Design and Test*, 40:67–74, 4 2023.

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

[19] P. Bodmann, G. Papadimitriou, D. Gizopoulos, and P. Rech. The impact of SoC integration and OS deployment on the reliability of Arm processors. In *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 223–225, 3 2021.

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

[21] C. Bolchini, L. Cassano, A. Miele, and A. Toschi. Fast and accurate error simulation for CNNs against soft errors. *IEEE Transactions on Computers*, 72:984–997, 4 2023.

[22] A. Bosio, D. Ménard, and O. Sentieys, editors. *Approximate Computing Techniques*. Springer International Publishing, 2022.

[23] J. Budroweit and H. Patscheider. Risk assessment for the use of COTS devices in space systems under consideration of radiation effects. *Electronics*, 10(9), 2021.

[24] 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.

[25] K. Chellapilla, S. Puri, and P. Simard. High performance convolutional neural networks for document processing. In *10th International Workshop on Frontiers in Handwriting Recognition*, 2006.

[26] K. Chen, W. Liu, and F. Lombardi. Approximate arithmetic circuits: Design and applications. In W. Liu and F. Lombardi, editors, *Approximate Computing*, pages 3–21. Springer International Publishing, Cham, 2022.

[27] Y. Chen, H. Chen, X. Zhang, and P. Lai. Failure localization and mechanism analysis in system-on-chip (SOC) using advanced failure analysis techniques. In *13th International Conference on Electronic Packaging Technology and High Density Packaging (ICEPT-HDP)*, pages 1348–1351, 2012.

[28] Y. Chen, T. Li, X. Chen, Z. Cai, and T. Su. High-frequency systolic array-based transformer accelerator on field programmable gate arrays. *Electronics*, 12(4), 2023.

[29] Y.-H. Chen, J. Emer, and V. Sze. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. In *ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, pages 367–379, 2016.

[30] Y.-H. Chen, T. Krishna, J. Emer, and V. Sze. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. In *IEEE International Solid-State Circuits Conference, ISSCC 2016, Digest of Technical Papers*, pages 262–263, 2016.

[31] Z. Chen, G. Li, and K. Pattabiraman. 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)*, pages 1–13, 2021.

[32] E. Cheng, S. Mirkhani, L. G. Szafaryn, C.-Y. Cher, H. Cho, K. Skadron, M. R. Stan, K. Lilja, J. A. Abraham, P. Bose, and S. Mitra. CLEAR: Cross-layer exploration for architecting resilience - combining hardware and software techniques to tolerate soft errors in processor cores. In *Design Automation Conference (DAC)*, 4 2016.

[33] N. Cherezova, M. Jenihhin, and A. Jutman. IJTAG-compatible symptom-based SEU monitors for FPGA DNN accelerators. In *2024 IEEE International Conference on Design, Test and Technology of Integrated Systems (DTTIS)*, pages 1–6, 2024.

[34] N. Cherezova, M. Jenihhin, and A. Jutman. Scalable reliability assessment of vision transformers on systolic arrays via fault propagation analysis. In *IEEE 32nd International Symposium on On-Line Testing and Robust System Design (IOLTS)*, pages 1–7, 2026.

[35] N. Cherezova, A. Jutman, and M. Jenihhin. FORTALESA: Fault-tolerant reconfigurable systolic array for DNN inference. *Microprocessors and Microsystems*, 119:105222, 12 2025.

[36] N. Cherezova, S. Pappalardo, A. Ruospo, B. Deveautour, L. Fezza, A. Jutman, E. Sanchez, A. Bosio, M. S. Reorda, and M. Jenihhin. Special session: Reliability assessment of DNN models and inference on systolic arrays. In *IEEE VLSI Test Symposium (VTS)*, 2026.

[37] 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 *2024 IFIP/IEEE 32nd International Conference on Very Large Scale Integration (VLSI-SoC)*, pages 1–4, 2024.

[38] N. Cherezova, K. Shibin, M. Jenihhin, and A. Jutman. Understanding fault-tolerance vulnerabilities in advanced SoC FPGAs for critical applications. *Microelectronics Reliability*, 146:115010, 2023.

[39] H. Cho, E. Cheng, T. Shepherd, C. Y. Cher, and S. Mitra. System-level effects of soft errors in uncore components. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 36:1497–1510, 9 2017.

[40] H. Cho, C. Y. Cher, T. Shepherd, and S. Mitra. Understanding soft errors in uncore components. In *Design Automation Conference (DAC)*, 7 2015.

[41] G. Cora, E. Vacca, C. D. Sio, S. Azimi, and L. Sterpone. Repair: Reconfigurable platform for ai resilience within risc-v ecosystem. *Lecture Notes in Computer Science*, 15594 LNCS:71–87, 2025.

[42] D. Danopoulos, G. Zervakis, K. Siozios, D. Soudris, and J. Henkel. AdaPT: Fast emulation of approximate DNN accelerators in PyTorch. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2022.

[43] A. B. De Oliveira, G. S. Rodrigues, F. L. Kastensmidt, N. Added, E. L. Macchione, V. A. Aguiar, N. H. Medina, and M. A. Silveira. Lockstep dual-core ARM A9: implementation and resilience analysis under heavy ion-induced soft errors. *IEEE Transactions on Nuclear Science*, 65(8):1783–1790, aug 2018.

[44] J. Deng, Y. Fang, Z. Du, Y. Wang, H. Li, O. Temam, P. Ienne, D. Novo, X. Li, Y. Chen, and C. Wu. Retraining-based timing error mitigation for hardware neural networks. In *Design, Automation and Test in Europe Conference (DATE)*, pages 593–596, 2015.

[45] 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.

[46] S. Devadze, C. E. Nielsen, N. Cherezova, D. Mihhailov, and P. Ellervee. Architectural exploration and implementation of CERN LHC trigger algorithm with FPGA. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 33(11):2998–3007, 2025.

[47] B. Deveautour, M. Traiola, A. Virazel, and P. Girard. QAMR: An approximation-based fully reliable TMR alternative for area overhead reduction. In *2020 IEEE European test symposium (ETS)*, pages 1–6. IEEE, 2020.

[48] H. D. Dixit, S. Pendharkar, M. Beadon, C. Mason, T. Chakravarthy, B. Muthiah, and S. Sankar. Silent data corruptions at scale, 2021.

[49] F. F. Dos Santos, P. Rech, A. Kritikakou, and O. Sentieys. Neutron-induced error rate of vision transformer models on GPUs. In *RADECS*, pages 1–4, 2023.

[50] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *ICLR*, 2021.

[51] X. Du, C. He, S. Liu, D. Luo, X. Du, W. Yang, Y. Li, and Y. Fan. Analysis of sensitive blocks of soft errors in the Xilinx Zynq-7000 system-on-chip. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 940:125–128, 10 2019.

[52] X. Du, C. He, S. Liu, Y. Zhang, Y. Li, and W. Yang. Measurement of single event effects induced by alpha particles in the Xilinx Zynq-7010 System-on-Chip. *Journal of Nuclear Science and Technology*, 54:1–6, 12 2016.

[53] J. J. Elliott, F. Mueller, M. K. Stoyanov, and C. G. Webster. Quantifying the impact of single bit flips on floating point arithmetic. Technical report, Oak Ridge National Laboratory (ORNL), Oak Ridge, TN (United States), 08 2013.

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

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

[56] S. T. Fleming, D. B. Thomas, and F. Winterstein. A power-aware adaptive FDIR framework using heterogeneous system-on-chip modules. In F. Kastensmidt and P. Rech, editors, *FPGAs and Parallel Architectures for Aerospace Applications: Soft Errors and Fault-Tolerant Design*, pages 75–90. Springer International Publishing, 2016.

[57] S. Fouad, F. Ghaffari, M. E. A. Benkhelifa, and B. Granado. Context-aware resources placement for SRAM-based FPGA to minimize checkpoint/recovery overhead. In *2014 International Conference on Reconfigurable Computing and FPGAs (ReConFig)*, feb 2014.

[58] C. M. Fuchs, K. Marinis, P. Chou, X. Wen, N. M. Murillo, G. Furano, S. Holst, A. Tavoularis, S.-K. Lu, and A. Plaat. A fault-tolerant MPSoC for CubeSats. In *2019 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, pages 1–6. IEEE, 10 2019.

[59] C. M. Fuchs, N. M. Murillo, A. Plaat, E. van der Kouwe, D. Harsono, and P. Wang. Software-defined dependable computing for spacecraft. In *2018 IEEE 23rd Pacific Rim International Symposium on Dependable Computing (PRDC)*, 12 2018.

[60] G. Gavarini, A. Ruospo, and E. Sanchez. Evaluation and mitigation of faults affecting Swin transformers. In *IEEE 29th International Symposium on On-Line Testing and Robust System Design (IOLTS)*, 2023.

[61] F. Geissler, S. Qutub, M. Paulitsch, and K. Pattabiraman. A low-cost strategic monitoring approach for scalable and interpretable error detection in deep neural networks. In *Computer Safety, Reliability, and Security: 42nd International Conference, SAFECOMP*, page 75–88, 2023.

[62] H. Genc, S. Kim, A. Amid, A. Haj-Ali, V. Iyer, P. Prakash, J. Zhao, D. Grubb, H. Liew, H. Mao, A. Ou, C. Schmidt, S. Steffl, J. Wright, I. Stoica, J. Ragan-Kelley, K. Asanovic, B. Nikolic, and Y. S. Shao. Gemmini: Enabling systematic deep-learning architecture evaluation via full-stack integration. In *58th Annual Design Automation Conference (DAC)*, 2021.

[63] R. Glein, F. Rittner, A. Becher, D. Ziener, J. Frickel, J. Teich, and A. Heuberger. Reliability of space-grade vs. COTS SRAM-based FPGA in N-modular redundancy. In *2015 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, pages 1–8, 2015.

[64] B. F. Goldstein, V. C. Ferreira, S. Srinivasan, D. Das, A. S. Nery, S. Kundu, and F. M. Franca. A lightweight error-resiliency mechanism for deep neural networks. In *International Symposium on Quality Electronic Design (ISQED)*, pages 311–316, 4 2021.

[65] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016.

[66] P. S. Graham, M. P. Caffrey, J. A. Zimmerman, P. Sundararajan, E. J. Johnson, and C. D. Patterson. Consequences and categories of SRAM FPGA configuration SEUs. In *Int. Conf. MAPLD*, 9 2003.

[67] J. Guo, N. Jia, and J. Bai. Transformer based on channel-spatial attention for accurate classification of scenes in remote sensing image. *Scientific Reports*, 12, 2022.

[68] K. Guo, S. Zeng, J. Yu, Y. Wang, and H. Yang. A survey of FPGA-based neural network inference accelerators. *ACM Trans. on Reconfigurable Technology and Systems (TRETS)*, 12(1):1–26, 2019.

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

[70] M. A. Hanif and M. Shafique. Dependable Deep Learning: Towards Cost-Efficient Resilience of Deep Neural Network Accelerators against Soft Errors and Permanent Faults. In *2020 26th IEEE International Symposium on On-Line Testing and Robust System Design (IOLTS)*, jul 2020.

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

[72] J. He, Y. Liu, C. Xu, X. Liao, and Y. Yang. Fine-grained fault sensitivity analysis of vision transformers under soft errors. *Electronics*, 14:2418, 2025.

[73] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[74] 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 *Design, Automation and Test in Europe Conference and Exhibition (DATE)*, pages 1241–1246, mar 2020.

[75] P. Hochschild, P. Turner, J. Mogul, R. Govindaraju, P. Ranganathan, D. Culler, and A. Vahdat. Cores that don't count. In *HotOS'21: Workshop on Hot Topics in Operating Systems*, pages 9–16, 06 2021.

[76] K. A. Hoque, O. A. Mohamed, Y. Savaria, and C. Thibeault. Probabilistic model checking based DAL analysis to optimize a combined TMR-blind-scrubbing mitigation technique for FPGA-based aerospace applications. In *12th ACM/IEEE International Conference on Methods and Models for System Design (MEMOCODE)*, pages 175–184, 11 2014.

[77] T. Horita, T. Murata, and I. Takanami. A multiple-weight-and-neuron-fault tolerant digital multilayer neural network. In *IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, pages 554–562, 2006.

[78] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. MobileNets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[79] K. Höflinger, S. Müller, T. Peng, M. Ulmer, D. Lüdtke, and A. Gerndt. Dynamic fault tree analysis for a distributed onboard computer. *IEEE Aerospace Conference Proceedings*, 2019-March, 3 2019.

[80] 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, 12 2020.

[81] M. Ibtesam, U. S. Solangi, J. Kim, M. A. Ansari, and S. Park. Highly efficient test architecture for low-power ai accelerators. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 41:2728–2738, 8 2022.

[82] Intel. *Intel Agilex FPGAs and SoCs Device Overview*, 12 2021.

[83] Intel. *Intel Agilex Hard Processor System Technical Reference Manual*, 11 2021.

[84] Intel. *Intel Agilex SEU Mitigation User Guide*, 10 2021.

[85] Intel. *Intel Stratix 10 GX/SX Device Overview*, 04 2021.

[86] Intel. *Intel Stratix 10 SEU Mitigation User Guide*, 10 2021.

[87] X. Iturbe, B. Venu, and E. Ozer. Soft error vulnerability assessment of the real-time safety-related ARM Cortex-R5 CPU. *2016 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems, DFT 2016*, pages 91–96, 10 2016.

[88] X. Iturbe, B. Venu, E. Ozer, J.-L. Poupat, G. Gimenez, and H.-U. Zurek. The arm triple core lock-step (TCLS) processor. *ACM Transactions on Computer Systems*, 36:1–30, 2018.

[89] JEDEC Solid State Technology Association. *JESD88E. JEDEC Dictionary of Terms for Solid-State Technology, Sixth Edition*, 2013.

[90] M. Jenihhin, M. Taheri, and N. Cherezova. Quantization-aided cost-efficient reliability of CNN accelerators for edge AI. In S. Liu, P. Reviriego, Z. Gao, and F. Lombardi, editors, *Machine Learning Systems: The Role of Hardware Design for Dependable Computing*. Springer Nature Switzerland, 2026.

[91] H. Jiang, F. J. H. Santiago, H. Mo, L. Liu, and J. Han. Approximate arithmetic circuits: A survey, characterization, and recent applications. *Proceedings of the IEEE*, 108(12):2108–2135, 2020.

[92] N. Jing, J.-Y. Lee, Z. Feng, W. He, Z. Mao, and L. He. SEU fault evaluation and characteristics for SRAM-based FPGA architectures and synthesis algorithms. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 18:1–18, 1 2013.

[93] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, P.-l. Cantin, C. Chao, C. Clark, J. Coriell, M. Daley,

M. Dau, J. Dean, B. Gelb, T. V. Ghaemmaghami, R. Gottipati, W. Gulland, R. Hagmann, C. R. Ho, D. Hogberg, J. Hu, R. Hundt, D. Hurt, J. Ibarz, A. Jaffey, A. Jaworski, A. Kaplan, H. Khaitan, D. Killebrew, A. Koch, N. Kumar, S. Lacy, J. Laudon, J. Law, D. Le, C. Leary, Z. Liu, K. Lucke, A. Lundin, G. MacKean, A. Maggiore, M. Mahony, K. Miller, R. Nagarajan, R. Narayanaswami, R. Ni, K. Nix, T. Norrie, M. Omernick, N. Penukonda, A. Phelps, J. Ross, M. Ross, A. Salek, E. Samadiani, C. Severn, G. Sizikov, M. Snelham, J. Souter, D. Steinberg, A. Swing, M. Tan, G. Thorson, B. Tian, H. Toma, E. Tuttle, V. Vasudevan, R. Walter, W. Wang, E. Wilcox, and D. H. Yoon. In-datacenter performance analysis of a tensor processing unit. In *44th Annual International Symposium on Computer Architecture*, page 1–12, 2017.

[94] S. Kasap, E. W. Wachter, X. Zhai, S. Ehsan, and K. D. McDonald-Maier. Novel lockstep-based approach with roll-back and roll-forward recovery to mitigate radiation-induced soft errors. In *2020 IEEE Nordic Circuits and Systems Conference (NORCAS)*, oct 2020.

[95] 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 *Design Automation Conference*, 7 2020.

[96] R. M. Kodamanchili, N. Cherezova, M. Taheri, and M. Jenihhin. Adaptive fault resilience for early-exit DNNs. In *2025 IEEE International Test Conference in Asia (ITC-Asia)*, pages 108–113, 2025.

[97] A. Kritikakou, O. Sentieys, G. Hubert, Y. Helen, J.-F. Coulon, and P. Deroux-Dauphin. FLODAM: Cross-layer reliability analysis flow for complex hardware designs. In *25th IEEE/ACM Design, Automation and Test in Europe (DATE)*, pages 1–6, 3 2022.

[98] A. Krizhevsky. Learning multiple layers of features from tiny images, 2009.

[99] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, 2012.

[100] R. V. Kshirsagar and S. Sharma. Comparative study of the proposed shifting algorithm for fault tolerance in FPGA. In *2014 International Conference on Advances in Engineering and Technology Research (ICAETR)*, 2014.

[101] H. T. Kung. Why systolic architectures? *Computer*, 15(1):37–46, 1982.

[102] F. Lahrach, A. Doumar, and E. Châtelet. Fault tolerance of SRAM-based FPGA via configuration frames. In *2011 IEEE Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS)*, pages 139–142, 2011.

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

[104] Y. LeCun and C. Cortes. The MNIST handwritten digit database.

[105] C.-J. Lee and T. T. Yeh. ReSA: reconfigurable systolic array for multiple tiny dnn tensors. *ACM Transactions on Architecture and Code Optimization*, 37:24, 9 2024.

[106] H. Lee, J. Kim, J. Park, and S. Kang. Strait: Self-test and self-recovery for ai accelerator. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 42:3092–3104, 9 2023.

[107] H. Lee, J. Lee, and S. Kang. A robust test architecture for low-power ai accelerators. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 44:1581–1594, 4 2025.

[108] H. Lee, J. Park, and S. Kang. An area-efficient systolic array redundancy architecture for reliable ai accelerator. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 32:1950–1954, 10 2024.

[109] G. Lentaris, K. Maragos, I. Stratakos, L. Papadopoulos, O. Papanikolaou, D. Soudris, M. Lourakis, X. Zabulis, D. Gonzalez-Arjona, and G. Furano. High-performance embedded computing in space: Evaluation of platforms for vision-based navigation. *Journal of Aerospace Information Systems*, 15(4):178–192, 2018.

[110] R. Leveugle, A. Calvez, P. Maistri, and P. Vanhauwaert. Statistical fault injection: Quantified error and confidence. In *IEEE Design, Automation and Test in Europe Conference and Exhibition (DATE)*, pages 502–506, 2009.

[111] 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 *International Conference for High Performance Computing, Networking, Storage and Analysis*, volume 12, 11 2017.

[112] J. Li and Y. Wang. Transient fault tolerance on multicore processor in AMP mode. In *8th International Conference on Dependable Systems and Their Applications (DSA)*, pages 332–337, 2021.

[113] Y. Li, E. Cheng, S. Makar, and S. Mitra. Self-repair of uncore components in robust system-on-chips: An OpenSPARC T2 case study. In *International Test Conference (ITC)*, 2013.

[114] F. Libano, P. Rech, and J. Brunhaver. Efficient error detection for matrix multiplication with systolic arrays on FPGAs. *IEEE Transactions on Computers*, 72:2390–2403, 8 2023.

[115] 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*, 66:216–222, 1 2019.

[116] F. Lima Kastensmidt, L. Sterpone, L. Carro, and M. Sonza Reorda. On the optimal design of triple modular redundancy logic for SRAM-based FPGAs. In *Design, Automation and Test in Europe (DATE)*, pages 1290–1295, 2005.

[117] J. Liu, Z. Zhu, and C. Deng. A novel and adaptive transient fault-tolerant algorithm considering timing constraint on heterogeneous systems. *IEEE Access*, 8:103047–103061, 2020.

[118] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 9992–10002, 2021.

[119] J. Lojda, J. Podivinsky, and Z. Kotasek. Reliability indicators for automatic design and analysis of fault-tolerant FPGA systems. In *20th IEEE Latin American Test Symposium (LATS)*, may 2019.

[120] 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, 2019.

[121] B. Loureiro Coelho, P. Rafael Bodmann, N. Cavagnero, C. Frost, and P. Rech. Vision transformer reliability evaluation on the Coral Edge TPU. *IEEE Transactions on Nuclear Science*, 72(4), 2025.

[122] W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, 5:115–133, 1943.

[123] Microchip Technology Inc. *PolarFire SoC MSS Technical Reference Manual*, 2021.

[124] Microchip Technology Inc. *PolarFire SoC Product Overview*, 2021.

[125] J. N. Mitchell. Computer multiplication and division using binary logarithms. *IRE Transactions on Electronic Computers*, (4):512–517, 1962.

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

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

[128] V. Mrazek, R. Hrbacek, Z. Vasicek, and L. Sekanina. EvoApprox8b: Library of approximate adders and multipliers for circuit design and benchmarking of approximation methods. In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 258–261, 2017.

[129] S. Mukherjee. *Architecture Design for Soft Errors*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2008.

[130] S. S. Mukherjee, C. Weaver, J. Emer, S. K. Reinhardt, and T. Austin. A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor. *Annual International Symposium on Microarchitecture, MICRO*, 2003-January:29–40, 2003.

[131] H. Mushtaq, Z. Al-Ars, and K. Bertels. Fault tolerance on multicore processors using deterministic multithreading. In *2013 8th IEEE Design and Test Symposium*, 2013.

[132] S. Narayanamoorthy, H. A. Moghaddam, Z. Liu, T. Park, and N. S. Kim. Energy-efficient approximate multiplication for digital signal processing and classification applications. *IEEE transactions on very large scale integration (VLSI) systems*, 23(6):1180–1184, 2014.

[133] T. Nidhin, A. Bhattacharyya, R. Behera, T. Jayanthi, and K. Velusamy. Understanding radiation effects in SRAM-based field programmable gate arrays for implementing instrumentation and control systems of nuclear power plants. *Nuclear Engineering and Technology*, 49:1589–1599, 12 2017.

[134] M. Nourazar, V. Rashtchi, A. Azarpeyvand, and F. Merrikh-Bayat. Code acceleration using memristor-based approximate matrix multiplier: Application to convolutional neural networks. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 26(12):2684–2695, 2018.

[135] A. B. D. Oliveira, L. A. Tambara, F. Benevenuti, L. A. Benites, N. Added, V. A. Aguiar, N. H. Medina, M. A. Silveira, and F. L. Kastensmidt. Evaluating soft core RISC-V processor in SRAM-based FPGA under radiation effects. *IEEE Transactions on Nuclear Science*, 67:1503–1510, 7 2020.

[136] D. A. G. D. Oliveira, L. L. Pilla, T. Santini, and P. Rech. Evaluation and mitigation of radiation-induced soft errors in graphics processing units. *IEEE Transactions on Computers*, 65:791–804, 3 2016.

[137] 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*, 39:3250–3262, 11 2020.

[138] E. Ozen and A. Orailoglu. 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, Digest of Technical Papers, ICCAD*, 11 2020.

[139] G. Papadimitriou and D. Gizopoulos. Demystifying the system vulnerability stack: Transient fault effects across the layers. In *International Symposium on Computer Architecture*, pages 902–915, 6 2021.

[140] D. Park, C. Nicopoulos, J. Kim, N. Vijaykrishnan, and C. R. Das. Exploring fault-tolerant network-on-chip architectures. In *International Conference on Dependable Systems and Networks*, volume 2006, pages 93–104, 2006.

[141] C. Peltekis, D. Filippas, and G. Dimitrakopoulos. Error checking for sparse systolic tensor arrays. In *2024 IEEE 6th International Conference on AI Circuits and Systems (AICAS)*, pages 552–556, 04 2024.

[142] C. Peltekis, D. Filippas, G. Dimitrakopoulos, C. Nicopoulos, and D. Pnevmatikatos. ArrayFlex: A systolic array architecture with configurable transparent pipelining. In *Design, Automation and Test in Europe Conference (DATE)*, 2023.

[143] C. Peltekis, C. Nicopoulos, and G. Dimitrakopoulos. Periodic online testing for sparse systolic tensor arrays. In *2025 14th International Conference on Modern Circuits and Systems Technologies (MOCAST)*, pages 1–6. IEEE, 6 2025.

[144] M. Pignol. DMT and DT2: Two fault-tolerant architectures developed by CNES for COTS-based spacecraft supercomputers. In *12th IEEE International On-Line Testing Symposium (IOLTS)*, pages 203–212, 2006.

[145] V. M. Placinta, L. N. Cojocariu, and C. Ravariu. Proton-induced radiation effects in the I/O blocks of an SRAM-based FPGA. *Journal of Instrumentation*, 14:T10001, 10 2019.

[146] A. Prodromou, A. Panteli, C. Nicopoulos, and Y. Sazeides. NoCAlert: An on-line and real-time fault detection mechanism for network-on-chip architectures. In *IEEE/ACM 45th International Symposium on Microarchitecture (MICRO)*, pages 60–71, 2012.

[147] H. Quinn. Radiation effects in reconfigurable FPGAs. *Semicond. Sci. Technol*, 32, 2017.

[148] P. Rech. Artificial neural networks for space and safety-critical applications: Reliability issues and potential solutions. *IEEE Transactions on Nuclear Science*, 71(4):377–404, 2024.

[149] S. Rehman, M. Shafique, and J. Henkel. *Reliable Software for Unreliable Hardware: A Cross Layer Perspective*. Springer Publishing Company, Incorporated, 2016.

[150] G. Rodrigues, F. L. Kastensmidt, and A. Bosio. Survey on approximate computing and its intrinsic fault tolerance. *Electronics 2020, Vol. 9, Page 557*, 9:557, 3 2020.

[151] M. Rogenmoser, N. Wistoff, P. Vogel, F. Gurkaynak, and L. Benini. On-demand redundancy grouping: Selectable soft-error tolerance for a multicore cluster. In *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 398–401, 2022.

[152] N. Rollins, M. Wirthlin, M. Caffrey, and P. Graham. Reliability of programmable input/output pins in the presence of configuration upsets. In *MAPLD '02 Laurel MD*, 9 2002.

[153] L. Roquet, F. F. D. Santos, P. Rech, M. Traiola, O. Sentieys, and A. Kritikakou. Cross-layer reliability evaluation and efficient hardening of large vision transformers models. In *61st ACM/IEEE Design Automation Conference (DAC)*, 2024.

[154] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65 (6):386–408, 1958.

[155] 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.

[156] 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.

[157] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet large scale visual recognition challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[158] S. Saeedi, A. Carpegna, A. Savino, and S. Di Carlo. Prediction of the impact of approximate computing on spiking neural networks via interval arithmetic. In *2022 IEEE 23rd Latin American Test Symposium (LATS)*, pages 1–6. IEEE, 2022.

[159] M. Safarpour, R. Inanlou, and O. Silvén. Algorithm level error detection in low voltage systolic array. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 69:569–573, 2 2022.

[160] S. S. Sahoo, B. Veeravalli, and A. Kumar. CL(R)Early: An early-stage DSE methodology for cross-layer reliability-aware heterogeneous embedded systems. In *Design Automation Conference (DAC)*, 7 2020.

[161] A. Samajdar, J. M. Joseph, Y. Zhu, P. Whatmough, M. Mattina, and T. Krishna. A systematic methodology for characterizing scalability of DNN accelerators using SCALE-Sim. In *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 58–68, 8 2020.

[162] 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*, 68:663–677, 6 2019.

[163] S. Sapkal, U. Zahid, G. Gambardella, H.-G. Stratigopoulos, and B. Clerkin. On the fault sensitivity of natural language embeddings computation. In *IEEE European Test Symposium (ETS)*, 2025.

[164] A. Savino, A. Vallero, and S. D. Carlo. ReDO: Cross-layer multi-objective design-exploration framework for efficient soft error resilient systems. *IEEE Transactions on Computers*, 67:1462–1477, 10 2018.

[165] L. Sayadi, S. Timarchi, and A. Sheikh-Akbari. Two efficient approximate unsigned multipliers by developing new configuration for approximate 4:2 compressors. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 70(4):1649–1659, 2023.

[166] J.-F. Schulte, B. Ramhorst, C. Sun, J. Mitrevski, N. Ghielmetti, E. Lupi, D. Danopoulos, V. Loncar, J. Duarte, D. Burnette, L. Laatu, S. Tzelepis, K. Axiotis, Q. Berthet, H. Wang, P. White, S. Demirsoy, M. Colombo, T. Aarrestad, S. Summers, M. Pierini, G. D. Guglielmo, J. Ngadiuba, J. Campos, B. Hawks, A. Gandrakota, F. Fahim, N. Tran, G. Constantinides, Z. Que, W. Luk, A. Tapper, D. Hoang, N. Paladino, P. Harris, B.-C. Lai, M. Valentin, R. Forelli, S. Ogrenci, L. Gerlach, R. Flynn, M. Liu, D. Diaz, E. Khoda, M. Quinnan, R. Solares, S. Parajuli, M. Neubauer, C. Herwig, H. F. Tsoi, D. Rankin, S.-C. Hsu, and S. Hauck. hls4ml: A flexible, open-source platform for deep learning acceleration on reconfigurable hardware, 2025.

[167] K. Shibin, S. Devadze, A. Jutman, M. Grabmann, and R. Pricken. Health management for self-aware SoCs based on IEEE 1687 infrastructure. *IEEE Design & Test*, 34(6):27–35, 2017.

[168] A. Shye, J. Blomstedt, T. Moseley, V. J. Reddi, and D. A. Connors. PLR: A software approach to transient fault tolerance for multicore architectures. *IEEE Transactions on Dependable and Secure Computing*, 6(2):135–148, 2009.

[169] 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.

[170] A. Siddique and K. A. Hoque. Exposing reliability degradation and mitigation in approximate dnns under permanent faults. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 31(4):555–566, 2023.

[171] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations (ICLR)*, 2015.

[172] C. W. Slayman. Cache and memory error detection, correction, and reduction techniques for terrestrial servers and workstations. *IEEE Transactions on Device and Materials Reliability*, 5:397–404, 9 2005.

[173] H. So, M. Didehban, A. Shrivastava, and K. Lee. A software-level redundant multithreading for soft/hard error detection and recovery. In *Design, Automation and Test in Europe Conference and Exhibition (DATE)*, pages 1559–1562, may 2019.

[174] Q. Song, Y. Dai, H. Lu, and G. Jin. High-throughput systolic array-based accelerator for hybrid transformer-CNN networks. *Journal of King Saud University - Computer and Information Sciences*, 36(8):102194, 2024.

[175] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, 2015.

[176] M. Taheri, N. Cherezova, M. S. Ansari, M. Jenihhin, A. Mahani, M. Daneshtalab, and J. Raik. Exploration of activation fault reliability in quantized systolic array-based DNN accelerators. In *2024 25th International Symposium on Quality Electronic Design (ISQED)*, pages 1–8, 2024.

[177] M. Taheri, N. Cherezova, S. Nazari, A. Azarpeyvand, T. Ghasempouri, M. Daneshtalab, J. Raik, and M. Jenihhin. AdAM: Adaptive approximate multiplier for fault tolerance in DNN accelerators. *IEEE Transactions on Device and Materials Reliability*, 25(1):66–75, 2025.

[178] 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.

[179] 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 *27th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS)*, pages 19–24, 2024.

[180] M. Taheri, P. Patne, N. Cherezova, A. Mahani, C. Herglotz, and M. Jenihhin. RL-agent-based early-exit DNN architecture search framework. In *2025 IEEE 28th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS)*, pages 145–148, 2025.

[181] M. Taheri, M. Riazati, M. H. Ahmadilivani, M. Jenihhin, M. Daneshtalab, J. Raik, M. Sjodin, and B. Lisper. DeepAxe: a framework for exploration of approximation and reliability trade-offs in DNN accelerators. In *2023 24th International Symposium on Quality Electronic Design (ISQED)*, pages 1–8, 2023.

[182] L. A. Tambara, F. L. Kastensmidt, P. Rech, F. Lins, N. H. Medina, N. Added, V. A. Aguiar, and M. A. Silveira. Reliability-performance analysis of hardware and software co-designs in SRAM-Based APSoCs. *IEEE Transactions on Nuclear Science*, 65:1935–1942, 8 2018.

[183] L. A. Tambara, P. Rech, E. Chielle, J. Tonfat, and F. L. Kastensmidt. Analyzing the impact of radiation-induced failures in programmable SoCs. *IEEE Transactions on Nuclear Science*, 63:2217–2224, 8 2016.

[184] J. Tan, Q. Wang, K. Yan, X. Wei, and X. Fu. Saca-FI: A microarchitecture-level fault injection framework for reliability analysis of systolic array based CNN accelerator. *Future Generation Computer Systems*, 147:251–264, 10 2023.

[185] F. Z. Tazi, C. Thibeault, and Y. Savaria. Detailed analysis of radiation-induced delays on I/O blocks of an SRAM-based FPGA. In *Canadian Conference on Electrical and Computer Engineering*, 10 2016.

[186] F. Z. Tazi, C. Thibeault, Y. Savaria, S. Pichette, and Y. Audet. On extra delays affecting I/O blocks of an SRAM-based FPGA due to ionizing radiation. *IEEE Transactions on Nuclear Science*, 61:3138–3145, 12 2014.

[187] C. Thibeault, S. Pichette, Y. Audet, Y. Savaria, H. Rufenacht, E. Gloutnay, Y. Blaquiere, F. Moupfouma, and N. Batani. On extra combinational delays in SRAM FPGAs due to transient ionizing radiations. *IEEE Transactions on Nuclear Science*, 59:2959–2965, 12 2012.

[188] H. Touvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles, and H. Jegou. Training data-efficient image transformers & distillation through attention. In *International Conference on Machine Learning*, pages 10347–10357, 2021.

[189] M. Traiola, B. Deveautour, A. Bosio, P. Girard, and A. Virazel. *Test and Reliability of Approximate Hardware. Approximate Computing*, pages 233–266. Springer International Publishing, 2022.

[190] M. Traiola, J. Echavarria, A. Bosio, J. Teich, and I. O'Connor. Design space exploration of approximation-based quadruple modular redundancy circuits. In *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pages 1–9. IEEE, 2021.

[191] R. Travessini, P. R. Villa, F. L. Vargas, and E. A. Bezerra. Processor core profiling for SEU effect analysis. In *IEEE 19th Latin-American Test Symposium (LATS)*, pages 1–6, 4 2018.

[192] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre, and K. Vissers. FINN: A framework for fast, scalable binarized neural network inference. In *ACM/SIGDA international symposium on field-programmable gate arrays*, pages 65–74, 2017.

[193] E. Vacca, G. Ajmone, and L. Sterpone. RunSAFER: A novel runtime fault detection approach for systolic array accelerators. In *The 41st IEEE International Conference on Computer Design*, 2023.

[194] E. Vacca, S. Azimi, and L. Sterpone. A comprehensive analysis of transient errors on systolic arrays. In *26th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS)*, pages 175–180, 2023.

[195] S. Vahdat, M. Kamal, A. Afzali-Kusha, and M. Pedram. LETAM: A low energy truncation-based approximate multiplier. *Computers & Electrical Engineering*, 63:1–17, 2017.

[196] S. Vahdat, M. Kamal, A. Afzali-Kusha, and M. Pedram. TOSAM: An energy-efficient truncation-and rounding-based scalable approximate multiplier. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 27(5):1161–1173, 2019.

[197] A. Vallero, A. Savino, A. Chatzidimitriou, M. Kaliorakis, M. Kooli, M. Riera, M. Anglada, G. D. Natale, A. Bosio, R. Canal, A. Gonzalez, D. Gizopoulos, R. Mariani, and S. D. Carlo. SyRA: Early system reliability analysis for cross-layer soft errors resilience in memory arrays of microprocessor systems. *IEEE Transactions on Computers*, 68:765–783, 5 2019.

[198] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NIPS)*, 2017.

[199] R. Velazco and F. J. Franco. Single event effects on digital integrated circuits: Origins and mitigation techniques. In *IEEE International Symposium on Industrial Electronics*, pages 3322–3327, 2007.

[200] S. I. Venieris and C.-S. Bouganis. fpgaConvNet: mapping regular and irregular convolutional neural networks on FPGAs. *IEEE Trans. on neural networks and learning systems*, 30(2):326–342, 2018.

[201] S. I. Venieris, A. Kouris, and C.-S. Bouganis. Toolflows for mapping convolutional neural networks on FPGAs: A survey and future directions. *ACM Computing Surveys*, 51(3), jun 2018.

[202] J. Vidmar, P. Maillard, T. Jones, M. Sawant, G. Gambardella, and N. Fraser. Space DPU: Constructing a radiation-tolerant, FPGA-based platform for deep learning acceleration on space payloads. In *2nd European Workshop on On-Board Data Processing (OBDP2021)*, 6 2021.

[203] S. Wang, G. Zhang, J. Wei, Y. Wang, J. Wu, and Q. Luo. Understanding silent data corruptions in a large production CPU population. In *Proceedings of the 29th Symposium on Operating Systems Principles*, SOSP '23, page 216–230, 2023.

[204] Z. Wang and A. Chattopadhyay. *High-level Estimation and Exploration of Reliability for Multi-Processor System-on-Chip*. Springer, 2017.

[205] P. Wu, Q. Guan, N. DeBardeleben, S. Blanchard, D. Tao, X. Liang, J. Chen, and Z. Chen. Towards practical algorithm based fault tolerance in dense linear algebra. In *25th ACM International Symposium on High-Performance Parallel and Distributed Computing (HPDC)*, pages 31–42, 5 2016.

[206] Xilinx, Inc. *Versal: The First Adaptive Compute Acceleration Platform (ACAP)*, 9 2020.

[207] Xilinx, Inc. *Zynq UltraScale+ Device Technical Reference Manual*, 12 2020.

[208] Xilinx, Inc. *Versal ACAP AI Engine Architecture Manual*, 10 2021.

[209] Xilinx, Inc. *Versal ACAP Programmable Network on Chip and Integrated Memory Controller*, 11 2021.

[210] Xilinx, Inc. *Versal ACAP Technical Reference Manual*, 10 2021.

[211] Xilinx, Inc. *Zynq UltraScale+ MPSoC Data Sheet: Overview*, 5 2021.

[212] D. Xu, K. Xing, C. Liu, Y. Wang, Y. Dai, L. Cheng, H. Li, and L. Zhang. Resilient neural network training for accelerators with computing errors. In *IEEE 30th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, pages 99–102, 2019.

[213] R. Xu, S. Ma, Y. Wang, and Y. Guo. HeSA: heterogeneous systolic array architecture for compact CNNs hardware accelerators. In *Design, Automation and Test in Europe (DATE)*, pages 657–662, 2 2021.

[214] X. Xue, C. Liu, Y. Wang, B. Yang, T. Luo, L. Zhang, H. Li, and X. Li. Soft error reliability analysis of vision transformers. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 31(12):2126–2136, 2023.

[215] J. Yang, H. Guo, L. Chen, Y. Zhang, and X. Li. SEU sensitivity evaluation of JTAG circuit used for SRAM-based FPGA. In *2017 IEEE 24th International Symposium on the Physical and Failure Analysis of Integrated Circuits (IPFA)*, pages 1–5. IEEE, 7 2017.

[216] W. Yang, X. Du, C. He, S. Shi, L. Cai, N. Hui, G. Guo, and C. Huang. Microbeam heavy-ion single-event effect on Xilinx 28-nm system on chip. *IEEE Transactions on Nuclear Science*, 65:545–549, 1 2018.

[217] S. Yin, P. Ouyang, S. Tang, F. Tu, X. Li, S. Zheng, T. Lu, J. Gu, L. Liu, and S. Wei. A high energy efficient reconfigurable hybrid neural network processor for deep learning applications. *IEEE Journal of Solid-State Circuits*, 53:968–982, 4 2018.

[218] J. Young Ryu, H. Lee, and H.-T. Lee. Detect and avoid AI system model using a deep neural network. In *2022 IEEE/AIAA 41st Digital Avionics Systems Conference (DASC)*, pages 1–8, 2022.

[219] U. Zahid, G. Gambardella, N. J. Fraser, M. Blott, and K. Vissers. FAT: Training neural networks for reliable inference under hardware faults. In *International Test Conference*, 11 2020.

[220] R. Zendegani, M. Kamal, M. Bahadori, A. Afzali-Kusha, and M. Pedram. RoBA multiplier: A rounding-based approximate multiplier for high-speed yet energy-efficient digital signal processing. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 25(2):393–401, 2016.

[221] J. J. Zhang, K. Basu, and S. Garg. Fault-tolerant systolic array based accelerators for deep neural network execution. *IEEE Design and Test*, 36:44–53, 10 2019.

[222] 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, jul 2021.

[223] T. Zhao, S. Miao, S. Lu, J. Cao, J. Qiu, X. Shi, K. Wang, and L. He. Towards a reconfigurable systolic array with multi-level packing for transformers. In *Architecture and System Support for Transformer Models (ASSYST@ ISCA 2023)*, 2023.

[224] Y. Zhao, K. Wang, and A. Louri. FSA: An efficient fault-tolerant systolic array-based DNN accelerator architecture. In *IEEE International Conference on Computer Design: VLSI in Computers and Processors*, pages 545–552, 2022.

Acknowledgements

This dissertation is supported in part by the European Union under Horizon Europe Grant Agreement No. 101160182 “TAICHIP” and by the Estonian Research Council under grants PRG1467 “CRASHLESS”.

Abstract
Cross-Layer Reliability Evaluation and Fault Tolerance for Deep Neural Network Hardware Accelerators

Recent years have been marked by a tremendous increase in the adoption of Machine Learning (ML) algorithms, specifically Deep Neural Networks (DNNs), and their deployment in mission- and safety-critical application domains, such as automotive, avionics, and space. The computational complexity of DNN models necessitates specialized hardware accelerators. However, as nanoelectronics used in hardware accelerators are prone to faults caused by radiation and aging, concerns about their reliability become of utmost importance. Moreover, with continuous technology scaling and increased silicon density, these accelerators become even more susceptible to faults. With the high integration density of modern nanoelectronic devices, the system error rate is increasing rapidly, as it is proportional to the number of transistors.

The objective of the thesis is to develop new methods for reliability evaluation and fault tolerance enhancement of modern hardware platforms, specifically DNN accelerators. The fundamental challenges are the complexity of modern hardware architectures and DNN models.

Heterogeneous systems-on-chip (SoCs), featuring scalar and vector processors for software programmability and FPGA logic for hardware programmability, are a common platform for implementing hardware accelerators for intelligent autonomous systems. However, efficient deployment of heterogeneous computing platforms requires a sufficient understanding of their sensitivity to possible faults. To address this problem, this thesis presents a survey of fault-tolerance vulnerabilities in modern heterogeneous systems for critical applications. The survey analyzes the most common error types caused by faults in SoC components and identifies the most critical and vulnerable ones. It examines vendor-provided and user-space reliability solutions, highlights existing fault-tolerance gaps, and outlines a vision for future solutions.

Reliability evaluation is a prerequisite for enhancing reliability since it shows how faults affect the system and identifies the most vulnerable components. Reliability evaluation of DNNs can be performed at the model level, without considering the hardware architecture on which it will be executed, or at the hardware level. Typically, model-level assessment methods are fast; however, because they do not account for hardware architecture, they tend to be less accurate. Hardware-level methods evaluate fault effects with higher accuracy but suffer from longer simulation times. To address the need for a scalable reliability evaluation of DNN hardware accelerators, this thesis presents a reliability assessment method for systolic arrays based on fault propagation analysis (SAFPA). It provides several orders of magnitude speedup over microarchitecture-level fault injection. The method supports assessment for two modern DNN architectures for computer vision tasks, namely Convolutional Neural Networks (CNNs) and Vision Transformers (ViT). Moreover, it enables a design space exploration of reliability-performance trade-offs in CNNs deployed on systolic arrays with reconfigurable size.

An efficient reliability solution for modern accelerator platforms should be tailored to the mission profile and operating conditions. It should account for the complexity of the hardware platforms used and the application. To address the need for a cost-efficient reliability solution, this thesis presents a fault-tolerant reconfigurable systolic array for DNN inference, FORTALESA. FORTALESA supports three execution modes for efficient reliability–performance tradeoff. The proposed architecture is used to enhance the reliability of DNN inference on a systolic array through heterogeneous mapping of network layers to

different execution modes. Finally, to enable reliability on the individual module level, the thesis presents an adaptive fault-tolerant approximate multiplier tailored for DNN accelerators, AdAM. AdAM relies on an unconventional use of the leading-one position in the inputs for fault mitigation by optimizing unused resources, thereby enabling negative-overhead fault tolerance compared to exact multipliers. It achieves a reliability level close to the multipliers protected with Triple Modular Redundancy (TMR), while requiring less area and power than the unprotected exact multiplier.

Kokkuvõte
Süvanärvivõrkude riistvarakiirendite kihtidevaheline töökindluse hindamine ja veakindlus

Viimaseid aastaid on iseloomustanud masinõppe (ML) algoritmide, eelkõige süvanärvivõrkude (DNN), kasutuselevõtu hüppeline kasv ning nende rakendamine missiooni- ja ohutuskriitilistes valdkondades, nagu autotööstus, lennundus ja kosmosetehnika. DNN-mudelite arvutuslik keerukus tingib vajaduse spetsiaalsete riistvarakiirendite järele. Kuna riistvarakiirendites kasutatav nanoelektroonika on aga vastuvõtlik kiirgusest ja vananemisest põhjustatud riketele, muutub nende töökindlus äärmiselt oluliseks. Pidev tehnoloogia mõõtkava vähenemine ja räni tiheduse kasv muudavad need kiirendid riketele veelgi vastuvõtlikumaks. Tänapäeva nanoelektroonikaseadmete suure integratsioonitiheduse tõttu kasvab süsteemi veamäär kiiresti, kuna see on võrdeline transistoride arvuga.

Doktoritöö eesmärk on välja töötada uued meetodid tänapäevaste riistvaraplatvormide, eelkõige DNN-kiirendite töökindluse hindamiseks ja veakindluse suurendamiseks. Põhilised väljakutsed on tänapäevaste riistvara-arhitektuuride ja DNN-mudelite keerukus.

Heterogeensed kiipsüsteemid (SoC), mis sisaldavad tarkvaralise programmeeritavuse jaoks skalaar- ja vektorprotsessoreid ning riistvaralise programmeeritavuse jaoks FPGA-loogikat, on levinud platvorm intelligentsete autonoomsete süsteemide riistvarakiirendite realiseerimiseks. Heterogeensete arvutusplatvormide tõhus kasutuselevõtt eeldab aga piisavat arusaama nende tundlikkusest võimalike rikete suhtes. Selle probleemi käsitlemiseks esitatakse doktoritöös ülevaade veakindluse haavatavustest tänapäevastes heterogeensetes süsteemides, mis on mõeldud kriitiliste rakenduste jaoks. Ülevaates analüüsitakse SoC-komponentide riketest põhjustatud levinumaid veatüüpe ning tuvastatakse kõige kriitilisemad ja haavatavamad komponendid. Käsitletakse nii tootjapoolseid kui ka kasutajatasandi töökindluslahendusi, tuuakse esile olemasolevad veakindluse puudujäägid ning visandatakse tulevaste lahenduste suund.

Töökindluse hindamine on töökindluse suurendamise eeldus, kuna see näitab, kuidas rikked süsteemi mõjutavad, ja tuvastab kõige haavatavamad komponendid. DNN-ide töökindlust saab hinnata kas mudeli tasandil, arvestamata riistvara-arhitektuuri, millel mudel käivitatakse, või riistvara tasandil. Mudelitasandi hindamismeetodid on tavaliselt kiired, kuid kuna need ei arvesta riistvara-arhitektuuriga, on need üldjuhul vähem täpsed. Riistvaratasandi meetodid hindavad rikete mõju suurema täpsusega, kuid eeldavad pikemat simulatsiooniaega. Selleks et rahuldada vajadus DNN-riistvarakiirendite skaleeritava töökindluse hindamise järele, esitatakse doktoritöös rikete leviku analüüsil põhinev süstoolsete massiivide töökindluse hindamise meetod (SAFPA). See on mitu suurusjärku kiirem kui mikroarhitektuuri tasandi rikete sisestamine (fault injection). Meetod toetab hindamist kahe tänapäevase arvutinägemise ülesannete DNN-arhitektuuri puhul: konvolutsiooniliste närvivõrkude (CNN) ja Vision Transformer'ite (ViT) korral. Lisaks võimaldab see uurida töökindluse ja jõudluse vahelisi kompromisse projekteerimisruumis CNN-ide puhul, mis on realiseeritud muudetava suurusega süstoolsetel massiividel.

Tänapäevaste kiirendiplatvormide tõhus töökindluslahendus peaks olema kohandatud missiooniprofiili ja töötingimustega. See peaks arvestama kasutatavate riistvaraplatvormide ja rakenduse keerukusega. Selleks et rahuldada vajadus kulutõhusa töökindluslahenduse järele, esitatakse doktoritöös DNN-i järeldamiseks mõeldud veakindel ümberkonfigureeritav süstoolne massiiv FORTALESA. FORTALESA toetab kolme täitmisrežiimi, mis võimaldavad tõhusat kompromissi töökindluse ja jõudluse vahel. Pakutud arhitektuuri kasutatakse DNN-i järeldamise töökindluse suurendamiseks süstoolsel massiivil, kaardistades võrgu kihid heterogeenselt erinevatele täitmisrežiimidele. Lõpetuseks, töökindluse ta-

gamiseks üksikmooduli tasandil, esitatakse doktoritöös DNN-kiirendite jaoks kohandatud kohanduv veakindel ligikaudne korrutaja AdAM. AdAM tugineb sisendite juhtiva ühebiti (leading-one) positsiooni ebatraditsioonilisele kasutamisele rikete parandamiseks, optimeerides kasutamata ressursse ja võimaldades seeläbi täpsete korrutajatega võrreldes negatiivse lisakuluga veakindlust. See saavutab töökindluse taseme, mis on lähedane kolmekordse moodulliiasusega (TMR) kaitstud korrutajatele, vajades samas vähem pindala ja võimsust kui kaitsmata täpne korrutaja.

Appendix 1

I

N. Cherezova, K. Shibin, M. Jenihhin, and A. Jutman. Understanding fault-tolerance vulnerabilities in advanced SoC FPGAs for critical applications. *Microelectronics Reliability*, 146:115010, 2023

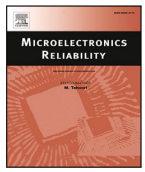
Microelectronics Reliability 146 (2023) 115010



Contents lists available at ScienceDirect

Microelectronics Reliability

journal homepage: www.elsevier.com/locate/microrel



Review paper



Understanding fault-tolerance vulnerabilities in advanced SoC FPGAs for critical applications

Natalia Cherezova [a,*], Konstantin Shibin [a,b], Maksim Jenihhin [a], Artur Jutman [b]

[a] Department of Computer Systems, Tallinn University of Technology, Tallinn, Estonia
[b] Testonica Lab, Tallinn, Estonia

ARTICLE INFO

Keywords:
Soft errors
Fault effects
Fault-tolerance
Cross-layer reliability
SoC FPGA

ABSTRACT

The emergence of heterogeneous FPGA-based SoCs and their growing complexity fueled by the introduction of various accelerators bring the reliability aspect of these systems to the front. The concern is particularly important for critical applications, such as safety-critical autonomous vehicles, real-time systems, space missions, health, and security cyber–physical systems. This paper presents an analysis of the most common types of errors caused by faults in different components and identifies the most critical and vulnerable components based on the literature survey. The study looks into vendor-provided and user-space reliability solutions. The paper highlights the existing fault-tolerance gaps in the modern SoC FPGAs and outlines a vision for future solutions.

1. Introduction

Modern heterogeneous SoC FPGAs, that provide software programmability through scalar and vector processors and hardware programmability through FPGA logic, are getting more complex and advanced, integrating more components on a single chip. However, their dense structure is also increasingly susceptible to various hardware faults. Their reliability assessment has certain challenges: simulation and testing are difficult due to the complexity of the design, and faults in one core can induce errors in other cores [1].

Hardware faults can be caused by various reasons: radiation (soft errors), aging, manufacturing process variations, manufacturing defects, and early-life failures. Nonetheless, soft errors are considered dominant for today's digital systems, hence, the analysis in this paper is focused on them.

The most common type of soft errors is Single Event Upset (SEU), a change of the logic state of the sequential element. Less common but more critical is Single Event Functional Interrupt (SEFI) — a soft error that causes a functional interrupt in circuits, interfaces, or entire chips. Another type of soft error is Single Event Transient (SET), a current or voltage spike in a signal. SET is transitory in nature, however, if captured by a sequential element, it can become an SEU.

The heterogeneous nature of SoC FPGAs is the reason that faults in different components manifest differently and require tailored fault-tolerance solutions, which makes the reliability assurance of the chip a challenging task.

The contributions of this paper are:

- a taxonomy of an advanced SoC FPGA structure;
- an analysis of fault effects caused by faults in the different components of the structure;
- a review of vendor-provided fault-tolerance mechanisms and analysis of fault-tolerance vulnerabilities and gaps;
- a review of complementary user-space fault-tolerance solutions.

The rest of the article is organized in the following way. The typical structure of the modern SoC FPGA and a discussion on the most common types of errors caused by faults in different components are presented in Section 2. The most critical and vulnerable components are identified there as well. Reliability solutions provided by vendors are given in Section 3. Section 4 highlights the fault tolerance gaps. User-space solutions are described in Section 5. Section 6 presents discussions and conclusions, and Section 7 summarizes the paper.

2. Faults and fault effects in SoC FPGAs

This section presents a typical structure of modern FPGA-based SoC along with faults and fault effects characterizing each component.

2.1. Typical structure of modern FPGA-based SoC

The structure is based on the latest series from AMD-Xilinx, Intel, and Microchip: Versal ACAP (Adaptive Compute Acceleration Platform) [2] and Zynq UltraScale+ [3] (AMD-Xilinx); Agilex [4] and

* Corresponding author.
E-mail address: natalia.cherezova@taltech.ee (N. Cherezova).

https://doi.org/10.1016/j.microrel.2023.115010
Received 28 June 2022; Received in revised form 11 April 2023; Accepted 8 May 2023
Available online 18 May 2023
0026-2714/© 2023 The Author(s). Published by Elsevier Ltd. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

Stratix 10 [5] SoC families (Intel), and PolarFire SoC [6] (Microchip). A modern FPGA-based SoC includes multi-core processors (ARM Cortex-A and Cortex-R cores in AMD-Xilinx and Intel devices, and RISC-V cores in Microchip SoC), FPGA logic, hard peripheral controllers, embedded on-chip memories, and system infrastructure represented by non-functional utility resources, such as chip monitors, ECC controllers, test facilities, etc. The most notable among them is a separate processor that acts as a device manager. Device manager, called Platform Management Unit (PMU) in AMD-Xilinx devices, Secure Device Manager (SDM) in Intel, and monitor core in Microchip, sets the root-of-trust, configures the device after the power-up, and monitors its state during the lifetime. Some platforms have accelerators, like embedded GPU for video rendering or custom engines for AI and ML tasks processing (Zynq UltraScale+ and Versal), and specialized modules, like radio frequency (RF) data converters and video processing units (Zynq UltraScale+). Versal ACAP represents the new generation of programmable SoCs from AMD-Xilinx and has the most complex structure. The processing system, FPGA logic, AI Engines (custom blocks for fast data processing), and peripheral controllers are connected through Programmable Network-on-Chip (PNoC). One can assume that the next FPGA-based SoCs, generally, follow the Versal model.

The typical structure described is presented in Fig. 1. The presented structure can be considered as a super-set. More complex system-on-chips include more elements from the set, and less complex ones include fewer, but they still stay within the given super-set.

2.2. Faults and fault effects in different components

Below is the analysis based on the literature survey of the most common types of errors caused by faults in different components and the most critical and vulnerable elements in each module. Table 1 summarizes fault effects at the module and system levels. System failures are divided into three classes: system crash (SC), application crash (AC), and application silent data corruption (ASDC). The most common system-level failure is ASDC. AC can be caused by control flow errors, and SC might be caused by errors related to the SoC configuration or control circuitry.

Processing system (PS). Structurally faults for a processor were organized into three large groups: faults in registers (including general-purpose and special-purpose ones), caches, and data processing units (ALU/FPU) [7–12]. It can be seen that faults in different components affect processors in two ways: silent data corruption (SDC) and control flow errors that can lead to functional interrupts or crashes. Functional interrupts can be caused by errors in control or special purpose registers, errors in addressing, and errors in instruction (wrong instruction, wrong operand, or wrong immediate) [8]. While functional interrupts require a reset and result in the system being unavailable during the restart, SDC does not disable the system but might be of great concern for a user, as they corrupt results without giving any indications that something went wrong. The most critical elements are registers (particularly special-purpose ones) and caches [7–13] since faults in those components directly affect the data being processed and can introduce control flow errors. Additionally, studies show that processor cores integrated into SoC platforms have higher rates of crashes and functional interrupts compared to standalone processors [14].

FPGA. Structurally, faults in FPGA can be organized into three groups as well: faults in configuration memory, user memory, and control-related logic [15]. Unlike processors, the functionality of FPGA is not hardened and is defined by the values loaded into configuration memory (called CRAM). Therefore, the main fault effect for them is the change in circuit functionality, which might result in silent data corruption as well as application or even system crash.

Configuration memory consists of routing elements and logic resources, such as LUTs (Look Up Tables) and control bits.

Routing resources can be local (inside logic blocks) or global (between logic blocks). Local routing is represented by multiplexers (MUX) controlling inputs and outputs of the logic block and LUTs. Logic blocks are connected through switch boxes and connection boxes by the means of Programmable Interconnect Points (PIPs). PIP may be represented by a simple pass transistor controlled by one configuration bit or a complex structure created using several multiplexers controlled by a group of configuration bits.

For PIP, the following fault models are defined: Open, Conflict, Input Antenna, Output Antenna, and Bridge [16]. Open fault happens when a previously used connection is disabled. Conflict is characterized by a new connection being added between a used input node and a used output node. Input Antenna is a new connection between an unused input node and a used output node. Output Antenna is a new connection between an unused input node and a used output node. Bridge happens when a previously used connection is disabled, and an unrelated input is connected to the output from the disabled connection.

Along with the change in circuit functionality, SEU in routing configuration bits that create an undesired bridge between a signal and an unused wire adds an extra parasitic capacitance (and thus an extra delay) to the signal path. The impact is cumulative and can lead to delays that are significant enough to cause delay faults [17].

Routing faults are the most common and the most critical for FPGAs since the routing structure occupies a large portion of the FPGA area and configuration bits [18]. Moreover, the most failure-sensitive routing elements are local MUXes [19,20].

Faults in logic include LUT values change and control bits changes. LUTs are used to hold the data (when used as distributed RAM or shift register) and implement data processing or Finite State Machine (FSM) to control the workflow. Therefore, faults in LUT values can cause data corruption as well as control flow errors. However, whether it produces an error or not depends on the inputs to the circuit and which part of the logic function is impacted.

On the contrary, changes in control bits generally cause errors for all, or almost all, possible circuit inputs. Control bits define LUT mode (whether it is used as a LUT, distributed RAM, or a shift register), configure storage elements as flip-flops or latches, and define Input/Output Block (IOB) parameters, such as electrical standard, mode (input, output, bi-directional), termination (pull-up or pull-down resistors), signal drive strength, optional storage element, input delay, and a slew rate. Faults in IOB parameters can create unintended outputs [21] and additional delays [22–24].

User memory includes embedded RAM and flip-flops in the logic blocks. It stores the data produced by the application. Faults in user memory are straightforward and cause silent data corruption. While faults in embedded RAM data remain until a new value is stored there or the fault is corrected, original flip-flop values might be restored in the next clock cycle. Faults in user memory are considered less critical than faults in the configuration memory. The former ones do not affect the circuit functionality and can be fixed without device reconfiguration.

Faults in control logic include faults in reset circuitry and configuration ports, such as JTAG TAP (Test Access Port) controller, ICAP (Internal Configuration Access Port), PCAP (Processor Configuration Access Port), and SelectMAP interface. SEU in reset circuitry can clear out the device functionality or all of the flip-flops. Faults in the JTAG TAP controller can damage the FPGA or other system devices as well as cause a loss of control [25]. Faults within the programmable SelectMAP interface configuration pins result in a malfunctioning interface: the corrupted interface can return bad values during a read operation and corrupt the configuration during a write operation [15]. Faults in control logic have serious consequences and require device reconfiguration, though, in practice, their rates are very low [15].

Moreover, in SoC, the FPGA part is proved to be more prone to soft errors [8,26,27], while processors and peripherals have more vanished errors.

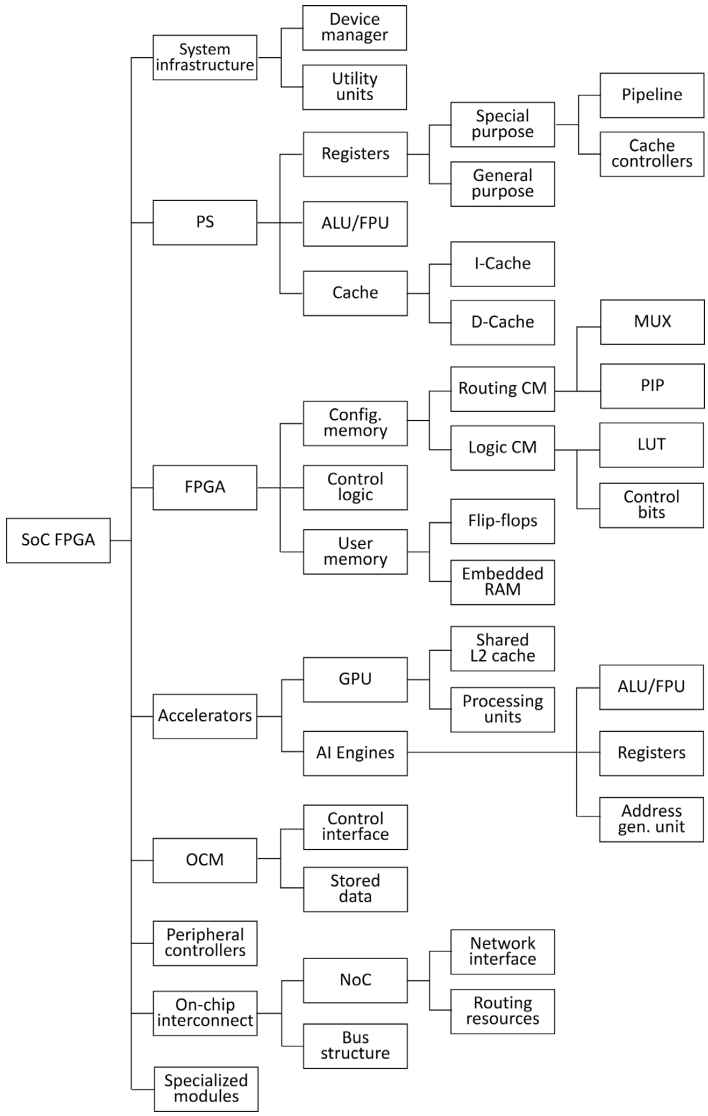


Fig. 1. Typical structure of modern FPGA-based SoC.

Accelerators. Two types of accelerators are considered in this work: embedded GPU and custom AI Engines suggested by AMD-Xilinx. Versal AI Engines [28] consist of two processors, for scalar and vector arithmetic, and thus their structural faults are similar to the processor faults. Faults in instructions can cause data and control flow errors, and faults in registers and in data processing units can cause silent data corruption. AI Engines do not have cache memory, though they do have shared memory for storing the data.

For GPU, faults in global buffers (shared L2 cache) can be considered the most critical, since they may spread in multiple locations and can cause many erroneous outputs [29–31].

Faults in accelerators are not expected to cause crashes but rather produce wrong outputs. While data errors in machine learning applications might have serious consequences, erroneous pixels in a video frame just degrade the user experience.

Peripherals and memory controllers. Experiments show that faults in address and control registers cause many data errors and interrupts [8, 32]. However, there are only a few works that explore faults in so-called "uncore" components, such as peripherals, on-chip memories, etc. [33–35]. However, loss of configuration in peripheral controllers might result in a system crash and require device reconfiguration.

On-chip interconnect. Different components in heterogeneous SoCs are usually connected through a bus-based communication architecture. However, Versal ACAP represents an interesting case that additionally utilizes network-on-chip (NoC). Versal NoC architecture is similar to a typical NoC structure: network interface units connect processing elements to the NoC, and routers perform the transportation of the data through the NoC. Network interface units are represented by NoC Master Units (NMU) and NoC Slave Units (NSU) that denote ingress and egress blocks accordingly. Routers are represented by NoC

Table 1
Faults and fault effects in SoC FPGA.

Fault location		Error within module	System level failures		
			SC	AC	ASDC
Processor					
Register file	General-purpose registers	Silent data corruption, control flow errors	×	•	•
	Special purpose registers	Control flow errors	×	•	∘
Cache	Instruction cache	Control flow errors	×	•	∘
	Data cache	Silent data corruption	×	∘	•
Data processing units (ALU/FPU)		Silent data corruption	×	∘	•
FPGA					
Configuration memory	Routing elements	Functional errors (change in circuit functionality)	•	•	•
	Logic resources				
User memory	Flip-flops	Silent data corruption	×	∘	•
	Embedded RAM				
Control logic	Reset	Configuration loss, functional interrupts	•	•	•
	JTAG TAP controller				
	SelectMAP interface				
Accelerators					
GPU	L2 shared cache	Silent data corruption, control flow errors	×	×	∘
	Processor units				
AI Engine	Register file	Silent data corruption, control flow errors	×	×	•
	Data processing units				
	Address generation unit				
OCM and peripherals					
OCM	Control interface	Silent data corruption, control flow errors	∘	•	•
	Stored data		×	∘	•
Peripheral controllers		Control flow errors	•	∘	∘
On-chip interconnect					
NoC	Processing units	Silent data corruption, configuration loss	•	•	•
	Connections				
Bus structure		Silent data corruption, configuration loss	•	•	•

SC — system crash. AC — application crash. ASDC — application silent data corruption.
×— not expected. ∘ — possible. • — highly likely.

Packet Switch (NPS). It is a crossbar switch that determines the direction of the packet based on the special routing tables. Though Versal NoC is programmable, its configuration is statically defined during the design time, therefore, it includes an additional unit called NoC Peripheral Interconnect (NPI). NPI is an internal register programming interconnect of the NoC that controls and configures NMU, NSU, and NPS. The NPI programs NoC registers that define the aforementioned routing tables [36].

In the classic NoCs, faults in interconnects are considered dominant [37,38]. Additionally, since the configuration of Versal PNoC is programmable, faults in routing components and NPI registers can have serious consequences. They can lead to a loss of configuration and, therefore, to an application or a system crash.

For the AMBA AHB system bus, experiments in [39] show that SDC is the most common error, however, faults in the address bus have a high probability to cause an application or system crash.

3. Vendor-provided fault tolerance mechanisms

The heterogeneous nature of modern SoC FPGAs, combining different components, requires a complex fault tolerance solution that covers the whole chip.

The main fault detection and fault tolerance mechanisms employed by vendors are the following:

- Device manager
- Modular redundancy (usually TMR)
- ECC, CRC, parity checking
- Data scrubbing
- Dual lock-step
- Memory protection units
- Built-in tests
- Watchdog timers

A comparison between fault detection and fault tolerance methods on different platforms is presented in Table 2.

Device manager. As already mentioned, all considered platforms have a dedicated processor or two acting as a device manager: Platform Management Unit (PMU) on AMD-Xilinx platforms, Secure Device Manager (SDM) on Intel platforms, and monitor core on Microchip PolarFire SoC. The device manager holds the root-of-trust, configures the device after the power-up, performs error management, and monitors its state during the lifetime to support the functional safety of the platform.

Triple-modular redundancy (TMR). On AMD-Xilinx and Intel platforms TMR is used to protect the device manager and critical state registers. Additionally, Intel and Microchip offer functionality to generate a triplicated design for FPGA logic. AMD-Xilinx offers similar functionality for Virtex family devices only. For modern platforms, it offers TMR MicroBlaze IP (soft processor).

Error correction code (ECC). ECC is the standard protection mechanism for memories. AMD-Xilinx, Intel, and Microchip have ECC (single-error correction, double error detection) on all processor memories, on-chip memories, FPGA embedded memories, and DDR4 controllers. ECC mechanism supports error injection functionality. On Intel platforms, USB, SD/MMC, EMAC, DMA, and NAND flash controllers have

Table 2
Vendor-provided fault tolerance mechanisms in different platforms.

Method	AMD-Xilinx		Intel		Microchip	
	Realization	What is protected	Realization	What is protected	Realization	What is protected
Device manager	Platform Management Unit (PMU)	System	Secure Device Manager (SDM)	System	RISC-V Monitor core	System
Modular redundancy	Hard TMR	Device manager + critical state registers	Hard TMR	Device manager + critical on-chip state machines	—	
	—		Wizard for TRM generation on FPGA logic	Design on FPGA logic	Wizard for TRM generation on FPGA logic	Design on FPGA logic
ECC	SECDED	Caches, OCM, FPGA embedded memories, DDR4 controller	SECDED/multi-bit correction	Caches, OCM, FPGA embedded memories, DDR4 controller, NAND flash controller	SECDED	Caches, FPGA embedded RAM, DDR3/4 controller
Lock-step	Dual lock-step	Cortex-R5F core	—		—	
Data scrubbing	XilSEM IP	FPGA CRAM	Intel EDC feature	FPGA CRAM	—	
Memory protection units	ARM TrustZone, XMPU, XPPU	Memories, peripherals	ARM TrustZone, SMMU	Memories	MMU, PMP	Memories
Built-in tests	LBIST	XMPU, lock-step, and ECC checkers	*(No info found)*		MBIST	L2 cache
	MBIST	Cortex-R5F core memories				
	STL	Memories, PMU, XMPU, XPPU, clocks, voltages, temperature				
Watchdog timers	Hardware watchdogs	Processing system	Hardware watchdogs	Processing system	Hardware watchdogs	Processing system

integrated ECC-protected memories. While most ECCs provide a single-error correction, double-error detection, for M20K blocks (internal 20 Kbit memories), Intel suggests single-error, double-adjacent-error, and triple-adjacent-error correction in a 32-bit word. However, it is reported that with the ECC feature enabled, M20K runs slower [40]. In order to decrease the possibility of multi-bit errors, all vendors use a memory interleaving technique.

Dual lock-step. AMD-Xilinx platforms include ARM Cortex-R5F dual-core processor that supports the lock-step functionality. In the lock-step mode, two cores perform the same task in parallel.

Data scrubbing. AMD-Xilinx and Intel, both suggest solutions for soft-errors correction in the FPGA configuration memory (CRAM) based on the background scan. Xilinx Soft Error Mitigation (XilSEM) is available as soft IP in the UltraScale+ series [41], however, in the Versal series, it is a part of PMU [42]. Triplicated PMU provides more protection for the module itself. XilSEM uses ECC and CRC for single-error correction and double-error detection. On Versal, XilSEM can additionally detect errors in NPI (NoC Peripheral Interconnect) registers that connect NoC with some modules using SHA techniques. XilSEM supports error injection functionality.

Intel Error Detection Circuit (EDC) with integrated ECC corrects single- and double-bit errors and detects up to 8 bits that fit in a rectangular box (8×1, 4×2, 1×8, 2×4). EDC supports error injection and priority scrubbing. Some portions of the design can be marked as high-priority sectors, those sectors are scanned more frequently. Additionally, there is a sensitivity processing and hierarchy tagging functionality that defines the criticality of the CRAM bit and suggests corrective actions based on the classification [43,44].

Since PolarFire SoC incorporates flash-based FPGA, which is more robust towards soft errors, there is no scrubbing solution suggested.

Memory protection units (MPU). While memory protection units are not explicitly designed for soft error protection, they can help with them as well. One of the main purposes of MPU is to restrict memory access of a process to some defined region. If there is an error in the address due to the SEU so that it no longer belongs to the allowed region, access is blocked, which might prevent SDC or control flow error.

AMD-Xilinx platforms have a Memory Protection Unit (XMPU) and Peripheral Protection Unit (XPPU) to protect on-chip memory blocks and peripheral programming registers [42]. Intel platforms have System Memory Management Unit (SMMU), and one of its functions is memory access permissions. PolarFire has Memory Protection Unit as well. Additionally, since AMD-Xilinx and Intel SoC include ARM processor cores, they are protected by ARM TrustZone.

Built-in tests. AMD-Xilinx provides LBIST for check logic such as XMPU, lock-step, and ECC checkers, and MBIST for Cortex-R5F core memories. Additionally, memories, lock-step functionality, device manager, XMPU, XPPU, clocks, voltages, and temperature can be checked by the dedicated software test library (STL) [42]. Microchip provides MBIST for L2 cache [45].

Watchdog timers. Used for detection of the hanged or stuck in an infinite loop process, watchdog timers are provided by all three considered vendors.

4. Analysis and conclusions about the gaps

4.1. Vendor-provided fault tolerance methods analysis

Fault tolerance methods described in the previous section have certain overheads, such as area, power consumption, or execution time. They also have functional limitations on what kind of errors they can detect or correct and how fast they can do it. Therefore, for this study, it is important to see how vendor-provided methods protect the system from soft errors and what parts of SoC FPGAs are left unprotected.

Device manager. Implementation of the device manager requires one or two additional processors, which results in power consumption overhead. Moreover, the device manager does not protect the system from soft errors.

Triple-modular redundancy. TMR is based on utilizing three copies of the same module working in parallel, which introduces more than 200% overhead in the area and power consumption. This might be critical for energy-restricted applications. Therefore, TMR is only used to protect the device manager which acts as a safety island.

ECC. When enabled, ECC requires additional bits to store the codes, which results in 13 and 22% overhead for 32-bit and 64-bit data accordingly. In most cases, it can only correct single-bit errors and detect double-bit errors. Multi-bit correction and detection versions are possible but might slow down the memory. Since ECC is a memory protection technique, it mostly prevents SDC errors, however, in the case of Instruction Cache, it might prevent control flow errors as well.

Data scrubbing. The scrubbing solution that scans FPGA configuration memory in the background brings power consumption overhead. It has a latency of several milliseconds, during which the device might operate incorrectly. Studies show that scrubbing does not protect the device against immediate failures since it cannot fix errors propagated from faults in configuration memory [46]. Its main functionality is to prevent the accumulation of faults. Besides, it is not intended for use in a high-irradiation environment, such as space. An experiment in [47] shows the significant decrease in neural network accuracy deployed on the AMD-Xilinx UltraScale+ platform under beam radiation testing even with XilSEM enabled.

Dual lock-step. When two cores are running in the dual lock-step mode, they perform the same tasks with one clock cycle shift, which means the system can perform fewer tasks than in the split mode. Furthermore, since lock-step is a fault detection technique, it requires additional time to rerun the task in case of an error. Additionally, lock-step functionality is not supported by every processor.

MPU. As it was mentioned before, the memory protection unit is not designed to handle faults, though it can detect faults in some cases and prevent them from propagating further.

Watchdog timers. As a symptom-based fault detection method, the watchdog timer can neither isolate the fault nor correct it. The watchdog timer can either create an interrupt or restart the processor core.

4.2. Gaps in vendor-provided protection

Based on the information from previous sections, the fault tolerance taxonomy for modern SoC FPGA was created (Fig. 2). Critical and vulnerable components pointed out by research papers are marked by a red square, and components that have some vendor-provided fault detection or fault tolerance solutions are marked using corresponding colors.

As can be seen from the figure, the most protected module in the modern SoC FPGA is the processing system that has watchdog timers, lock-step capabilities, ECC for caches, and memory protection units. However, it should be noted that not every processor supports lock-step functionality, watchdog timers cannot isolate and recover the fault, and memory protection units are not designed to handle faults but can only detect errors in some cases. ECC prevents errors originating from caches, however, the lack of protection in the other components of the processor, including registers that are considered critical as well, means that faults stemming from the other components are not covered. The lock-step mechanism protects the whole processor, however, in the considered SoC families, lock-step functionality is only supported in the ARM R-cores, leaving A-cores vulnerable.

The next protected module is FPGA logic which has scrubbing solutions for configuration memory and ECC for embedded memories. User memory flip-flops, though, are not covered by any means. The least protected modules are peripherals, interconnects, accelerators, and specialized modules, the ones that are less explored in the research articles as well.

The following gaps can be formulated:

1. Most of the vendor-provided fault-tolerance methods protect the processing system and memory components. ECC effectively covers single-bit faults in memories. Since memories usually occupy the major part of the chip, a high percentage of fault tolerance can be achieved by covering only memories using a considerably inexpensive and easy-to-implement mechanism like ECC. However, that leaves computational and control units less protected. They might occupy less area than memories, but faults in those components might lead to a crash.
2. The least protected modules are uncore components that are less considered in the literature as well. However, due to the complexity of the SoC, it is not enough to protect only the processing system. In fact, all components may have an impact on the reliability of the whole system [48].
3. The vendor-provided fault-tolerance methods for cost-efficient COTS devices are not designed for harsh environments or critical applications.
4. Different fault effects and used fault-tolerance methods require a systematic and hierarchical approach for efficient device protection. However, the current approach seems fragmented. The main reason to use the SoC FPGA platform is to use as many of its components as possible, not only the processor or FPGA. In such cases, a communication system, for example, plays an important role. For the application to run correctly, it is important to protect the whole system.

5. Mapping known user-space methods

Clearly, all major FPGA vendors are seeking the most cost-efficient methods to offer basic protection against in-field failures. Advanced or full protection becomes too expensive as the main volume of FPGA devices goes into commercial applications such as industrial, telecom, or video processing, where fault tolerance mechanisms would only introduce additional cost and overhead and, hence, reduce the vendor's competitive advantage. Therefore, gaps in protection are inevitable and have to be addressed carefully by the users targeting commercial off-the-shelf (COTS) devices for mission or safety-critical applications such as aerospace.

The pursuit of improving fault tolerance is not limited to the features that the manufacturer embeds into their devices. There is ongoing research on methods to analyze and improve the fault tolerance [49] using the controls available to the user of an SoC, such as programming of the processor cores or the FPGA logic in a way to make the operation more reliable. Several of those methods are typically based on exploiting some sort of redundancy that is already existing or can be created by the user.

Since the modern SoC FPGAs inevitably contain heterogeneous modules, which are also commonly found separately, it makes sense to look at the methods which are already employed for such modules.

General purpose processing units (PS). The availability of multi-core CPUs in modern SoCs allows for exploiting the inherent redundancy for improving transient fault tolerance in software. Where available in hardware, dual-core or triple-core lock-step redundant core configurations can provide significant protection against transient errors [9,50]. A hybrid lock-step approach with hard CPU cores in combination with redundant soft cores is also possible and demonstrated in [51]. Fault tolerance methods described in [52] allow using temporal redundancy and target non-protected COTS CPUs. They enable the protection by trading higher performance and a larger quantity of these CPUs (compared to fault-tolerant alternatives, if any are present) for fault tolerance by employing redundant code execution with some variable but predictable performance penalty. These methods use structural or temporal *active* redundancy where the reaction to fault detection as well as recovery from an error can be flexible and can depend on the executed application. Another software-centric approach proposed

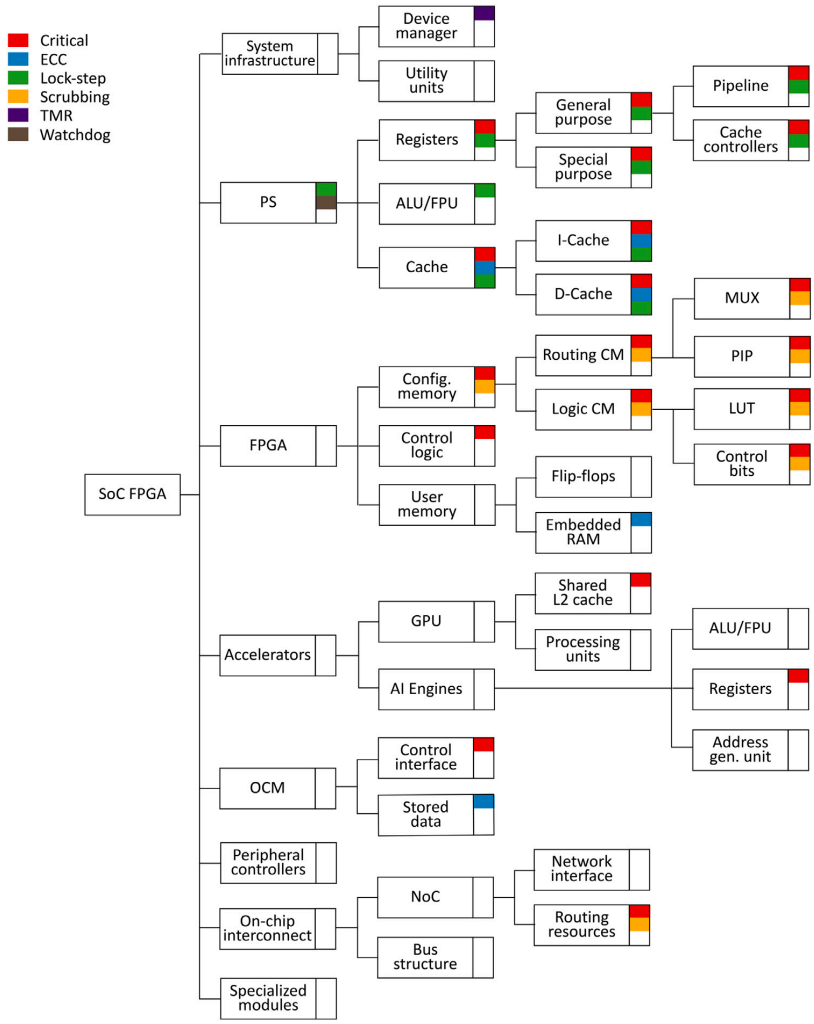


Fig. 2. Fault tolerance taxonomy for SoC FPGA based on vendor-provided methods.

in [53] uses process-level redundancy (PLR), where a set of redundant copies of an application process is created and the results are systematically compared to guarantee correct execution. Redundant multi-threaded processes are proposed for fault tolerance in user-space programs running on multi-core CPUs [54] where the results of two threads are compared for error detection while another thread is used as a watchdog. There are other similar methods from other groups which aim at increasing fault tolerance in software [55,56]. Several methods aim at soft-core CPUs. One of them uses a combination of redundancy in multi-core processors and extra hardware measures for managing redundancy configurations in asymmetric multiprocessing mode (AMP) [57]. In [58], authors implement fault detection in software through thread-replication and coarse-grain lock-step for weakly coupled processing cores.

FPGA fabric is configurable and, from the functional point of view, can work like any other digital hardware and, therefore, can implement hardware methods like TMR, ECC, and others in so-called soft cores. The underlying physical structure of FPGA, however, is very different from usual digital hardware because the function depends on the configuration memory as described earlier. FPGA works properly as long as the configuration memory is intact. In the case of SRAM-based FPGA, the configuration memory is susceptible to SEUs, and these may result in a change of the function. Therefore, usual hardware fault-tolerance methods need to be applied carefully. For example, TMR can be combined with scrubbing [59] in order to detect and correct the errors in the configuration memory, while TMR protects from soft errors in user logic/memory. SEUs in configuration memory can also affect routing, therefore partitioning and placement of the logic in an FPGA can be exploited to increase fault tolerance [60,61]. For example, it can change the probability of errors affecting TMR [62]. Other methods also aim at repairing broken configuration by partial dynamic reconfiguration [63].

Accelerators. Various hardware accelerators are used to increase the performance and/or efficiency of the processing on SoCs, and they, too, can benefit from added fault tolerance. For example, neural network accelerators are susceptible to hardware-induced faults, but there are methods to mitigate the effects of faults [64]. For fixed-hardware neural network accelerators, there are software methods like result

range restriction [65] and algorithm-based fault tolerance [66]. Soft-core accelerators can be augmented with an extra layer for improved fault tolerance [67].

6. Discussion and conclusions

This paper reviewed the state of the art in fault-tolerance mechanisms offered by SoC FPGA vendors with a focus on the most popular COTS devices of the latest SoC families. There has been substantial progress in such protection mechanisms during the last decade as opposed to simple technology-level hardening. As a result, there is a clear industrial trend to build mission- and safety-critical systems around COTS devices [68]. For instance, in the space segment, this trend is further escalated by the need for in-orbit high-performance computing [69].

Still, as the vast majority of SoC FPGAs are employed at non-critical commercial applications, there are gaps in the protection offered off the shelf by vendors. Advanced SoC FPGAs comprise a diversity of tightly integrated components that should all be protected to ensure the system's reliability and availability. However, as was shown, the most protected components are the processing system and memories, leaving the rest of the system vulnerable. Uncore components are not protected and are less considered in the literature. The current approach to fault tolerance is fragmented and not systematic. Finally, vendor-provided fault-tolerance solutions for COTS devices are not intended for harsh environments and critical applications.

Hence, it is necessary to implement complementary user-space fault-tolerance mechanisms tailoring them to target mission and application profiles, from simple custom TMR [70] to advanced fault handling schemes [71]. For better protection, user-space methods should interact with vendor-provided ones resulting in an integrated fault-tolerance solution. For example, the scrubbing procedure can be triggered by a TMR voter in the FPGA logic or lower-cost fault detection checkers or monitors in the FPGA logic. Also, in some scenarios, fault detection may be supported by employing higher-level techniques such as time-domain redundancy for unprotected processors, software-based checking, and external performance/process monitoring. Here, the user-space fault-tolerance mechanisms would both protect the operation and initiate the recovery procedure offered by the vendor, thus improving the overall protection efficiency. More advanced fault handling mechanisms involving cross-layer interaction and data exchange are also possible [72].

However, implementing fault-tolerance solutions on different layers without considering the whole picture might result in an overprotected system or even a degradation in the overall reliability. A prominent solution to cover the aforementioned gaps efficiently is a cross-layer reliability approach.

Cross-layer approach to reliability has been gaining popularity in recent years since it can help to develop a globally optimized fault management system. Several frameworks have been developed by the academic community for the reliability-aware design space exploration to find the best combination of fault-tolerance methods [73,74] and for the cross-layer reliability assessment of the system [75]. Those frameworks primarily target processors at the early design stage, considering HW/SW co-design process. However, as already mentioned, COTS devices are used more and more for critical applications. Thus, a framework targeting heterogeneous SoC FPGAs at the stage of embedded system development would be useful.

7. Summary

This paper presents an analysis of the most common types of errors caused by faults in different components and identifies the most critical and vulnerable components based on the literature survey. The study looks into vendor-provided and user-space reliability solutions. The paper highlights the existing fault-tolerance gaps in the modern SoC FPGAs and outlines a vision for future solutions.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

Acknowledgments

This work was supported in part by the European Union through European Social Fund in the frames of the "Information and Communication Technologies (ICT) programme" ("ITA-IoIT" topic), by the Estonian Research Council grant PUT PRG1467 "CRASHLES", and by ESA GSTP Contract No. 4000139825/22/NL/AS.

References

[1] C.M. Fuchs, K. Marinis, P. Chou, X. Wen, N.M. Murillo, G. Furano, S. Holst, A. Tavoularis, S.-K. Lu, A. Plaat, A fault-tolerant MPSoC for CubeSats, in: 2019 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems, DFT, IEEE, 2019, pp. 1–6, http://dx.doi.org/10.1109/DFT.2019.8875417.
[2] Versal: The First Adaptive Compute Acceleration Platform (ACAP), Xilinx, Inc., 2020.
[3] Zynq UltraScale+ MPSoC Data Sheet: Overview, Xilinx, Inc., 2021.
[4] Intel Agilex FPGAs and SoCs Device Overview, Intel, 2021.
[5] Intel Stratix 10 GX/SX Device Overview, Intel, 2021.
[6] PolarFire SoC Product Overview, Microchip Technology Inc., 2021.
[7] W. Yang, X. Du, C. He, S. Shi, L. Cai, N. Hui, G. Guo, C. Huang, Microbeam heavy-ion single-event effect on Xilinx 28-nm system on chip, IEEE Trans. Nucl. Sci. 65 (2018) 545–549, http://dx.doi.org/10.1109/TNS.2017.2776244.
[8] X. Du, C. He, S. Liu, Y. Zhang, Y. Li, W. Yang, Measurement of single event effects induced by alpha particles in the Xilinx Zynq-7010 system-on-chip, J. Nucl. Sci. Technol. 54 (2016) 1–6, http://dx.doi.org/10.1080/00223131.2016.1262294.
[9] X. Iturbe, B. Venu, E. Ozer, Soft error vulnerability assessment of the real-time safety-related ARM Cortex-R5 CPU, in: 2016 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems, DFT 2016, Institute of Electrical and Electronics Engineers Inc., 2016, pp. 91–96, http://dx.doi.org/10.1109/DFT.2016.7684076.
[10] L.A. Tambara, P. Rech, E. Chielle, J. Tonfat, F.L. Kastensmidt, Analyzing the impact of radiation-induced failures in programmable SoCs, IEEE Trans. Nucl. Sci. 63 (2016) 2217–2224, http://dx.doi.org/10.1109/TNS.2016.2522508.
[11] L.A. Tambara, F.L. Kastensmidt, P. Rech, F. Lins, N.H. Medina, N. Added, V.A. Aguiar, M.A. Silveira, Reliability-performance analysis of hardware and software co-designs in SRAM-Based APSoCs, IEEE Trans. Nucl. Sci. 65 (2018) 1935–1942, http://dx.doi.org/10.1109/TNS.2018.2844250.
[12] R. Travessini, P.R. Villa, F.L. Vargas, E.A. Bezerra, Processor core profiling for SEU effect analysis, in: 2018 IEEE 19th Latin-American Test Symposium, Vol. 2018-January, LATS 2018, Institute of Electrical and Electronics Engineers Inc., 2018, pp. 1–6, http://dx.doi.org/10.1109/LATW.2018.8347235.
[13] G. Papadimitriou, D. Gizopoulos, Demystifying the system vulnerability stack: Transient fault effects across the layers, in: Proceedings - International Symposium on Computer Architecture, Vol. 2021-June, Institute of Electrical and Electronics Engineers Inc., 2021, pp. 902–915, http://dx.doi.org/10.1109/ISCA52012.2021.00075.
[14] P. Bodmann, G. Papadimitriou, D. Gizopoulos, P. Rech, The impact of SoC integration and OS deployment on the reliability of ARM processors, in: Proceedings - 2021 IEEE International Symposium on Performance Analysis of Systems and Software, ISPASS 2021, Institute of Electrical and Electronics Engineers Inc., 2021, pp. 223–225, http://dx.doi.org/10.1109/ISPASS51385.2021.00040.
[15] H. Quinn, Radiation effects in reconfigurable FPGAs, Semicond. Sci. Technol. 32 (2017) http://dx.doi.org/10.1088/1361-6641/aa57f6.
[16] C. Bernardeschi, L. Cassano, A. Domenici, L. Sterpone, ASSESS: A simulator of soft errors in the configuration memory of SRAM-based FPGAs, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 33 (2014) 1342–1355, http://dx.doi.org/10.1109/TCAD.2014.2329419.
[17] C. Thibeault, S. Pichette, Y. Audet, Y. Savaria, H. Rufenacht, E. Gloutnay, Y. Blaquiere, F. Moupfouma, N. Batani, On extra combinational delays in SRAM FPGAs due to transient ionizing radiations, IEEE Trans. Nucl. Sci. 59 (2012) 2959–2965, http://dx.doi.org/10.1109/tns.2012.2222668.
[18] T. Nidhin, A. Bhattacharyya, R. Behera, T. Jayanthi, K. Velusamy, Understanding radiation effects in SRAM-based field programmable gate arrays for implementing instrumentation and control systems of nuclear power plants, Nucl. Eng. Technol. 49 (2017) 1589–1599, http://dx.doi.org/10.1016/j.net.2017.09.002.

[19] N. Jing, J.-Y. Lee, Z. Feng, W. He, Z. Mao, L. He, SEU fault evaluation and characteristics for SRAM-based FPGA architectures and synthesis algorithms, Vol. 18, Association for Computing Machinery (ACM), 2013, pp. 1–18, http://dx.doi.org/10.1145/2390191.2390204.

[20] P.S. Graham, M.P. Caffrey, J.A. Zimmerman, P. Sundararajan, E.J. Johnson, C.D. Patterson, Consequences and categories of SRAM FPGA configuration SEUs, in: Int. Conf. MAPLD, 2003.

[21] N. Rollins, M. Wirthlin, M. Caffrey, P. Graham, Reliability of programmable input/output pins in the presence of configuration upsets, in: MAPLD '02 Laurel MD, 2002.

[22] F.Z. Tazi, C. Thibeault, Y. Savaria, S. Pichette, Y. Audet, On extra delays affecting I/O blocks of an SRAM-based FPGA due to ionizing radiation, IEEE Trans. Nucl. Sci. 61 (2014) 3138–3145, http://dx.doi.org/10.1109/tns.2014.2369417.

[23] F.Z. Tazi, C. Thibeault, Y. Savaria, Detailed analysis of radiation-induced delays on I/O blocks of an SRAM-based FPGA, in: Canadian Conference on Electrical and Computer Engineering, Vol. 2016-October, Institute of Electrical and Electronics Engineers Inc., 2016, http://dx.doi.org/10.1109/CCECE.2016.7726600.

[24] V.M. Placinta, L.N. Cojocariu, C. Ravariu, Proton-induced radiation effects in the I/O blocks of an SRAM-based FPGA, J. Instrum. 14 (2019) T10001, http://dx.doi.org/10.1088/1748-0221/14/10/T10001.

[25] J. Yang, H. Guo, L. Chen, Y. Zhang, X. Li, SEU sensitivity evaluation of JTAG circuit used for SRAM-based FPGA, in: 2017 IEEE 24th International Symposium on the Physical and Failure Analysis of Integrated Circuits, IPFA, IEEE, 2017, pp. 1–5, http://dx.doi.org/10.1109/IPFA.2017.8060186.

[26] J.D. Anderson, J.C. Leavitt, M.J. Wirthlin, Neutron radiation beam results for the Xilinx UltraScale+ MPSoC, in: 2018 IEEE Nuclear & Space Radiation Effects Conference, NSREC 2018, IEEE, 2018, pp. 1–7, http://dx.doi.org/10.1109/NSREC.2018.8584297.

[27] K. Höflinger, S. Müller, T. Peng, M. Ulmer, D. Lüdtke, A. Gerndt, Dynamic fault tree analysis for a distributed onboard computer, in: IEEE Aerospace Conference Proceedings, Vol. 2019-March, IEEE Computer Society, 2019, http://dx.doi.org/10.1109/AERO.2019.8742128.

[28] Versal ACAP AI Engine Architecture Manual, Xilinx, Inc., 2021.

[29] F.F.D. Santos, P.F. Pimenta, C. Lunardi, L. Draghetti, L. Carro, D. Kaeli, P. Rech, Analyzing and increasing the reliability of convolutional neural networks on GPUs, IEEE Trans. Reliab. 68 (2019) 663–677, http://dx.doi.org/10.1109/TR.2018.2878387.

[30] G. Li, S.K.S. Hari, M. Sullivan, T. Tsai, K. Pattabiraman, J. Emer, 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, Vol. 12, SC 2017, Association for Computing Machinery, Inc, 2017, http://dx.doi.org/10.1145/3126908.3126964.

[31] D.A.G.D. Oliveira, L.L. Pilla, T. Santini, P. Rech, Evaluation and mitigation of radiation-induced soft errors in graphics processing units, IEEE Trans. Comput. 65 (2016) 791–804, http://dx.doi.org/10.1109/TC.2015.2444855.

[32] X. Du, C. He, S. Liu, D. Luo, X. Du, W. Yang, Y. Li, Y. Fan, Analysis of sensitive blocks of soft errors in the Xilinx Zynq-7000 system-on-chip, Nucl. Instrum. Methods Phys. Res. A 940 (2019) 125–128, http://dx.doi.org/10.1016/J.NIMA.2019.06.015.

[33] Y. Li, E. Cheng, S. Makar, S. Mitra, Self-repair of uncore components in robust system-on-chips: An OpenSPARC T2 case study, in: Proceedings - International Test Conference, 2013, http://dx.doi.org/10.1109/TEST.2013.6651907.

[34] H. Cho, C.Y. Cher, T. Shepherd, S. Mitra, Understanding soft errors in uncore components, in: Proceedings - Design Automation Conference, Vol. 2015-July, Institute of Electrical and Electronics Engineers Inc., 2015, http://dx.doi.org/10.1145/2744769.2744923.

[35] H. Cho, E. Cheng, T. Shepherd, C.Y. Cher, S. Mitra, System-level effects of soft errors in uncore components, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 36 (2017) 1497–1510, http://dx.doi.org/10.1109/TCAD.2017.2651824.

[36] Versal ACAP Programmable Network on Chip and Integrated Memory Controller, Xilinx, Inc., 2021.

[37] D. Park, C. Nicopoulos, J. Kim, N. Vijaykrishnan, C.R. Das, Exploring fault-tolerant network-on-chip architectures, in: Proceedings of the International Conference on Dependable Systems and Networks, Vol. 2006, 2006, pp. 93–104, http://dx.doi.org/10.1109/DSN.2006.35.

[38] A. Prodromou, A. Panteli, C. Nicopoulos, Y. Sazeides, NoCAlert: An on-line and real-time fault detection mechanism for network-on-chip architectures, in: Proceedings - 2012 IEEE/ACM 45th International Symposium on Microarchitecture, MICRO 2012, 2012, pp. 60–71, http://dx.doi.org/10.1109/MICRO.2012.15.

[39] Y. Chen, H. Chen, X. Zhang, P. Lai, Failure localization and mechanism analysis in system-on-chip (SOC) using advanced failure analysis techniques, in: ICEPT-HDP 2012 Proceedings - 2012 13th International Conference on Electronic Packaging Technology and High Density Packaging, 2012, pp. 1348–1351, http://dx.doi.org/10.1109/ICEPT-HDP.2012.6474856.

[40] Intel Agilex Hard Processor System Technical Reference Manual, Intel, 2021.

[41] Zynq UltraScale+ Device Technical Reference Manual, Xilinx, Inc., 2020.

[42] Versal ACAP Technical Reference Manual, Xilinx, Inc., 2021.

[43] Intel Stratix 10 SEU Mitigation User Guide, Intel, 2021.

[44] Intel Agilex SEU Mitigation User Guide, Intel, 2021.

[45] PolarFire SoC MSS Technical Reference Manual, Microchip Technology Inc., 2021.

[46] K.A. Hoque, O.A. Mohamed, Y. Savaria, C. Thibeault, Probabilistic model checking based DAL analysis to optimize a combined TMR-blind-scrubbing mitigation technique for FPGA-based aerospace applications, in: 12th ACM/IEEE International Conference on Methods and Models for System Design, MEMOCODE 2014, Institute of Electrical and Electronics Engineers Inc., 2014, pp. 175–184, http://dx.doi.org/10.1109/MEMCOD.2014.6961856.

[47] J. Vidmar, P. Maillard, T. Jones, M. Sawant, G. Gambardella, N. Fraser, Space DPU: Constructing a radiation-tolerant, FPGA-based platform for deep learning acceleration on space payloads, in: OBDP2021 - 2nd European Workshop on on-Board Data Processing, OBDP2021, 2021, http://dx.doi.org/10.5281/ZENODO.5639613.

[48] A.B.D. Oliveira, L.A. Tambara, F. Benevenuti, L.A. Benites, N. Added, V.A. Aguiar, N.H. Medina, M.A. Silveira, F.L. Kastensmidt, Evaluating soft core RISC-V processor in SRAM-based FPGA under radiation effects, IEEE Trans. Nucl. Sci. 67 (2020) 1503–1510, http://dx.doi.org/10.1109/TNS.2020.2995729.

[49] J. Lojda, J. Podivinsky, Z. Kotasek, Reliability indicators for automatic design and analysis of fault-tolerant FPGA systems, in: LATS 2019 - 20th IEEE Latin American Test Symposium, Institute of Electrical and Electronics Engineers Inc., 2019, http://dx.doi.org/10.1109/LATW.2019.8704593.

[50] A.B. De Oliveira, G.S. Rodrigues, F.L. Kastensmidt, N. Added, E.L. Macchione, V.A. Aguiar, N.H. Medina, M.A. Silveira, Lockstep dual-core ARM A9: implementation and resilience analysis under heavy ion-induced soft errors, IEEE Trans. Nucl. Sci. 65 (8) (2018) 1783–1790, http://dx.doi.org/10.1109/TNS.2018.2852606.

[51] S. Kasap, E.W. Wachter, X. Zhai, S. Ehsan, K.D. McDonald-Maier, Novel lockstep-based approach with roll-back and roll-forward recovery to mitigate radiation-induced soft errors, in: 2020 IEEE Nordic Circuits and Systems Conference, NORCAS 2020 - Proceedings, Institute of Electrical and Electronics Engineers Inc., 2020, http://dx.doi.org/10.1109/NORCAS51424.2020.9265137.

[52] M. Pignol, DMT and DT2: Two fault-tolerant architectures developed by CNES for COTS-based spacecraft supercomputers, in: Proceedings - IOLTS 2006: 12th IEEE International on-Line Testing Symposium, Vol. 2006, 2006, pp. 203–212, http://dx.doi.org/10.1109/IOLTS.2006.24.

[53] A. Shye, J. Blomstedt, T. Moseley, V.J. Reddi, D.A. Connors, PLR: A software approach to transient fault tolerance for multicore architectures, IEEE Trans. Dependable Secure Comput. 6 (2) (2009) 135–148, http://dx.doi.org/10.1109/TDSC.2008.62.

[54] H. Mushtaq, Z. Al-Ars, K. Bertels, Fault tolerance on multicore processors using deterministic multithreading, in: 2013 8th IEEE Design and Test Symposium, IEEE, 2013.

[55] H. So, M. Didehban, A. Shrivastava, K. Lee, A software-level redundant multithreading for soft/hard error detection and recovery, in: Proceedings of the 2019 Design, Automation and Test in Europe Conference and Exhibition, DATE 2019, Institute of Electrical and Electronics Engineers Inc., 2019, pp. 1559–1562, http://dx.doi.org/10.23919/DATE.2019.8715089.

[56] J. Liu, Z. Zhu, C. Deng, A novel and adaptive transient fault-tolerant algorithm considering timing constraint on heterogeneous systems, IEEE Access 8 (2020) 103047–103061, http://dx.doi.org/10.1109/ACCESS.2020.2999092.

[57] J. Li, Y. Wang, Transient fault tolerance on multicore processor in AMP mode, in: Proceedings - 2021 8th International Conference on Dependable Systems and their Applications, DSA 2021, Institute of Electrical and Electronics Engineers Inc., 2021, pp. 332–337, http://dx.doi.org/10.1109/DSA52907.2021.00051.

[58] C.M. Fuchs, N.M. Murillo, A. Plaat, E. van der Kouwe, D. Harsono, P. Wang, Software-defined dependable computing for spacecraft, in: 2018 IEEE 23rd Pacific Rim International Symposium on Dependable Computing, PRDC, IEEE, 2018, http://dx.doi.org/10.1109/prdc.2018.00043.

[59] L.A. Benites, F. Benevenuti, A.B. De Oliveira, F.L. Kastensmidt, N. Added, V.A. Aguiar, N.H. Medina, M.A. Guazzelli, Reliability calculation with respect to functional failures induced by radiation in TMR ARM Cortex-M0 soft-core embedded into SRAM-based FPGA, IEEE Trans. Nucl. Sci. 66 (7) (2019) 1433–1440, http://dx.doi.org/10.1109/TNS.2019.2921796.

[60] R.V. Kshirsagar, S. Sharma, Comparative study of the proposed shifting algorithm for fault tolerance in FPGA, in: 2014 International Conference on Advances in Engineering and Technology Research, ICAETR 2014, Institute of Electrical and Electronics Engineers Inc., 2014, http://dx.doi.org/10.1109/ICAETR.2014.7012944.

[61] S. Fouad, F. Ghaffari, M.E.A. Benkhelifa, B. Granado, Context-aware resources placement for SRAM-based FPGA to minimize checkpoint/recovery overhead, in: 2014 International Conference on Reconfigurable Computing and FPGAs, ReConFig 2014, Institute of Electrical and Electronics Engineers Inc., 2014, http://dx.doi.org/10.1109/RECONFIG.2014.7032506.

[62] F. Lima Kastensmidt, L. Sterpone, L. Carro, M. Sonza Reorda, On the optimal design of triple modular redundancy logic for SRAM-based FPGAs, in: Proceedings – Design, Automation and Test in Europe, Vol. II, DATE '05, 2005, pp. 1290–1295, http://dx.doi.org/10.1109/DATE.2005.229.

[63] F. Lahrach, A. Doumar, E. Châtelet, Fault tolerance of SRAM-based FPGA via configuration frames, in: Proceedings of the 2011 IEEE Symposium on Design and Diagnostics of Electronic Circuits and Systems, DDECS 2011, 2011, pp. 139–142, http://dx.doi.org/10.1109/DDECS.2011.5783066.

[64] M.A. Hanif, M. Shafique, Dependable Deep Learning: Towards Cost-Efficient Resilience of Deep Neural Network Accelerators against Soft Errors and Permanent Faults, in: Proceedings - 2020 26th IEEE International Symposium on on-Line Testing and Robust System Design, IOLTS 2020, Institute of Electrical and Electronics Engineers Inc., 2020, http://dx.doi.org/10.1109/IOLTS50870.2020.9159734.

[65] L.H. Hoang, M.A. Hanif, M. Shafique, FT-ClipAct: Resilience Analysis of Deep Neural Networks and Improving their Fault Tolerance using Clipped Activation, in: Proceedings of the 2020 Design, Automation and Test in Europe Conference and Exhibition, DATE 2020, Institute of Electrical and Electronics Engineers Inc., 2020, pp. 1241–1246, http://dx.doi.org/10.23919/DATE48585.2020.9116571, arXiv:1912.00941.

[66] K. Zhao, S. Di, S. Li, X. Liang, Y. Zhai, J. Chen, K. Ouyang, F. Cappello, Z. Chen, FT-CNN: Algorithm-based fault tolerance for convolutional neural networks, IEEE Trans. Parallel Distrib. Syst. 32 (7) (2021) 1677–1689, http://dx.doi.org/10.1109/TPDS.2020.3043449, arXiv:2003.12203.

[67] T. Horita, T. Murata, I. Takanami, A multiple-weight-and-neuron-fault tolerant digital multilayer neural network, in: Proceedings - IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, 2006, pp. 554–562, http://dx.doi.org/10.1109/DFT.2006.8.

[68] J. Budroweit, H. Patscheider, Risk assessment for the use of COTS devices in space systems under consideration of radiation effects, Electronics 10 (9) (2021) http://dx.doi.org/10.3390/electronics10091008.

[69] G. Lentaris, K. Maragos, I. Stratakos, L. Papadopoulos, O. Papanikolaou, D. Soudris, M. Lourakis, X. Zabulis, D. Gonzalez-Arjona, G. Furano, High-performance embedded computing in space: Evaluation of platforms for vision-based navigation, J. Aerosp. Inf. Syst. 15 (4) (2018) 178–192, http://dx.doi.org/10.2514/1.I010555, arXiv:10.2514/1.I010555.

[70] R. Glein, F. Rittner, A. Becher, D. Ziener, J. Frickel, J. Teich, A. Heuberger, Reliability of space-grade vs. COTS SRAM-based FPGA in N-modular redundancy, in: 2015 NASA/ESA Conference on Adaptive Hardware and Systems, AHS, 2015, pp. 1–8, http://dx.doi.org/10.1109/AHS.2015.7231159.

[71] S.T. Fleming, D.B. Thomas, F. Winterstein, A power-aware adaptive FDIR framework using heterogeneous system-on-chip modules, in: F. Kastensmidt, P. Rech (Eds.), FPGAs and Parallel Architectures for Aerospace Applications: Soft Errors and Fault-Tolerant Design, Springer International Publishing, Cham, 2016, pp. 75–90, http://dx.doi.org/10.1007/978-3-319-14352-1_6.

[72] K. Shibin, S. Devadze, A. Jutman, M. Grabmann, R. Pricken, Health management for self-aware SoCs based on IEEE 1687 infrastructure, IEEE Des. Test 34 (6) (2017) 27–35, http://dx.doi.org/10.1109/MDAT.2017.2750902.

[73] E. Cheng, S. Mirkhani, L.G. Szafaryn, C.-Y. Cher, H. Cho, K. Skadron, M.R. Stan, K. Lilja, J.A. Abraham, P. Bose, S. Mitra, CLEAR: Cross-layer exploration for architecting resilience - combining hardware and software techniques to tolerate soft errors in processor cores, in: Proceedings - Design Automation Conference, Vol. 05-09-June-2016, Institute of Electrical and Electronics Engineers Inc., 2016, http://dx.doi.org/10.1145/2897937.2897996, http://arxiv.org/abs/1604.03062.

[74] A. Savino, A. Vallero, S.D. Carlo, ReDO: Cross-layer multi-objective design-exploration framework for efficient soft error resilient systems, IEEE Trans. Comput. 67 (2018) 1462–1477, http://dx.doi.org/10.1109/TC.2018.2818735.

[75] A. Kritikakou, O. Sentieys, G. Hubert, Y. Helen, J.-F. Coulon, P. Deroux-Dauphin, FLODAM: Cross-layer reliability analysis flow for complex hardware designs, in: DATE 2022 - 25th IEEE/ACM Design, Automation and Test in Europe, Antwerp, Belgium, 2022, pp. 1–6.

Appendix 2

II

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

AdAM: Adaptive Fault-Tolerant Approximate Multiplier for Edge DNN Accelerators

Mahdi Taheri[1], Natalia Cherezova[1*], Samira Nazari[3*], Ahsan Rafiq[1], Ali Azarpeyvand[1,3], Tara Ghasempouri[1], Masoud Daneshtalab[1,2], Jaan Raik[1] and Maksim Jenihhin[1]

[1]Tallinn University of Technology, Tallinn, Estonia
[2]Mälardalen University, Västerås, Sweden
[3]University of Zanjan, Zanjan, Iran
[1]mahdi.taheri@taltech.ee

2024 IEEE European Test Symposium (ETS) | 979-8-3503-4932-0/24/$31.00 ©2024 IEEE | DOI: 10.1109/ETS61313.2024.10567161

***Abstract*—Multiplication is the most resource-hungry operation in the neural network's processing elements. In this paper, we propose an architecture of a novel adaptive fault-tolerant approximate multiplier tailored for ASIC-based DNN accelerators. AdAM employs an adaptive adder relying on an unconventional use of the leading one position value of the inputs for fault detection through the optimization of unutilized adder resources. The proposed architecture uses a lightweight fault mitigation technique that sets the detected faulty bits to zero. The hardware resource utilization and the DNN accelerator's reliability metrics are used to compare the proposed solution against the triple modular redundancy (TMR) in multiplication, unprotected exact multiplication, and unprotected approximate multiplication. It is demonstrated that the proposed architecture enables a multiplication with a reliability level close to the multipliers protected by TMR utilizing 63.54% less area and having 39.06% lower power-delay product compared to the exact multiplier.**

***Index Terms*—deep neural networks, approximate computing, circuits design, reliability, resiliency assessment**

I. Introduction

The role of Deep Neural Networks (DNNs) in a wide range of safety- and mission-critical applications (e.g., autonomous driving) is expanding. Therefore, deployment of a DNN accelerator requires addressing the trade-off between different design parameters and *reliability* [1] [2]. Even though DNNs possess certain intrinsic fault-tolerant and error-resilient characteristics, it is insufficient to conclude the reliability of DNNs without considering the different characteristics of a hardware accelerator for vital applications. With the continuous scaling-down of the process, there is a discernible trend indicating that the Soft Error Rate (SER) of combinational circuits may surpass that of sequential circuits [3] [4]. Therefore, the main focus of this study is introducing a novel reliability technique to mitigate the soft errors in the combinational logic of an AI computation core.

This work presents the architecture of a novel adaptive fault-tolerant approximate multiplier (AdAM) tailored for ASIC-based DNN accelerators. Yet, the proposed multiplier can be implemented on FPGA as well. The contributions of the paper are as follows:

- The architecture of a novel adaptive fault-tolerant approximate multiplier tailored for DNN accelerators, including an adaptive adder relying on an unconventional use of the leading one position value of the inputs for fault detection through the optimization of unutilized adder resources
- Implementation and validation of the multiplier in a design synthesized for ASIC
- Reliability assessment and comparison of the proposed multiplier with exact and approximate state-of-the-art multipliers using several DNN benchmarks

The main objective of the proposed multiplier is to have the best trade-off between power-delay product (PDP) and vulnerability (accuracy drop due to the fault) which is demonstrated in the results section. Moreover, it is demonstrated that the proposed architecture enables a multiplication with a reliability level close to the multipliers protected by TMR utilizing 63.54% less area and having 39.06% lower PDP compared to the exact multiplier.

The remainder of the paper is organized as follows. Section II summarizes related works, the proposed method is presented in Section III, Section IV provides the experimental setup and discusses the results, and finally, the work is concluded in Section V.

II. Related Works

Multipliers are one of the primary arithmetic building blocks widely used in DNNs. Various approximate multipliers are proposed in the literature. ScaleTRIM is a scalable approximate unsigned LOD multiplier for DNNs that exploits curve fitting and linearization for fitting input products and a novel error compensation method using lookup tables [5]. More details about recent works on the approximation for DNNs can be found in [6].

Error introduced by approximation is deterministic and its impact can be studied on the accuracy drop of the network comprehensively. However, soft errors are unpredictable effects in contaminated and harsh environments that may lead to DNNs malfunction and accuracy drop drastically [7]. Recent research investigates the reliability of DNNs alongside approximation [8] [9] and quantization [10]. In [11], DNNs and approximated DNNs are tested in the presence of faults,

* These authors contributed equally

979-8-3503-4932-0/24/$31.00 ©2024 IEEE

Authorized licensed use limited to: Tallinn University of Technology. Downloaded on May 19,2026 at 09:11:42 UTC from IEEE Xplore. Restrictions apply.

and the results demonstrated that approximated DNNs are more resilient under special conditions.

To increase reliability and mitigate faults, Triple Modular Redundancy (TMR) and Gate-Sizing (GS) are two well-established hardening methods widely employed to mitigate the soft error rate in combinational circuits. Despite achieving 100% fault coverage for a single fault in one module of a combinational circuit, TMR incurs a substantial near 200% area and power overhead [12]. Therefore, numerous algorithms and frameworks are developed to enhance the efficiency of applying these methods and balance their hardening effects and design costs [13].

Approximate TMR (ATMR) is a technique that replaces some modules of TMR with approximate ones while ensuring the majority voter gives the correct output [14]. However, ATMR still requires duplicating the whole combinational circuit, even at the finest level of granularity.

To tackle this issue, this work presents an adaptive reliable multiplier that provides a high level of reliability while using less area than an exact multiplier.

III. AdAM architecture

The proposed architecture is an adaptive fault-tolerant approximate multiplier tailored for DNN accelerators. This architecture includes an adaptive adder relying on an unconventional use of the leading one position value of the inputs for fault *detection* and *mitigation* through the optimization of unutilized adder resources. The proposed multiplier is an adaptation of the classical Mitchell multiplier [15]. Mitchell multiplier employs approximate logarithms of the input values. By adding these logarithms, Mitchell's algorithm estimates the product. The final result is obtained by taking the antilogarithm of this sum. Another level of approximation is introduced in the adaptive adder considering the application of this multiplier in DNNs with a proven negligible impact on the network accuracy (see Subsection III-A and Table II).

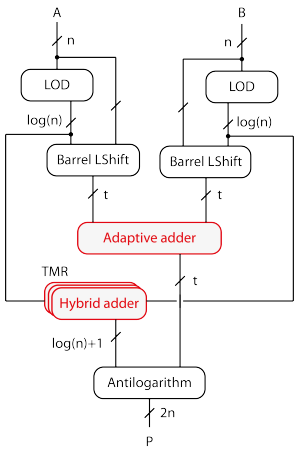


Fig. 1: AdAM architecture

The proposed architecture of the multiplier is presented in Fig. 1 (the contributions and extensions to the logarithmic Mitchell multiplier are marked with red color). Assuming each operand has $n = 8$ bits, a Leading One Detector (LOD) circuit is used to find the index of the first '1' bit in each operand. This index denoted as k, is the characteristic or integer part of the logarithm and has $log_2(n) = 3$ bits. The multiplier shifts the operands left by k bits, aligning the leading one with the Most Significant Bit (MSB). $(n–1)$ bits after the leading one represent the mantissa part denoted as m. The mantissa is truncated to $t = 5$ bits. The truncated operands are passed to the adaptive $(n–1)$-bit adder that adds mantissa together and duplicates the addition of 2 or 3 MSBs depending on the k value of the biggest operand for fault *detection* and *mitigation*. The architecture of the adaptive adder is shown in Fig. 2. The adder is based on the carry lookahead adder. Duplicated results are compared, and if there is a fault, the faulty bit is set to zero using AND gates (marked on the figure with a red rectangle). Due to the truncation of the mantissa, up to 2 Least Significant Bits (LSB) are excluded from the calculation, which affects only the bigger numbers with the k equal to 7 or 6. This introduces a small error compared to the original Mitchell algorithm that is discussed in the results section. The k values of the operands are added separately. This adder is replicated three times, as the order of the final output depends on the result of this addition. A majority voter selects the final result. Then, the antilogarithm algorithm is used to get the product of multiplication. The sum of k values determines the position of the leading one in the output product, which is followed by the sum of the mantissa parts using the appropriate shift operation.

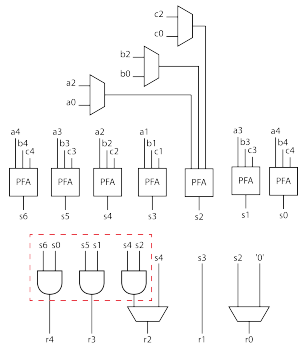


Fig. 2: Adaptive adder architecture: a and b are inputs, c is carry values and PFA stands for partial full adder

A. Adaptive adder

The adaptive adder is designed to perform fault detection and mitigation based on the LOD values of the multiplier inputs. Fig. 3 shows the scheme in which the proposed multiplier introduces fault tolerance. As shown in this figure, five cases are considered. If the maximum LOD of the inputs is 7, two LSBs are discarded, and a two-bit adder of the

Authorized licensed use limited to: Tallinn University of Technology. Downloaded on May 19,2026 at 09:11:42 UTC from IEEE Xplore. Restrictions apply.

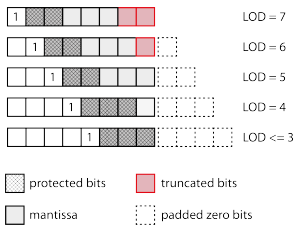


Fig. 3: Fault-tolerance and error introduced based on different LOD cases

adaptive adder is dedicated to recomputing the addition of two MSBs. These results are compared, and the mismatched bits are replaced by zeros in case of a mismatch.

For LOD = 6, only the LSB is discarded, and two higher-order bits are protected the same way as in the previous case. When LOD = 5, no bits are discarded, and two higher-order bits are protected. For LOD = 4, three higher-order bits are protected, and only LSB is not monitored. In the case of LOD $\leq$ 3, all bits are protected, enabling the proposed multiplier to provide comprehensive fault detection and mitigation for all inputs.

IV. Experimental Results

A. Experimental Setup

In this paper, the FreePDK 45 nm Nangate technology library is used in Cadence Genus 2023 to compare the hardware characteristics of the proposed methods with the state-of-the-art. The impact on the accuracy of the proposed adaptive multiplier is studied on different networks (i.e., LeNet-5, AlexNet, and VGG-16) trained on MNIST and CIFAR-10 using 8-bit INT with the help of the ADAPT framework [16]. Finally, the impact of the proposed multiplier on the reliability of DNNs is studied using AlexNet and VGG-16. To perform the reliability simulations for the case studies, a systolic-array simulator is developed and integrated into the ADAPT framework, and the impact of transient faults in the multiply-accumulate (MAC) units of the systolic array is studied in the network.

Random fault injection. Fault injection is performed, assuming the single bit-flip faults in the network's MAC operation of a systolic array for reliability assessment. Considering a prohibitively large number of fault combinations required for the multiple-bit fault model, it has been shown that high fault coverage obtained using the single-bit model results in a high fault coverage of multiple-bit faults [17]. According to the adopted single-bit fault model, a random bit-flip is injected into a random MAC unit of the systolic array core at a random execution time of the network, and the whole test set is fed to the network to obtain the accuracy of the network. This process is repeated several times to reach an acceptable confidence level, based on [18]. This work provides an equation to reach 95% confidence level and 1% error margin.

B. Hardware utilization

In this section, the adaptive multiplier is compared in terms of power and area with state-of-the-art designs.

In Table I, the accuracy, efficiency, and fault tolerance (FT) of 8-bit approximate multipliers are compared with the proposed method. Wallace, DRUM [19], TOSAM [20], and ScaleTrim [5] are used for this comparison. The accuracy is reported using Mean Absolute Relative Error (MARE). The proposed multiplier has similar hardware parameters to the state-of-the-art approximate multipliers with similar accuracy while providing reliability improvement with fault detection and mitigation capability.

TABLE I: Accuracy and efficiency of 8-bit approximate multipliers compared with the proposed method

Multiplier Architecture	Delay (ns)	Power (μW)	Area (μm^2)	MARE (%)	FT	PDP (pJ)
Exact (Wallace)	0.85	360	417	0.00	No	306
DRUM(3)	0.70	104	143	12.6	No	72.8
TOSAM(0,3)	0.68	144	198	7.7	No	97.9
DRUM(4)	1.00	172	208	6.4	No	172
TOSAM(1,5)	0.88	231	291	4.1	No	203.2
ScaleTrim(4,8)	1.8	143	216	3.3	No	257.4
AdAM	1.13	165	152	4.7	Yes	186.45

C. DNN accuracy

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

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

DNN	Baseline accuracy (%)	With proposed multiplier (%)
LeNet-5 (MNIST)	93.8	94.1
AlexNet (CIFAR-10)	78.0	77.7
VGG-16 (CIFAR-10)	93.4	94.0

D. Reliability analysis

To showcase the impact of the AdAM multiplier on reliability, the fault injection simulations are performed on AlexNet and VGG-16 with four different configurations. The DNN reliability is evaluated by comparing the output probability vector of the golden run (i.e. the DNN that behaves as expected, without faults) and the faulty run (i.e. the DNN that includes the fault). The SDC rate is defined as the proportion of faults that caused misclassification in comparison with the golden model. Since in DNNs, there is often not a single correct output, but a list of ranked outputs, each with a confidence score, the new criteria to determine what constitutes an SDC for a DNN application is defined in [21].

Authorized licensed use limited to: Tallinn University of Technology. Downloaded on May 19,2026 at 09:11:42 UTC from IEEE Xplore. Restrictions apply.

Fig. 4 demonstrates the fault tolerance comparison and reliability improvement of different networks by using the exact unprotected multipliers, using approximate unprotected multipliers (ScaleTRIM), using exact multipliers protected with TMR, and using AdAM. As illustrated, TMR has 100% of protection but it also requires about 200% of area overhead. Despite using TMR in our architecture for a small adder, we introduce very high reliability improvement without introducing hardware overhead. Since the main objective of the proposed multiplier is to have the best trade-off between PDP and vulnerability (accuracy drop due to the fault), Fig. 5 illustrates this comparison. In these charts, the closer to the origins (0,0), the higher the cost-efficiency of the fault tolerance, i.e. lower vulnerability and PDP. As shown, TMR is an inefficient solution for edge AI applications because of its high PDP, while the proposed method (AdAM) is the closest to the origin.

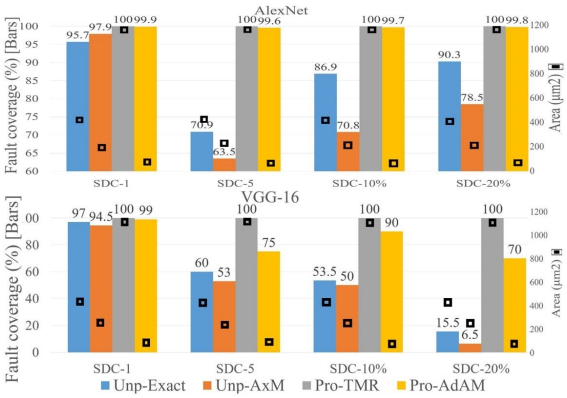


Fig. 4: Hardware efficiency (area) and fault resilience (fault coverage) trade-offs in AlexNet (up) and VGG-16 (down). Unp-exact: unprotected exact multiplier, Unp-AxM: unprotected approximate multiplier, Pro-TMR: exact multiplier protected by TMR, Pro-AdAM: proposed multiplier

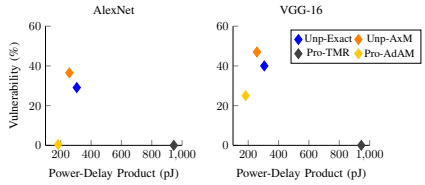


Fig. 5: PDP and vulnerability tradeoffs (considering SDC-5) in different methods

V. Conclusion

In this paper, we propose an architecture of a novel adaptive fault-tolerant approximate multiplier tailored for ASIC-based DNN accelerators. AdAM employs an adaptive adder relying on an unconventional use of the leading one position value of the inputs for fault detection through the optimization of unutilized adder resources. The proposed architecture uses a lightweight fault mitigation technique that sets the detected faulty bits to zero. It is demonstrated that the proposed multiplier provides a reliability level close to the multipliers protected by Triple Modular Redundancy (TMR) while utilizing 63.54% less area and having 39.06% lower power-delay product compared to the exact multiplier.

VI. Acknowledgement

This work was supported in part by the Estonian Research Council grant PUT PRG1467 ”CRASHLESS“ and by Estonian-French PARROT project ”EnTrustED”.

References

[1] M. H. Ahmadilivani and et al, “A systematic literature review on hardware reliability assessment methods for deep neural networks,” *arXiv preprint arXiv:2305.05750*, 2023.
[2] M. Taheri, “Dnn hardware reliability assessment and enhancement,” *27th IEEE European Test Symposium (ETS).*, May 2022.
[3] N. Mahatme and et al, “Comparison of combinational and sequential error rates for a deep submicron process,” *TNS*, pp. 2719–2725, 2011.
[4] M. Taheri and et al, “A novel fault-tolerant logic style with self-checking capability,” in *IOLTS*. IEEE, 2022, pp. 1–6.
[5] E. Farahmand and et al, “ScaleTRIM: scalable truncation-based integer approximate multiplier with linearization and compensation,” *arXiv preprint arXiv:2303.02495*, 2023.
[6] G. Armeniakos and et al, “Hardware approximate techniques for deep neural network accelerators: A survey,” *ACM Comput. Surv.*, vol. 55, no. 4, nov 2022.
[7] G. Li and et al, “Understanding error propagation in deep learning neural network (DNN) accelerators and applications,” in *SC’17*, 2017.
[8] M. H. Ahmadilivani and et al, “Special session: Approximation and fault resiliency of DNN accelerators,” in *VTS*, 2023, pp. 1–10.
[9] M. Taheri and et al, “Appraiser: Dnn fault resilience analysis employing approximation errors,” in *DDECS*. IEEE, 2023, pp. 124–127.
[10] M. Taheri, N. Cherezova, and et al, “Exploration of activation fault reliability in quantized systolic array-based dnn accelerators,” *arXiv preprint arXiv:2401.09509*, 2024.
[11] M. Taheri, M. H. Ahmadilivani, and et al, “Deepaxe: A framework for exploration of approximation and reliability trade-offs in DNN accelerators,” in *ISQED*. IEEE, 2023, pp. 1–8.
[12] S. Mittal, “A survey on modeling and improving reliability of DNN algorithms and accelerators,” *J. Syst. Archit.*, vol. 104, p. 101689, 2020.
[13] G. Ammes and et al, “ATMR design by construction based on two-level ALS,” in *36th SBCCI*. IEEE, 2023, pp. 1–6.
[14] T. Arifeen and et al, “Approximate triple modular redundancy: A survey,” *IEEE Access*, vol. 8, pp. 139 851–139 867, 2020.
[15] J. N. Mitchell, “Computer multiplication and division using binary logarithms,” *IRE Trans. on Electronic Computers*, pp. 512–517, 1962.
[16] D. Danopoulos and et al, “Adapt: Fast emulation of approximate DNN accelerators in Pytorch,” *IEEE TCAD*, 2022.
[17] M. Bushnell and V. Agrawal, *Essentials of electronic testing for digital, memory and mixed-signal VLSI circuits*. SSBM, 2004, vol. 17.
[18] R. Leveugle and et al, “Statistical fault injection: Quantified error and confidence,” in *DATE*, 2009, pp. 502–506.
[19] S. Hashemi, R. I. Bahar, and S. Reda, “Drum: A dynamic range unbiased multiplier for approximate applications,” in *ICCAD*, 2015, pp. 418–425.
[20] S. Vahdat and et al, “TOSAM: an energy-efficient truncation-and rounding-based scalable approximate multiplier,” *TVLSI*, vol. 27, no. 5, pp. 1161–1173, 2019.
[21] G. Li, S. K. S. Hari, M. Sullivan, T. Tsai, K. Pattabiraman, J. Emer, and S. W. Keckler, “Understanding error propagation in deep learning neural network (dnn) accelerators and applications,” in *SC17*, 2017.

Authorized licensed use limited to: Tallinn University of Technology. Downloaded on May 19,2026 at 09:11:42 UTC from IEEE Xplore. Restrictions apply.

Appendix 3

III

M. Taheri, N. Cherezova, S. Nazari, A. Azarpeyvand, T. Ghasempouri, M. Daneshtalab, J. Raik, and M. Jenihhin. AdAM: Adaptive approximate multiplier for fault tolerance in DNN accelerators. *IEEE Transactions on Device and Materials Reliability*, 25(1):66–75, 2025

AdAM: Adaptive Approximate Multiplier for Fault Tolerance in DNN Accelerators

Mahdi Taheri, *Graduate Student Member, IEEE*, Natalia Cherezova, Samira Nazari, Ali Azarpeyvand, Tara Ghasempouri, Masoud Daneshtalab, Jaan Raik, *Member, IEEE*, and Maksim Jenihhin, *Member, IEEE*

***Abstract*—Deep Neural Network (DNN) hardware accelerators are essential in a spectrum of safety-critical edge-AI applications with stringent reliability, energy efficiency, and latency requirements. Multiplication is the most resource-hungry operation in the neural network's processing elements. This paper proposes a scalable adaptive fault-tolerant approximate multiplier (AdAM) tailored for ASIC-based DNN accelerators at the algorithm and circuit levels. AdAM employs an adaptive adder that relies on an unconventional use of input Leading One Detector (LOD) values for fault detection by optimizing unutilized adder resources. A gate-level optimized LOD design and a hybrid adder design are also proposed as a part of the adaptive multiplier to improve the hardware performance. The proposed architecture uses a lightweight fault mitigation technique that sets the detected faulty bits to zero. The hardware resource utilization and the DNN accelerator's reliability metrics are used to compare the proposed solution against the Triple Modular Redundancy (TMR) in multiplication, unprotected exact multiplication, and unprotected approximate multiplication. It is demonstrated that the proposed architecture enables a multiplication with a reliability level close to the multipliers protected by TMR while at the same time utilizing 2.74× less area and with 39.06% less power-delay product compared to the exact multiplier. Moreover, it has similar area, delay, and power consumption parameters compared to the state-of-the-art approximate multipliers with similar accuracy while providing fault detection and mitigation capability.**

***Index Terms*—Deep neural networks, approximate computing, circuit design, reliability, DNN accelerator.**

I. Introduction

In THE past decades, Deep Neural Networks (DNNs) demonstrated a significant improvement in accuracy by adopting computationally intense models [1]. Consequently, the size of these models has increased drastically, imposing challenges in their deployment on resource-constrained platforms [2].

Different DNN compression techniques such as model quantization and pruning [2] as well as approximate computing (AxC) [3], [4] enable the use of DNNs in edge devices. While these techniques decrease the accuracy of DNNs, they bring the benefits of lower resource utilization and energy consumption and higher system efficiency [5]. As an example, quantizing DNNs down to 8-bit INT gained popularity in edge AI applications, because of the minimal impact on accuracy drop and a significant reduction in memory footprint [6].

On the other hand, the role of DNNs in a wide range of safety- and mission-critical applications, e.g., autonomous driving, is expanding. Therefore, deploying a DNN accelerator requires addressing the trade-off between different design parameters and *reliability* [4]. Although DNNs possess certain intrinsic fault-tolerant and error-resilient characteristics, it is insufficient to conclude the reliability of DNNs without considering the characteristics of the corresponding hardware accelerator.

Consider Fig. 1 that demonstrates different possible fault locations in an AI accelerator and their negative effect on the object detection task. In the example, a pedestrian has been identified as a bird, and a red light is misclassified as a green one leading to a potentially disastrous situation. In this work, which is an extension of the previous paper by the authors [8], the faults in the computational core of the AI chip are considered.

Moreover, with the continuous scaling-down of the process, there is a discernible trend indicating that the Soft Error Rate (SER) of combinational circuits may surpass that of sequential circuits [9], [10]. Therefore, the main focus of this study is introducing a novel reliability technique to mitigate soft errors in the combinational logic of DNN accelerators while keeping resource utilization and energy consumption low.

This work presents an architecture of an adaptive fault-tolerant approximate multiplier (AdAM) tailored for ASIC-based DNN accelerators. The multiplier is based on the logarithmic Mitchell's multiplier that substitutes multiplication with the addition of approximated logarithms of the operands. The proposed multiplier protects higher-order bits of the product based on the maximum position of the leading one bit in the input strands of the multiplier. This multiplier is

Received 20 September 2024; revised 21 December 2024; accepted 24 December 2024. Date of publication 27 December 2024; date of current version 19 March 2025. This work was supported in part by EU through the Project ''TAICHIP" under Grant 101160182; in part by the Estonian Research Council under Grant PUT PRG1467 "CRASHLESS"; and in part by the Estonian-French PARROT Project "EnTrustED." *(Mahdi Taheri and Natalia Cherezova contributed equally to this work.) (Corresponding author: Mahdi Taheri.)*

Mahdi Taheri, Natalia Cherezova, Tara Ghasempouri, Jaan Raik, and Maksim Jenihhin are with the Computer Systems Department, Tallinn University of Technology, 19086 Tallinn, Estonia (e-mail: mahdi.taheri@taltech.ee).

Samira Nazari is with the University of Zanjan, Zanjan 4516346119, Iran.

Ali Azarpeyvand is with the Tallinn University of Technology, 19086 Tallinn, Estonia, and also with the University of Zanjan, Zanjan 4516346119, Iran.

Masoud Daneshtalab is with the Computer Systems Department, Tallinn University of Technology, 19086 Tallinn, Estonia, and also with Mälardalen University, 722 20 Västerås, Sweden.

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TDMR.2024.3523386.

Digital Object Identifier 10.1109/TDMR.2024.3523386

1530-4388 © 2024 IEEE. All rights reserved, including rights for text and data mining, and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.
Authorized licensed use limited to: Tallinn University of Technology. Downloaded on May 19,2026 at 09:12:58 UTC from IEEE Xplore. Restrictions apply.

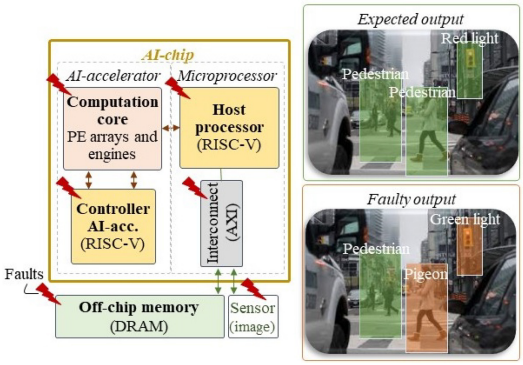


Fig. 1. Possible locations of faults in AI chips [7].

a negative overhead fault tolerance approximate multiplier compared to the exact multipliers. The contributions of the paper are as follows:

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

The proposed multiplier provides a reliability level close to the multipliers protected by Triple Modular Redundancy (TMR) while utilizing 2.74× less area and having 39% less power-delay product compared to the unprotected exact multiplier. In fact, it has similar area, delay, and power consumption parameters compared to the state-of-the-art approximate multipliers with similar accuracy while providing fault detection and mitigation capability.

The remainder of the paper is organized as follows: Section II summarizes related works. Section III presents the proposed method. Section IV presents the experimental setup and discusses the results. Finally, Section V concludes the work.

II. Related Work

Multipliers are one of the primary arithmetic building blocks widely used in DNNs. Approximate computing is a promising technique for designing digital circuits with lower area and power consumption while achieving a higher working frequency, particularly when the target application has some error resiliency, such as DNNs [3], [5]. Various approximate multipliers are proposed in the literature. These options encompass dynamic segment multiplier (DSM) structure [11], dynamic range unbiased multiplier (DRUM) structure [12], memristor-based multipliers [3], [13], simplified adder-based or truncated multipliers [14], utilization of inexact compressors [15], and approximate booth multipliers [16]. One of the other approximation techniques to further speed up the multiplication is to move to the logarithmic numbering system to compute addition instead of multiplication. The general idea is to obtain the logarithm of the inputs, calculate their sum, and convert the output value to the final result through an antilogarithm operation [17], [18]. The complexity and accuracy of this method come from the logarithm and antilogarithm steps. Truncating the input operand of multiplication and using logarithmic approximation also has a dual effect on the area, speed, and power consumption improvement. The first approximate logarithmic multiplier was proposed by Mitchell, who used binary logarithms to approximate multiplication [18]. Several studies have been conducted on improving the performance and accuracy of this method by considering DNN application [19]. TOSAM is a scalable approximate multiplier that reduces the number of partial products by truncating each of the input operands based on their leading one-bit position and improves delay, area, and energy consumption up to 41%, 90%, and 98% respectively [14]. ScaleTRIM is a scalable approximate unsigned LOD multiplier for DNNs that exploits curve fitting and linearization for fitting input products and a novel error compensation method using lookup tables [17]. More details about recent works on the approximate arithmetic circuits can be found in [20] and specifically about approximation for DNNs in [21].

The error introduced by the approximation is deterministic, and its impact can be studied comprehensively on the accuracy drop of the network. However, soft errors are unpredictable in contaminated and harsh environments that can lead to DNNs malfunction and accuracy drop drastically [22]. The design of approximate arithmetic circuits and their reliability have been extensively covered in the literature both without considering an application [23], [24], [25] and concerning DNNs [4], [5], [26], [27], [28], [29]. For example, the work presented in [30] introduces a comprehensive library of approximate adders and multipliers, primarily intended for use in image and video processing circuits. However, as highlighted in [31], there is a trade-off between fault resilience and energy efficiency when using the multipliers from this library. This indicates the need for specific approximate multiplier (AxM) designs to optimize energy savings and resource utilization while enhancing DNN accuracy. Additionally, the impact of approximate multipliers on DNN accuracy can vary significantly, with some leading to a substantial drop in performance. In contrast to the proposed approximate multipliers, AdAM considers reallocating resources saved by approximation for fault tolerance.

Traditional fault tolerance methods for DNN hardware accelerators include hardware redundancy, correction codes for parameter memories, and algorithm-based fault tolerance. Triple Modular Redundancy (TMR), despite achieving 100% fault coverage, incurs a substantial near 200% area and power overhead [32]. Therefore, numerous algorithms and frameworks are developed to enhance the efficiency of applying these methods and balance their hardening effects and design costs such as Approximate TMR (ATMR) [33]. ATMR is a technique that replaces some modules of TMR with approximate ones while ensuring that the majority voter gives the correct output [34]. However, ATMR still requires

Authorized licensed use limited to: Tallinn University of Technology. Downloaded on May 19,2026 at 09:12:58 UTC from IEEE Xplore. Restrictions apply.

duplicating the whole combinational circuit, even at the finest level of granularity [35], [36].

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

III. AdAM Architecture

We propose an architecture for adaptive fault-tolerant approximate multiplier tailored for DNN accelerators. This architecture includes an adaptive adder relying on an unconventional use of input Leading One Detector (LOD) values for fault *detection* and *mitigation* through the optimization of unutilized adder resources. A gate-level optimized LOD design and a lightweight triplicated hybrid adder design are used to enhance further the proposed architecture's reliability, resource utilization, and efficiency. The base for the proposed multiplier is the classical Mitchell multiplier [18]. However, the methodology can be applied to all logarithmic approximate multipliers. Another level of approximation is introduced in the adaptive adder (Fig. 3) considering the application of this multiplier in DNNs with a proven negligible impact on the network accuracy (see Table VI in the results section). Mitchell multiplier employs approximate logarithms of the input values. By adding these logarithms, Mitchell's algorithm estimates the product. The final result is obtained by taking the antilogarithm of this sum.

A. Definitions

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

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

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

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

$$B = 2^{k_B}\left(1 + \sum_{i=0}^{k_B-1} 2^{i-k_B} b_i\right) = 2^{k_B}(1 + X_B) \tag{3}$$

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

$$log(A) = k_A + log(1 + X_A) \tag{4}$$

$$log(B) = k_B + log(1 + X_B) \tag{5}$$

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

$$log(A) \approx k_A + X_A \tag{6}$$

$$log(B) \approx k_B + X_B \tag{7}$$

The logarithm of $A * B$ can be approximated as

$$log(A * B) \approx k_A + X_A + k_B + X_B \tag{8}$$

The antilogarithm of the above expression is the final product:

$$\hat{P} = 2^{k_A+k_B}(1 + X_A + X_B) \tag{9}$$

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

$$X_A^t = \sum_{i=0}^{k_A-1} 2^{i-k_A} a_i \cdot \left[|i - k_A| \geq n - t\right] \tag{10}$$

$$X_B^t = \sum_{i=0}^{k_B-1} 2^{i-k_B} b_i \cdot \left[|i - k_B| \geq n - t\right] \tag{11}$$

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

$$\hat{P} = 2^{k_A+k_B}\left(1 + X_A^t + X_B^t\right) \tag{12}$$

Truncation of the fractional parts introduces additional errors to the result. The absolute error $(P - \hat{P})$ behavior of the original 8-bit Mitchell's multiplier and AdAM is compared in Fig. 2 over the entire input domain. The maximum errors introduced by each method are marked in the figure. There is no additional error when the truncated bits in both operands are '0', as the number of '1' in the truncated bits increases, additional error also increases.

However, it has been shown in the literature that the trained synaptic weights in NN applications do not have a uniform distribution, and they are mostly centered around zero [37]. Since the proposed multiplier presents a higher level of protection for smaller values respecting the value of LOD, it can effectively protect most computations of the network during inference.

The impact of the proposed multiplier on the accuracy of different networks is reported in the result section.

The level of fault tolerance of the proposed multiplier is defined by two parameters: the aforementioned truncation parameter t and duplication level h. The truncation parameter t defines the minimum number of protected bits, and duplication level h defines the maximum possible number of protected bits. Smaller operands have smaller mantissa values that do not require the whole adder. Therefore, more adder resources can be used for fault tolerance by duplicating the addition of more higher-order bits. Duplication level h does not affect the accuracy of the multiplier, as a hardware-related parameter it affects area, power, and delay. Duplication level h defines the amount of adder resources that can be used for fault tolerance (see Section III-C for detailed description). A higher h value means a higher level of fault tolerance and higher resource utilization. A smaller value means a lower level of fault tolerance and lower resource utilization. Different configurations of the proposed multiplier are denoted as AdAM(t, h).

B. Hardware Implementation

The proposed architecture of the multiplier is presented in Fig. 3 (the contributions and extensions to the logarithmic

Authorized licensed use limited to: Tallinn University of Technology. Downloaded on May 19,2026 at 09:12:58 UTC from IEEE Xplore. Restrictions apply.

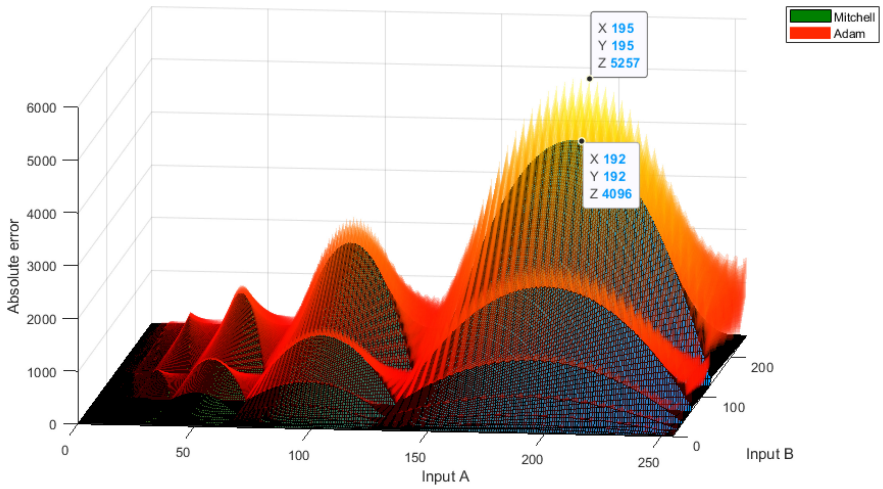


Fig. 2. Comparison of absolute error distribution between the original 8-bit Mitchell's multiplier and AdAM: top point shows the maximum absolute error for Adam, lower point shows the maximum absolute error for Mitchell's multiplier.

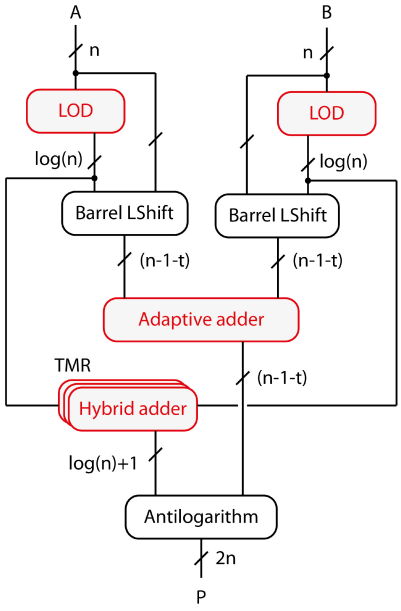


Fig. 3. AdAM architecture (the contributions and extensions to the logarithmic Mitchell multiplier are marked with red color).

Mitchell multiplier are marked with red color). First, a novel optimized *Leading One Detector (LOD)* circuit (Section III-D) is used to find the index of the first '1' bit in each operand. This index denoted as k, is the characteristic or integer part of the logarithm and has $log_2(n)$ bits. Then, the operands are shifted left by k bits, aligning the leading one with the Most Significant Bit (MSB). $(n-1)$ bits after the leading one represent the mantissa part denoted as m. The mantissa is truncated to $(n-1-t)$ bits.

The truncated operands are passed to the adaptive $(n-1)$-bit adder that adds mantissa together and duplicates the addition of several higher-order bits depending on the k value of the biggest operand for fault *detection* and *mitigation*. As it was already mentioned, the number of duplicated bits depends on truncation parameter t and duplication level h.

The architecture of the *adaptive adder* is shown in Fig. 4 (Section III-C). The adder is based on the carry-lookahead adder, and the carry generation logic is excluded from the figure to save space. Duplicated results are compared, and if there is a fault, the faulty bit is set to zero using AND gates (marked on the figure with a red rectangle). Due to the truncation of the mantissa, up to t lower-order bits are excluded from the calculation, which affects only the bigger numbers with the $k > (n-t)$. This introduces a small error compared to the original Mitchell algorithm (discussed in the following section).

The k values of the operands are added separately using a small *hybrid adder*. A hybrid between a resource-intensive but fast carry look-ahead adder and a lightweight but slow due to the accumulated delay ripple carry (CR) adder is selected for this task. This adder is replicated three times, for extra fault tolerance, as the order of the final output depends on the result of this addition. A majority voter selects the final result. Then, the antilogarithm algorithm is used to get the product of multiplication. The sum of k values determines the position of the leading one in the output product, followed by the sum of the mantissa parts using the appropriate shift operation.

C. Adaptive Adder

The adaptive adder is designed to detect and mitigate faults based on the multiplier inputs' k values. The adder is divided

Authorized licensed use limited to: Tallinn University of Technology. Downloaded on May 19,2026 at 09:12:58 UTC from IEEE Xplore. Restrictions apply.

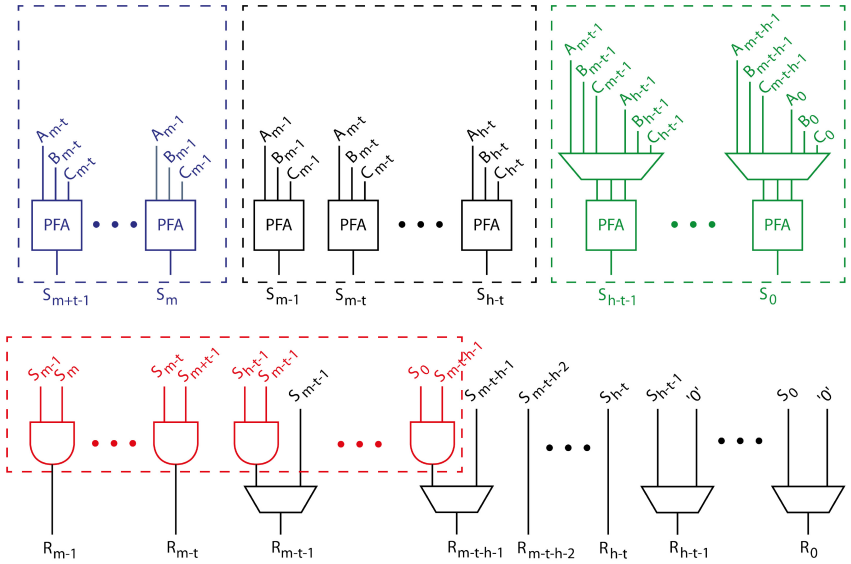


Fig. 4. Adaptive adder architecture: A and B are inputs, C is carry values, and PFA stands for partial full adder, m here denotes the size of the truncated mantissa $(n-1-t)$. The truncation parameter t defines the minimum number of protected bits, t PFAs that duplicate the addition of t higher-order bits of mantissa are colored blue. The duplication level h defines the maximum number of protected bits, $h-t$ PFAs with multiplexers that can duplicate the addition of higher-order bits or add lower-order bits are colored green.

into three parts: t PFAs (Partial Full Adders) for duplicating the addition of t higher-order bits (colored blue in Fig. 4), $(h-t)$ PFAS with multiplexers on inputs that can duplicate the addition of up to $(h-t)$ higher-order bits or add lower-order bits (colored green in Fig. 4), and $(n-1-h)$ PFAs for the addition of the mantissa. This way, for smaller operands that have smaller mantissa and do not require the whole adder, more resources can be used for fault tolerance. The size of the truncated mantissa can be expressed as $W_{X^t} = k \cdot [k < n-t] + (n-1-t) \cdot [k \geq n-t]$ using Iverson brackets notation. The available adaptive adder resources can be expressed as $W_{AR} = (n-1) - W_{X^t}$. Thus, the number of protected bits is $W_{PB} = W_{AR} \cdot [W_{AR} < h] + h \cdot [W_{AR} \geq h]$.

As an example, Fig. 5 shows the scheme in which the proposed multiplier introduces fault tolerance considering 8-bit inputs with a truncation parameter $t = 2$ and duplication level $h = 3$. As shown in this figure, five cases are considered. If the maximum LOD of the inputs is 7, two lower-order bits are discarded, and a two-bit adder of the adaptive adder is dedicated to recomputing the addition of two higher-order bits. These results are compared, and the mismatched bits are set to zero.

For $k = 6$, only the Least Significant Bit (LSB) is discarded, and two higher-order bits are protected the same way as in the previous case. When $k = 5$, no bits are discarded, and two higher-order bits are protected. For $k = 4$, three higher-order bits are protected, and only LSB is not monitored. In the case of $k \leq 3$, all bits are protected, enabling the proposed multiplier to provide comprehensive fault detection and mitigation for all inputs.

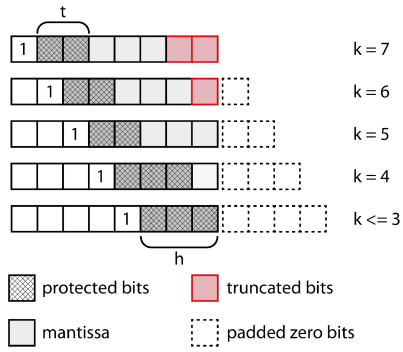


Fig. 5. Fault-tolerance and error introduced based on different input values $(n = 8, t = 2, h = 3)$.

D. LOD Design

The gate-level structure of the proposed LOD circuit is presented in Fig. 6. The circuit is divided into two functional parts: zero flag calculation and leading-one position detection (LOPD). LOPD consists of several stages. During the first stage, input is divided into nibbles that are processed in parallel. For each nibble, three signals are calculated except for the first one: a, b, and c. Only two signals are calculated for the first nibble: b and c. Signal a defines whether there is a '1' bit in the nibble; signal b defines whether the output is even or odd; and signal c defines whether there is '1' in the two higher bits of the nibble. During the second stage, those signals form the final result based on the highest nibble with a

Authorized licensed use limited to: Tallinn University of Technology. Downloaded on May 19,2026 at 09:12:58 UTC from IEEE Xplore. Restrictions apply.

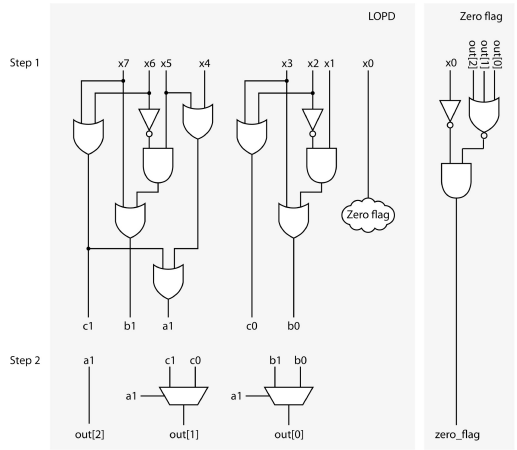


Fig. 6. Gate-level structure of the proposed LOD.

TABLE I
LOD HARDWARE COMPARISON BETWEEN THE PROPOSED METHOD AND THE STATE-OF-THE-ART

LOD Architecture	Delay (ps)	Power (μW)	PDP (ps$\times\mu$W)	Area (μm^2)
ScaleTrim [17]	156	1.12	174.72	9.84
Proposed	136	1.09	148.24	9.57

TABLE II
ACCURACY AND EFFICIENCY OF 8-BIT APPROXIMATE MULTIPLIERS COMPARED WITH THE PROPOSED METHOD

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

'1' bit. Since the zero flag is only required at the last stage of the multiplier to determine whether the final product should be zero, it is calculated using the LOPD output values. The proposed design requires fewer logic gates (as demonstrated in the results section) than designs found in the literature (e.g., ScaleTRIM [17]) by reusing the output values for calculating the zero flag, knowing that it can have a longer delay.

IV. EXPERIMENTAL RESULTS

A. Experimental Setup

In this paper, the FreePDK 45 nm Nangate technology library is used in Cadence Genus 2023 to compare the hardware characteristics (area, latency, power consumption) of the proposed methods with the state-of-the-art. The accuracy is reported using Mean Absolute Relative Error (MARE) calculated as

$$\text{MARE} = \frac{1}{N}\sum_{i=1}^{N}\left|\frac{P_i - \hat{P}_i}{P_i}\right|,$$

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

Additionally, a design of a MAC (multiply-accumulate) unit in a systolic array is synthesized for ASIC to better illustrate the results in an AI core. The impact on the accuracy of the proposed adaptive multiplier is studied on different networks (i.e., LeNet-5, AlexNet, ResNet-18, VGG-16, DenseNet) trained on MNIST and CIFAR-10 using 8-bit INT with the help of the AdaPT framework [38]. Finally, the impact of the proposed multiplier on the reliability of DNNs is studied using the mentioned benchmarks.

Fault injection. Fault injection is performed, assuming the single bit-flip faults in the network's MAC operation of a systolic array for reliability assessment. According to the adopted single-bit fault model, a random bit-flip is injected into a random MAC unit of the systolic array core at a random execution time of the network, and the whole test set is fed to the network to obtain the accuracy of the network. Transient faults (bit flips) within subcomponents of the multipliers, such as LOD, adders, and barrel shifters, were considered for the fault injection experiments. A simulation was done to determine how those faults will propagate to the output of the multiplier. For the fault propagation, the RTL model of the multiplier was utilized and a random set of potential fault positions within its unprotected components was identified. Faults were injected as bit flips during the simulation using Cadence Genus. For each considered fault position, all input combinations were applied to the multiplier to generate look-up tables with the outputs. Those look-up tables were used during the fault injection in the DNNs. The number of faults injected was chosen based on preliminary experiments and existing literature. Using equations provided in [39] a 95% confidence level and 1% error margin were reached.

B. Hardware Utilization

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

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

Tables II–IV report the accuracy, efficiency, and fault tolerance (FT) of 8-bit, 16-bit, and 32-bit approximate multipliers compared with the proposed method. Wallace, DRUM [12], TOSAM [14], and ScaleTrim [17] are used for this comparison. Logarithmic AxMs such as TOSAM show better performance compared to other state-of-the-art non-logarithmic AxMs, including DSM [11], ROBA [40], and LETAM [41]. For instance, TOSAM structures exhibit significantly lower energy consumption and area utilization compared to DRUM, DQ4:2C4, and AS-RoBA multipliers, while maintaining comparable accuracy. Specifically, the TOSAM(1,5) configuration achieves 4% lower delay, 78%

Authorized licensed use limited to: Tallinn University of Technology. Downloaded on May 19,2026 at 09:12:58 UTC from IEEE Xplore. Restrictions apply.

TABLE III
ACCURACY AND EFFICIENCY OF 16-BIT APPROXIMATE MULTIPLIERS COMPARED WITH THE PROPOSED METHOD

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

TABLE IV
ACCURACY AND EFFICIENCY OF 32-BIT APPROXIMATE MULTIPLIERS COMPARED WITH THE PROPOSED METHOD

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

lower area, and 70% lower energy consumption compared to the RoBA multiplier. The results also show that the overall impact of the TOSAM multiplier, one of the baseline approaches in this paper, on delay, power, area, and MARE, are lower than those of other approximate multipliers with similar MARE values. Therefore, to respect the page limits of the paper, we did not include other approximate multipliers in our comparison.

The proposed multiplier has similar hardware parameters to the state-of-the-art approximate multipliers with similar accuracy while providing reliability improvement with fault detection and mitigation capability.

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

C. DNN Accuracy

Table VI compares the accuracy of different CNN architectures using the proposed approximate multiplier with the baseline accuracy using the exact multiplier. The results in Table VI were obtained by directly substituting the accurate multiplier with the proposed multiplier without any retraining or fine-tuning. The evaluation shows that the accuracy of DNN with the proposed method is very close to the baseline. Hence, the proposed multiplier has a negligible effect on the accuracy of DNNs. It is observed that the accuracy of approximated networks can sometimes exceed that of exact networks. The observed higher accuracy with AdAM compared to the exact multiplier in some cases can be attributed to the probabilistic nature and inherent non-linearity of DNNs, as well as the sensitivity of certain bit positions to errors. This observation aligns with the previous studies [42] showing that errors in the arithmetics logic can result in better inference accuracy and consequently referring to them as "good faults". This behavior arises because faults, particularly those affecting lower-order bits (which can be interpreted as errors caused by approximate multipliers), do not always degrade accuracy. This phenomenon aligns with the broader understanding that DNNs are not strictly deterministic and can potentially exhibit more accuracy under specific fault conditions. Thus, the accuracy gains with AdAM can be seen conceptually as part of this broader context of DNN fault tolerance and behavior.

TABLE V
EFFICIENCY OF 8-BIT MAC UNIT OF A SYSTOLIC ARRAY WITH DIFFERENT MULTIPLIERS

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

TABLE VI
ACCURACY COMPARISON OF DIFFERENT CNNS WITH AN EXACT (BASELINE) AND THE PROPOSED APPROXIMATE MULTIPLIER

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

D. Reliability Analysis

To showcase the impact of the AdAM multiplier on reliability and performance trade-offs, the fault injection simulations are performed on a variety of 8-bit convolutional neural network architectures including AlexNet, DenseNet, Inception, VGG-11, VGG-13, VGG-19, ResNet-18, ResNet-34, and ResNet-50 with four different configurations: unprotected exact multipliers (Unp-Exact), unprotected approximate multipliers (Unp-AxM), protected exact multipliers with TMR (Pro-TMR), and protected approximate multiplier with AdAM (Pro-AdAM). The DNN reliability is evaluated by comparing

Authorized licensed use limited to: Tallinn University of Technology. Downloaded on May 19,2026 at 09:12:58 UTC from IEEE Xplore. Restrictions apply.

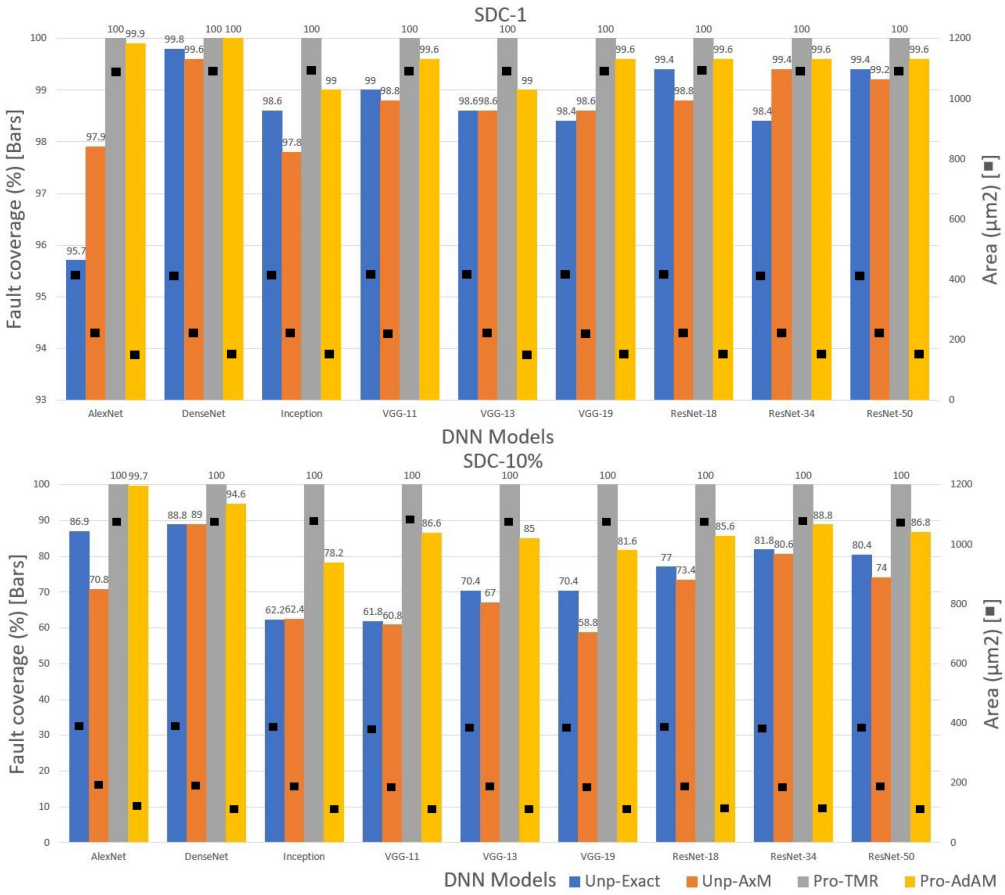


Fig. 7. Hardware efficiency (area) and fault resilience (fault coverage considering SDC-1 and SDC-10%) trade-offs in different benchmarks: AlexNet (MNIST), DenseNet (CIFAR), Inception (CIFAR), ResNet-18 (CIFAR), ResNet-34 (CIFAR), ResNet-50 (CIFAR), VGG-11 (CIFAR), VGG-13 (CIFAR), VGG-16 (CIFAR). Unp-Exact: unprotected exact multiplier, Unp-AxM: unprotected approximate multiplier, Pro-TMR: exact multiplier protected with TMR, Pro-AdAM: proposed multiplier.

the output probability vector of the golden run (i.e., the DNN that behaves as expected, without faults) and the faulty run (i.e., the DNN that includes the fault). The output probability vector represents the list of possible classes with their associated confidence scores predicted by the DNN. The SDC rate is defined as the proportion of faults that caused misclassification in comparison with the golden model. Since in DNNs, there is often not a single correct output, but a list of ranked outputs, each with a confidence score [43], we need to use new criteria to determine what constitutes an SDC for a DNN application. Therefore, we consider four types of SDCs as follows:

- *SDC-1:* The top-ranked element predicted by the DNN is different from that predicted by its fault-free execution. This is critical to assess because the top-ranked element is what is typically used for downstream processing. This criterion directly compares the class with the highest confidence score in the output probability vector of the faulty run to that of the golden run. If the top-ranked class differs between the two runs, this is counted as an SDC-1.
- *SDC-5%:* More than 5% of variation in the top-ranked output confidence score compared to the golden model. This criterion focuses on the confidence score of the top-ranked class in the output probability vector. If the confidence score deviates by more than 5% from the golden run, it is counted as an SDC-5%.
- *SDC-10%:* More than 10% of variation in the top-ranked output confidence score compared to the golden model. Similarly, this criterion looks for deviations in the confidence score of the top-ranked class by more than 10% from the golden run.
- *SDC-20%:* More than 20% of variation in the top-ranked output confidence score compared to the golden model. This criterion identifies faults where the confidence score of the top-ranked class in the output probability vector deviates by more than 20% from the golden run.

Authorized licensed use limited to: Tallinn University of Technology. Downloaded on May 19,2026 at 09:12:58 UTC from IEEE Xplore. Restrictions apply.

TABLE VII
FAULT COVERAGE (SDC-5%) IN DIFFERENT BENCHMARKS

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

TABLE VIII
FAULT COVERAGE (SDC-20%) IN DIFFERENT BENCHMARKS

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

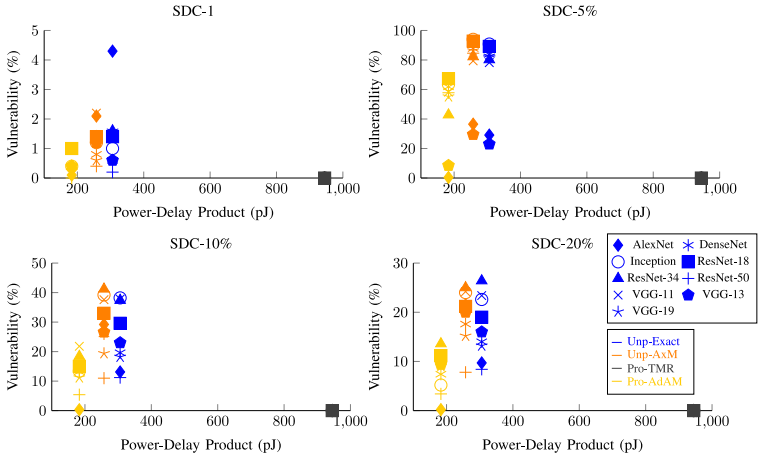


Fig. 8. PDP and vulnerability tradeoffs (considering different SDCs) in different benchmarks: AlexNet, DenseNet, Inception, ResNet-18, ResNet-34, ResNet-50, VGG-11, VGG-13, VGG-16. Unp-exact: unprotected exact multiplier, Unp-AxM: unprotected approximated multiplier, Pro-TMR: exact multiplier protected with TMR, Pro-AdAM: proposed multiplier.

All four criteria compare values of the output probability vector of the faulty run to that of the golden run. SDC-1 assesses the correctness of the top-ranked class, while SDC-5%, SDC-10%, and SDC-20% evaluate how much the confidence score of the top-ranked class changes in the faulty run relative to the golden run.

To quantify the effectiveness of each method in mitigating faults, *fault coverage* metric is used. It is defined as (100 − SDC value) representing the percentage of faults that are correctly handled (i.e., do not result in silent data corruption) by each method.

Fig. 7 demonstrates the fault tolerance comparison and reliability improvement (for SDC-1 and SDC-10% as two examples) of different networks by using the protected approximate multiplier proposed in this work compared to the unprotected exact and approximated networks, and protected networks using TMR. As illustrated, TMR has 100% of protection, but it also requires about 200% of area overhead. Different from TMR, in our technique we introduce a high-reliability improvement without introducing hardware overhead. The results for reliability improvement considering SDC-5% and SDC-20% are reported in Tables VII and VIII. Since the main objective of the proposed multiplier is to have the best trade-off in PDP and vulnerability, Fig. 8 illustrate these comparisons based on different vulnerability metrics (SDC-10, SDC-5%, SDC-10% and SDC-20%). In these charts, the closer to the origins (0,0), the higher the cost-efficiency of the fault tolerance, i.e., less vulnerability and less PDP. As shown, TMR is a less efficient solution for edge AI applications because of its high PDP, while the proposed method (AdAM) is the closest to the origin.

V. CONCLUSION

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

Authorized licensed use limited to: Tallinn University of Technology. Downloaded on May 19,2026 at 09:12:58 UTC from IEEE Xplore. Restrictions apply.

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

REFERENCES

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

[2] B. Rokh, A. Azarpeyvand, and A. Khanteymoori, "A comprehensive survey on model quantization for deep neural networks in image classification," *ACM Trans. Intell. Syst. Technol.*, vol. 14, no. 6, pp. 1–50, 2023.

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

[4] M. Taheri et al., "DeepAxe: A framework for exploration of approximation and reliability trade-offs in DNN accelerators," in *Proc. 24th Int. Symp. Qual. Electron. Design (ISQED)*, 2023, pp. 1–8.

[5] M. H. Ahmadilivani et al., "Special session: Approximation and fault resiliency of DNN accelerators," in *Proc. IEEE 41st VLSI Test Symp. (VTS)*, 2023, pp. 1–10.

[6] P. Nayak, D. Zhang, and S. Chai, "Bit efficient quantization for deep neural networks," in *Proc. 5th Workshop Energy Effic. Mach. Learn. Cogn. Comput.-NeurIPS Ed. (EMC2-NIPS)*, 2019, pp. 52–56.

[7] M. Taheri et al., "Exploration of activation fault reliability in quantized systolic array-based DNN accelerators," 2024, *arXiv:2401.09509*.

[8] M. Taheri et al., "AdAM: Adaptive fault-tolerant approximate multiplier for edge dnn accelerators," 2024. *arXiv:2403.02936*.

[9] M. Andjelkovic, O. Schrape, A. Breitenreiter, and M. Krstic, "Set and SEU hardened clock gating cell," in *Proc. 38th Conf. Design Circuits Integr. Syst. (DCIS)*, 2023, pp. 1–6.

[10] P. Rech, "Artificial neural networks for space and safety-critical applications: Reliability issues and potential solutions," *IEEE Trans. Nucl. Sci.*, vol. 71, no. 4, pp. 377–404, Apr. 2024.

[11] S. Narayanamoorthy, H. A. Moghaddam, Z. Liu, T. Park, and N. S. Kim, "Energy-efficient approximate multiplication for digital signal processing and classification applications," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 23, no. 6, pp. 1180–1184, Jul. 2015.

[12] S. Hashemi, R. I. Bahar, and S. Reda, "DRUM: A dynamic range unbiased multiplier for approximate applications," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, 2015, pp. 418–425.

[13] M. Taheri, H. Zandevakili, and A. Mahani, "A high-performance memristor-based smith-waterman dna sequence alignment using FPNI structure," *J. Appl. Res. Electr. Eng.*, vol. 1, no. 1, pp. 59–68, 2022.

[14] S. Vahdat, M. Kamal, A. Afzali-Kusha, and M. Pedram, "TOSAM: An energy-efficient truncation-and rounding-based scalable approximate multiplier," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 27, no. 5, pp. 1161–1173, May 2019.

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

[16] M. H. Haider and S.-B. Ko, "Booth encoding based energy efficient multipliers for deep learning systems," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 70, no. 6, pp. 2241–2245, Jun. 2023.

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

[18] J. N. Mitchell, "Computer multiplication and division using binary logarithms," *IRE Tran. Electron. Comput.*, vol. 11, no. 4, pp. 512–517, Aug. 1962.

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

[20] H. Jiang, F. J. H. Santiago, H. Mo, L. Liu, and J. Han, "Approximate arithmetic circuits: A survey, characterization, and recent applications," *Proc. IEEE*, vol. 108, no. 12, pp. 2108–2135, Dec. 2020.

[21] G. Armeniakos, G. Zervakis, D. Soudris, and J. Henkel, "Hardware approximate techniques for deep neural network accelerators: A survey," *ACM Comput. Surveys*, vol. 55, no. 4, pp. 1–36, 2022.

[22] 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 *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, 2017, pp. 1–12.

[23] G. Rodrigues, F. L. Kastensmidt, and A. Bosio, "Survey on approximate computing and its intrinsic fault tolerance," *Electronics*, vol. 9, no. 4, p. 557, 2020.

[24] A. Bosio et al., *Design, Verification, Test, and In-Field Implications of Approximate Digital Integrated Circuits*. Cham, Switzerland: Springer Int. Publ., 2022, pp. 349–385.

[25] M. Traiola, B. Deveautour, A. Bosio, P. Girard, and A. Virazel, *Test and Reliability of Approximate Hardware. Approximate Computing*. New York, NY, USA: Springer Int. Publ., 2022, pp. 233–266.

[26] H. Jiang, F. J. H. Santiago, H. Mo, L. Liu, and J. Han, "Approximate arithmetic circuits: A survey, characterization, and recent applications," *Proc. IEEE*, vol. 108, no. 12, pp. 2108–2135, Dec. 2020.

[27] A. Siddique and K. A. Hoque, "Exposing reliability degradation and mitigation in approximate DNNs under permanent faults," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 31, no. 4, pp. 555–566, Apr. 2023.

[28] S. Saeedi, A. Carpegna, A. Savino, and S. Di Carlo, "Prediction of the impact of approximate computing on spiking neural networks via interval arithmetic," in *Proc. IEEE 23rd Latin Amer. Test Symp. (LATS)*, 2022, pp. 1–6.

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

[30] V. Mrazek, R. Hrbacek, Z. Vasicek, and L. Sekanina, "Evoapprox8b: Library of approximate adders and multipliers for circuit design and benchmarking of approximation methods," in *Proc. Design, Autom. Test Eur. Conf. Exhibit. (DATE)*, 2017, pp. 258–261.

[31] A. Siddique, K. Basu, and K. A. Hoque, "Exploring fault-energy trade-offs in approximate dnn hardware accelerators," in *Proc. 22nd Int. Symp. Qual. Electron. Design (ISQED)*, 2021, pp. 343–348.

[32] S. Mittal, "A survey on modeling and improving reliability of DNN algorithms and accelerators," *J. Syst. Archit.*, vol. 104, Mar. 2020, Art. no. 101689.

[33] G. Ammes, G. B. Manske, P. F. Butzen, A. I. Reis, and R. P. Ribas, "ATMR design by construction based on two-level ALS," in *Proc. 36th SBC/SBMicro/IEEE/ACM Symp. Integr. Circuits Syst. Design (SBCCI)*, 2023, pp. 1–6.

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

[35] B. Deveautour, M. Traiola, A. Virazel, and P. Girard, "QAMR: An approximation-based fully reliable tmr alternative for area overhead reduction," in *Proc. IEEE Eur. Test Symp. (ETS)*, 2020, pp. 1–6.

[36] M. Traiola, J. Echavarria, A. Bosio, J. Teich, and I. O'Connor, "Design space exploration of approximation-based quadruple modular redundancy circuits," in *Proc. IEEE/ACM Int. Conf. Comput. Aided Design (ICCAD)*, 2021, pp. 1–9.

[37] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra, "Weight uncertainty in neural network," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 1613–1622.

[38] D. Danopoulos, G. Zervakis, K. Siozios, D. Soudris, and J. Henkel, "Adapt: Fast emulation of approximate DNN accelerators in PyTorch," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 42, no. 6, pp. 2074–2078, Jun. 2023.

[39] R. Leveugle, A. Calvez, P. Maistri, and P. Vanhauwaert, "Statistical fault injection: Quantified error and confidence," in *Proc. Design, Autom. Test Eur. Conf. Exhibit.*, 2009, pp. 502–506.

[40] R. Zendegani, M. Kamal, M. Bahadori, A. Afzali-Kusha, and M. Pedram, "RoBa multiplier: A rounding-based approximate multiplier for high-speed yet energy-efficient digital signal processing," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 25, no. 2, pp. 393–401, Feb. 2017.

[41] S. Vahdat, M. Kamal, A. Afzali-Kusha, and M. Pedram, "LETAM: A low energy truncation-based approximate multiplier," *Comput. Electr. Eng.*, vol. 63, pp. 1–17, Oct. 2017.

[42] A. Ruospo, E. Sanchez, M. Traiola, I. O'Connor, and A. Bosio, "Investigating data representation for efficient and reliable convolutional neural networks," *Microprocess. Microsyst.*, vol. 86, Oct. 2021, Art. no. 104318.

[43] G. Li et al., "Understanding error propagation in deep learning neural network (DNN) accelerators and applications," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, 2017, pp. 1–12.

Authorized licensed use limited to: Tallinn University of Technology. Downloaded on May 19,2026 at 09:12:58 UTC from IEEE Xplore. Restrictions apply.

Appendix 4

IV

N. Cherezova, S. Pappalardo, A. Ruospo, B. Deveautour, L. Fezza, A. Jutman, E. Sanchez, A. Bosio, M. S. Reorda, and M. Jenihhin. Special session: Reliability assessment of DNN models and inference on systolic arrays. In *IEEE VLSI Test Symposium (VTS)*, 2026

Special Session: Reliability Assessment of DNN Models and Inference on Systolic Arrays

Natalia Cherezova[1], Salvatore Pappalardo[2], Annachiara Ruospo[3],
Bastien Deveautour[2], Lorenzo Fezza[3], Artur Jutman[1], Ernesto Sanchez[3],
Alberto Bosio[2], Matteo Sonza Reorda[3], Maksim Jenihhin[1]
[1]Tallinn University of Technology, Estonia
[2]Ecole Centrale de Lyon, France
[3]Politecnico di Torino, Italy

2026 IEEE 44th VLSI Test Symposium (VTS) | 979-8-3315-6337-0/26/$31.00 ©2026 IEEE | DOI: 10.1109/VTS69484.2026.11563366

***Abstract*—Deep Neural Networks (DNNs) are increasingly deployed in safety-critical and high-performance computing systems, often relying on specialized hardware accelerators such as systolic arrays to achieve high throughput and energy efficiency. However, the growing complexity of DNN models and the continued scaling of semiconductor technologies expose these systems to reliability challenges caused by transient and permanent hardware faults. Assessing the reliability of DNN inference on systolic arrays is essential, yet existing approaches face significant challenges: microarchitecture-level fault injection is prohibitively slow for modern DNNs, while purely software-level methods may not accurately capture hardware-specific fault propagation. This special session presents three complementary contributions addressing these challenges. First, *Systolic Array Fault-Propagation Analysis (SAFPA)* is a method that enables reliability assessment orders of magnitude faster than microarchitecture-level simulation, with support for multiple systolic array dataflows. Second, *an error-driven iterative statistical fault injection methodology* is presented that minimizes the number of required fault injections while preserving statistical validity, which is particularly advantageous for large fault populations. Third, *a hardware-aware simulation-based fault injection framework (FLAUERS)* that bridges the gap between algorithm-level and RTL-level analysis by leveraging the mathematical structure of systolic arrays. Together, these contributions advance the state of the art in efficient and accurate reliability assessment of DNN accelerators.**

***Index Terms*—Deep neural networks, hardware accelerators, systolic arrays, reliability assessment, fault injection, fault propagation, safety-critical systems, edge AI.**

I. Introduction

Deep Neural Networks (DNNs) have achieved remarkable accuracy in tasks such as image classification, object detection, and natural language processing, driving their adoption in safety-critical domains including autonomous driving, robotics, and medical diagnostics [1]. In these contexts, the reliability of both the DNN model and its underlying hardware accelerator is not just desirable but mandated by functional safety standards such as ISO 26262 [2] and ISO PAS 8800 [3]. A silent data corruption during inference, caused, for example, by a transient particle strike or a permanent manufacturing defect, can alter a classification outcome with potentially life-threatening consequences.

Systolic arrays (SAs) have emerged as the predominant architecture for DNN acceleration, employed in commercial platforms such as Google's Tensor Processing Unit (TPU) [4] and numerous edge-AI accelerators. Their regular, tiled structure of Processing Elements (PEs), each performing Multiply-Accumulate (MAC) operations, enables high throughput and energy efficiency [5]. However, this same regular structure means that a single fault in a PE register can propagate through the dataflow and corrupt multiple output values, with the specific error pattern depending on the SA dataflow (i.e., weight-stationary, input-stationary, or output-stationary) and the type of affected register [6], [7].

Reliability assessment of DNN inference on SAs is therefore critical, yet it presents a significant computational challenge. The fault space is huge because modern DNNs contain millions to billions of parameters, and each bit in every register of the SA establishes a potential fault location. Exhaustive fault injection at the Register Transfer Level (RTL) is a very accurate approach but is prohibitively expensive, requiring hours or even days per single inference with fault injection [8], [9]. On the other hand, hardware-agnostic software-level fault injection is fast but fails to capture the specific fault propagation patterns that arise from the SA microarchitecture [10]. The accuracy-efficiency trade-off has motivated research seeking intermediate solutions that are both hardware-aware and computationally feasible [11]–[13].

Recent works have explored several directions to address this challenge. Microarchitecture-level simulators such as SacaFI [12] model the SA behavior in detail but remain too slow for large-scale campaigns. Hybrid approaches like SAFFIRA [11] use Uniform Recurrent Equations to model the SA core in software, achieving significant speedups over RTL simulation. Analytical methods based on fault propagation analysis [6], [14] avoid full simulation altogether by calculating the effect of faults directly on layer outputs. In the statistical domain, techniques for reducing the number of required fault injections through sampling have been studied extensively in the general dependability literature [15] and are now being adapted to the specific characteristics of DNNs [16], [17]. Despite this progress, there remains an open challenge of scalability to large models with tight error margins, as well as the availability of accessible open-source tools for the research community.

This special session brings together three research groups to present complementary solutions that collectively advance the state of the art in reliability assessment of DNN inference on systolic arrays:

979-8-3315-6337-0/26/$31.00 ©2026 IEEE

Authorized licensed use limited to: Tallinn University of Technology. Downloaded on November 10,2026 at 15:22:09 UTC from IEEE Xplore. Restrictions apply.

- Section II presents *Systolic Array Fault-Propagation Analysis (SAFPA)*, an analytical method that replaces microarchitecture-level simulation with closed-form equations to calculate fault-induced errors directly in the DNN layer output. It supports all three major SA dataflows, handles both transient and permanent faults, and achieves speedups of several orders of magnitude.
- Section III presents an *error-driven iterative statistical fault injection* methodology alongside a conservative method based on pre-estimated fault criticality, reducing the number of required injections by up to 50% compared to the traditional one-step approach.
- Section IV presents *FLAUERS*, a hardware-aware simulation-based fault injection framework that exploits the mathematical structure of Regular Iterative Algorithms to model SA-specific fault propagation within a PyTorch workflow, achieving approximately 10× speedup over RTL simulations.

Together, these three contributions cover the reliability assessment pipeline from analytical modeling through statistical sampling to simulation-based fault injection, providing the research community with a comprehensive and complementary set of tools and methodologies for ensuring the dependable deployment of DNNs on systolic array accelerators.

II. SAFPA: Fast analytical reliability assessment for DNN Inference on Systolic Arrays

This section presents an analytical fault-propagation-based method SAFPA that enables fast evaluation and supports multiple systolic array dataflows.

A variant of this method for output-stationary systolic array was applied for a reliability assessment step in [6].

A. Background

Systolic array architecture. Systolic Array (SA) is a 2D array of Processing Elements (PEs). Each PE consists of a Multiply-Accumulate (MAC) unit and registers to hold the intermediate values. The structure of PEs and the movement of data through the array are defined by the dataflow, such as weight-stationary (WS), input-stationary (IS), and output-stationary (OS) [5].

Convolution on systolic array. The convolution operation is mapped to the SA by transforming it to matrix multiplication using the im2col algorithm [18]. An input tensor is transformed into a $(H_{out} \cdot W_{out}) \times (H_k \cdot W_k \cdot C_{in})$ matrix and weights into a $(H_k \cdot W_k \cdot C_{in}) \times C_{out}$ matrix, where H_{out} and W_{out} are the height and width of the output tensor, H_k and W_k are the height and width of a kernel, C_{in} and C_{out} are the number of channels in the input and output tensor.

If the input matrices exceed the size of the SA N, matrix multiplication is performed in tiles. For the OS SA, the number of tiles is

$$T_a = \left\lceil \frac{H_{out} \cdot W_{out}}{N} \right\rceil,\ T_w = \left\lceil \frac{C_{out}}{N} \right\rceil, \tag{1}$$

where T_a and T_w are the number of tiles in activations and weights. For the WS SA, the number of tiles is

$$T_a = \left\lceil \frac{H_k \cdot W_k \cdot C_{in}}{N} \right\rceil,\ T_w = \left\lceil \frac{C_{out}}{N} \right\rceil \cdot T_a = T_w^C \cdot T_a\,. \tag{2}$$

For the IS SA, the number of tiles is

$$T_a = \left\lceil \frac{H_{out} \cdot W_{out}}{N} \right\rceil \cdot T_w = T_a^S \cdot T_w\,,\ T_w = \left\lceil \frac{H_k \cdot W_k \cdot C_{in}}{N} \right\rceil.$$

B. Methodology

This subsection presents the proposed reliability assessment method for SAs (SAFPA). It combines fault injection with fault propagation analysis, which is used to calculate the resulting error in the layer output tensor. This enables a notable speed-up of the reliability assessment of DNN inference on an SA. The method allows analyzing both transient and permanent faults.

Fault propagation in systolic arrays. Faults in PE registers are considered for the analysis. Due to the regular structure of an SA and data movement, faults in different components produce distinct error patterns in the output tensor. Fig. 1 presents error patterns for the convolution operation on different dataflows. Patterns are derived analytically and experimentally, using microarchitecture-level simulation of the SA, following examples from previous works for similar cases [7], [14].

Transient faults:

Input register (IREG): For both OS and WS SA, a bit flip in an input register (IREG) results in the propagation of faulty activation through the whole row, affecting one value in several output channels. In the IS SA, the fault persists until a new activation tile is loaded, affecting one value in all output channels. This ends up in the *bullet pattern* for the error.

Weight register (WREG): A bit flip in WREG, however, produces different error patterns. In WS, fault persists until new weights are loaded, resulting in the *partial- or whole-channel pattern*. In OS, faulty weight propagates through the whole column, affecting several output values in one channel. In IS, fault weight propagates through the whole row, still affecting several output values in one channel. This ends up in the *line pattern*.

Output register (OREG): A bit flip in OREG affects one value in one output channel, resulting in a *point pattern* for all dataflows.

Permanent faults. For WS SA, a stuck-at fault in all cases results in several whole-channel patterns. For OS, a fault in IREG results in several bullet patterns; a fault in WREG results in several line patterns; and a fault in OREG results in several whole-channel patterns, since the results are offloaded by propagating from top to bottom. For IS, faults in IREG and OREG result in several bullet patterns; a fault in WREG results in a line bullet pattern.

Fault propagation analysis for transient faults. Each fault is defined by parameters given in Table I.

An output of the convolutional layer can be expressed as

$$y_{(u,v)} = \sum_{c}^{C_{in}} \sum_{i}^{H_k} \sum_{j}^{W_k} w_{(c,i,j)} \cdot x_{(c,u+i,v+j)} + b\,, \tag{3}$$

Authorized licensed use limited to: Tallinn University of Technology. Downloaded on November 10,2026 at 15:22:09 UTC from IEEE Xplore. Restrictions apply.

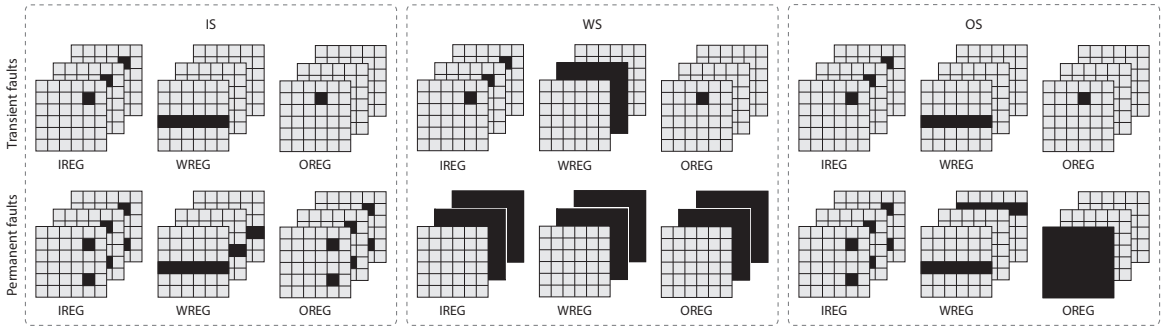


Fig. 1: Error patterns in the output tensor for different systolic array dataflows

TABLE I: Transient fault parameters

Symbol	Description
f_{type}	Type of fault (IREG, WREG, OREG)
ts	Cycle of a systolic array execution
t_w	Weight tile
t_a	Activation tile
p_{row}	Row of the PE
p_{col}	Column of the PE
β_f	Bit index

where $y_{(u,v)}$ is the output value with index (u, v), $w_{(c,i,j)}$ is an individual weight, $x_{(c,u+i,v+j)}$ is an activation (input), and b is a bias.

A bit flip in the numerical value can be expressed as an addition of an error term ε. For an integer, the error term is $\pm 2^{flipped_bit}$. For a floating point, the error term is different for the sign bit, exponent, and mantissa bits and is described in [19]. For this study, a 32-bit floating-point arithmetic is considered.

A fault in a register holding activation, weight, or partial sum results in adding an error e to the output of the convolutional layer. This error e can be calculated for each type of fault from the parameters given in Table I. Equations for calculating errors for the OS SA are given in [6]. Here, new equations for the WS and IS dataflows are presented.

Transient faults in WS dataflow.

Fault in IREG. The fault corrupts an activation, hence an error added to the output of the convolutional layer equals

$$e_{ireg} = w_{(c_f, i_f, j_f)} \cdot \varepsilon_x \,, \tag{4}$$

where $w_{(c_f,i_f,j_f)}$ is the weight multiplied with the faulty activation. This weight is found by its position (c_f, i_f, j_f):

$$c_f = \left\lceil \frac{(t_a - 1) \cdot N + p_{row}}{H_k \cdot W_k} \right\rceil \,, \tag{5}$$

$$k_f = ((t_a - 1) \cdot N + p_{row}) \bmod (H_k \cdot W_k) \,, \tag{6}$$

$$i_f = \left\lfloor \frac{k_f}{W_k} \right\rfloor \,, \quad j_f = k_f \bmod W_k \,. \tag{7}$$

Fault results in the bullet error pattern. Therefore, the following parameters define affected output values: output channels $c_{out,f}$ and position in the feature map (u_f, v_f). Affected output channels are calculated as follows:

$$c_{out,f} = \left[c^{start}_{out,f}, c^{end}_{out,f} \right] \,, \tag{8}$$

$$c^{start}_{out,f} = (t_w - 1) \cdot N + p_{col} + 1 \,, \tag{9}$$

$$c^{end}_{out,f} = \begin{cases} t_w \cdot N & \text{if} \quad C_{out} \geqslant t_w \cdot N \\ C_{out} & \text{if} \quad C_{out} < t_w \cdot N \,, \end{cases} \tag{10}$$

where $c_{out,f}$ is a range of affected output channels, starting with $c^{start}_{out,f}$ and ending with $c^{end}_{out,f}$.

The position (u_f, v_f) is found in the following way:

$$\square_f = ts - p_{col} - p_{row} \,, \tag{11}$$

$$u_f = \left\lfloor \frac{\square_f}{W_{out}} \right\rfloor \,, \quad v_f = \square_f \bmod W_{out} \,, \tag{12}$$

where $\square_f$ is the affected sliding window. An error e_{ireg} calculated using Eq. (4) is added to the affected output values.

Fault in WREG. An error added to the output of the convolutional layer in this case equals $e_{wreg} = x_{(c_f,i_f,j_f)} \cdot \varepsilon_w$.

Fault results in the partial or whole-channel error pattern, therefore, the following parameters define affected output values: output channel $c_{out,f}$ and a range of positions starting with $(u^{start}_f, v^{start}_f)$ and ending with (u^{end}_f, v^{end}_f). However, (u^{end}_f, v^{end}_f) is constant; it is the last pixel in the output. The affected output channel is calculated as follows:

$$c_{out,f} = (t_w - 1) \cdot N + p_{col} \,. \tag{13}$$

Positions are calculated from the affected sliding windows following Eq. (12):

$$\square^{start}_f = (t_a - 1) \cdot N + p_{row} \,, \tag{14}$$

$$\square^{end}_f = W_{out} \cdot H_{out} \,. \tag{15}$$

Fault in OREG. Since those faults result in a point error pattern, an error $e_{oreg} = \varepsilon_y$ is added to a single pixel in the output feature map. The output channel $c_{out,f}$ is found using Eq. (13), and the position is found using Eq. (12).

Transient faults in IS dataflow.

Authorized licensed use limited to: Tallinn University of Technology. Downloaded on November 10,2026 at 15:22:09 UTC from IEEE Xplore. Restrictions apply.

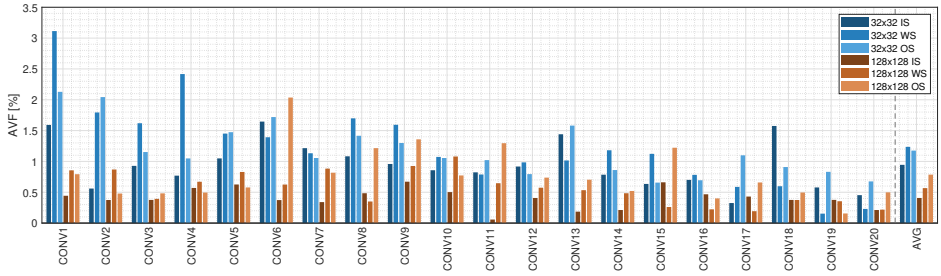


Fig. 2: Reliability assessment of ResNet-18 on different SAs considering transient faults

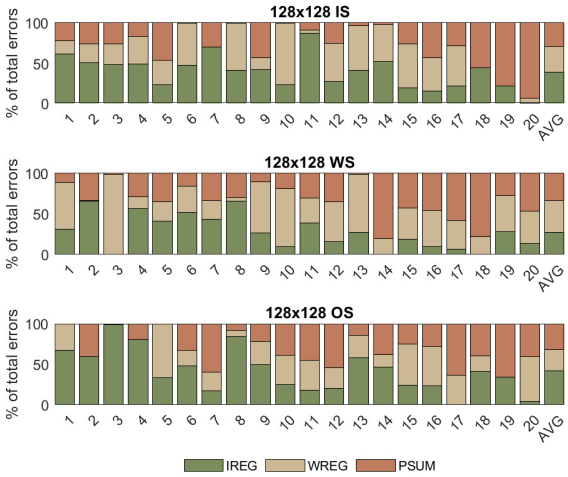


Fig. 3: Layer-wise fault distribution for ResNet-18

Fault in IREG. The weight $w_{(c_f,i_f,j_f)}$ multiplied with the faulty activation is found by its position (c_f, i_f, j_f):

$$c_f = \left\lceil \frac{(t_w - 1) \cdot N + p_{row} + 1}{H_k \cdot W_k} \right\rceil , \tag{16}$$

$$k_f = ((t_w - 1) \cdot N + p_{row}) \bmod (H_k \cdot W_k) , \tag{17}$$

$$i_f = \left\lfloor \frac{k_f}{W_k} \right\rfloor , \quad j_f = k_f \bmod W_k . \tag{18}$$

Affected output channels are calculated as:

$$c_{out,f}^{start} = ts - p_{col} - p_{row} , \quad c_{out,f}^{end} = C_{out} . \tag{19}$$

The position of the affected output value (u_f, v_f) is found from the sliding window using Eq. (12):

$$\square_f = (t_a - 1) \cdot N + p_{col} . \tag{20}$$

Fault in WREG. Fault results in the line error pattern; therefore, the following parameters define affected output values: output channel $c_{out,f}$ and a range of positions starting with $(u_f^{start}, v_f^{start})$ and ending with (u_f^{end}, v_f^{end}). The affected output channel is calculated as follows:

$$c_{out,f} = ts - p_{col} - p_{row} . \tag{21}$$

Positions are calculated from the affected sliding windows following Eq. (12):

$$\square_f^{start} = (t_a - 1) \cdot N + p_{col} \tag{22}$$

$$\square_f^{end} = \begin{cases} t_a \cdot N & \text{if} \quad W_{out} \cdot H_{out} \geqslant t_a \cdot N \\ W_{out} \cdot H_{out} & \text{if} \quad W_{out} \cdot H_{out} < t_a \cdot N \end{cases} \tag{23}$$

Fault in OREG. Fault results in a single-point error pattern as well. An error $e_{oreg} = \varepsilon_y$ is added to a single pixel in the output feature map. The output channel $c_{out,f}$ is found using Eq. (21), and the position is found using Eq. (20) and (12).

Fault propagation analysis for permanent faults. Permanent faults are present throughout execution and are defined by four parameters: fault type, PE row and column, and bit index.

Permanent faults result in several error patterns, one for each step. Therefore, the affected output values are defined by sets of parameters. E.g., for permanent faults in the WS IREG, the affected output values are defined by a set of output channels

$$\left\{ (c_{out,f}^{start}, c_{out,f}^{end})_i \mid i \in \{1, ..., T_w\} \right\} \tag{24}$$

and a set of positions of the affected activations

$$\left\{ (c_f, u_f + i_f, v_f + j_f)_i \mid i \in \{1, ..., T_a\} \right\} . \tag{25}$$

A set of output channels is found by substituting t_w with an iterator in Eq. (9) and (10), which gives a pair $(c_{out,f}^{start}, c_{out,f}^{end})$ for every weight tile.

A set of positions of the affected activations is found by, first, substituting t_a with an iterator in Eq. (5) and (6) to find the affected weight positions, which gives a position for every input tile. They are used to calculate the positions of the affected activations for each sliding window, since all pixels in the output channel are affected.

C. Results

Implementation. SAFPA is implemented in Python, and PyTorch is used to run the inference of DNNs, taking advantage of GPU speedup. To further speed up permanent fault injection, error calculations are implemented in C. The fault injection module gets the output of the targeted convolutional layer and modifies only affected output values according to the equations. The inference continues with the erroneous values. SAFPA will be available open-source at https://github.com/ntlchr/safpa.

Evaluation. SAFPA is compared with the representative fault injection framework that models SA microarchitecture, Saca-FI [12]. Table II presents the comparison of two methods. Runtime

Authorized licensed use limited to: Tallinn University of Technology. Downloaded on November 10,2026 at 15:22:09 UTC from IEEE Xplore. Restrictions apply.

TABLE II: Comparison of reliability assessment methods

Work	Fault model	Output	Dataflow	Runtime*
Saca-FI	Transient and permanent faults in registers	AVF	IS, WS, OS	37 s / 88 s
SAFPA (ours)	Transient and permanent faults in registers	AVF	IS, WS, OS	0.0006 s / 0.001 s

* Transient fault injection / permanent fault injection.

denotes the processing time of one input with fault injection for transient and permanent faults. For runtime comparison, the selected model is VGG-16 evaluated on ImageNet (Saca-FI supports only three DNNs, the biggest one was selected), SA size is 256×256, fault is injected in the first convolutional layer. Experiments were performed on an NVIDIA A100 GPU. Both methods give AVF (Architecture Vulnerability Factor) as an output, the probability that a fault in the hardware structure causes an application output error [20]. While the functionality of the methods is similar, SAFPA is several orders of magnitude faster than the microarchitecture-level fault injection.

Fault injection. ResNet-18 trained on ImageNet was selected for an example of reliability evaluation. Fig. 2 presents the results of the reliability assessment of ResNet-18 considering different dataflows and SA sizes (32×32 and 128×128) and transient faults. Results show that for the selected model, for the smaller SA size, WS dataflow on average has higher vulnerability, while OS dataflow has higher vulnerability on bigger SA size. Fig. 3 presents error rates of PE registers for each convolutional layer (128×128 SA). It can be seen that, on average, for IS and WS SAs, registers holding stationary values cause a higher percentage of output errors. For OS SA, IREG is the most vulnerable register.

III. Iterative and Conservative Statistical Fault Injections for DNN reliability assessments

Software-based fault injection (FI) methods are commonly employed to run a *preliminary* vulnerability evaluation of the AI system prior to deploying the DNN model on the target hardware accelerator. These experiments are of paramount importance, as they provide insight into the average failure rate (FR), or the most critical units or layers of a DNN. Based on these findings, targeted mitigation strategies can then be developed.

In an FI experiment, the system under assessment (e.g., the DNN) is exposed to a controlled set of faults, and its response is monitored. Since not all injected faults lead to a system failure, the objective of an FI campaign is to quantify the fraction of activated faults that result in a system failure (e.g., a DNN wrong prediction). This metric is typically quantified as the Failure Rate (FR), defined as the ratio of the total number of observed failures to the total number of incorrect inferences.

However, the cost of FI grows proportionally with the growing complexity of modern AI systems, often leading to unacceptably long FI experiments. Nowadays, DNNs are made of billions or trillions of parameters: carrying out an exhaustive FI is impractical and would require years to complete. Although the complexity of DNNs used in safety-critical systems is typically smaller, the reliability assessment process is still extremely time-consuming. The same reasoning can be applied to hardware-level FIs: today's hardware technology is shrinking with the design of integrated circuits to a few nanometers, incorporating far more transistors in the same area. Simulating the entire fault list is becoming very complex, computationally demanding, and costly, often exceeding acceptable time-to-market constraints.

To tackle this challenge, Statistical Fault Injection (SFI) approaches have been introduced to reduce the cost of the fault simulation procedure while preserving the statistical validity of the results [15]–[17], [21]. In general, SFI experiments inject a reduced and representative number of faults to achieve an estimate of the target characteristic with a maximum margin of error and a specific confidence level.

This section introduces the latest methodologies developed to extract a statistically valid failure rate while minimizing the computational cost and the time required to run the AI reliability assessment. In particular, two state-of-the-art (SOTA) solutions are described: a conservative [15], [16] and an iterative SFI approach [17]. Their respective trade-offs are analyzed, along with a discussion on the scenarios in which each method is most convenient.

A. Background on SFI

Statistical inference is the branch of statistics concerned with generalizing a characteristic of a population by observing only a reduced sample [22]. A statistical population (N) represents the complete set of measurements for all units of interest [22]. Since analyzing the entire population is often impractical, it is usually necessary to observe a sample (n). In any statistical analysis, it is essential to account for the statistical error, defined as the difference between the true value and the estimated value. For a large sample size (n $\geq$ 30), this error can be considered a random variable approximating the standard normal distribution. It is possible to compute the maximum error of the estimate (e) as $e = z_{\frac{\alpha}{2}} * \frac{\sigma}{\sqrt{n}}$, where: σ represents the standard deviation of the studied population N; n is the sample size; $z_{\frac{\alpha}{2}}$ is a constant that depends on the desired confidence level. It will be detailed in the following.

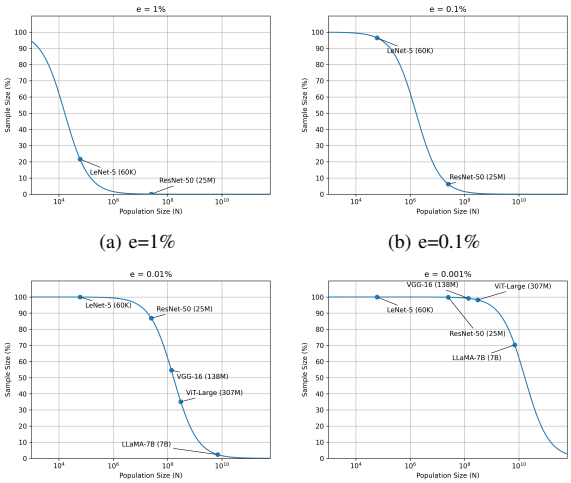
For a statistical inference, knowing the maximum error of the estimate e, and the measured statistic $\hat{x}$, the unknown value of the population we would like to measure (μ) falls within the confidence interval: $\hat{x} - e \leq \mu \leq \hat{x} + e$, with a $(1 - \alpha)$ confidence:

$$P\Big(\hat{x} - e \leq \mu \leq \hat{x} + e\Big) = \Big(1 - \alpha\Big) \tag{26}$$

A confidence level (t) represents the probability that a confidence interval contains the true population parameter. It says that, with a $(1-\alpha)$ confidence, the unknown parameter μ of the population will be covered by the confidence interval. Common choices for $(1-\alpha)$ are 0.95, 0.99, and 0.999; the corresponding values of $z_{\frac{\alpha}{2}}$ are 1.96, 2.58, and 3.09, respectively.

Therefore, to estimate the unknown value μ of a population, fixing a maximum acceptable margin of error (e), the sample size (n) is computed as follows:

Authorized licensed use limited to: Tallinn University of Technology. Downloaded on November 10,2026 at 15:22:09 UTC from IEEE Xplore. Restrictions apply.



(a) e=1% (b) e=0.1%

(c) e=0.01% (d) e=0.001%

Fig. 4: Required sample size (n) percentage as a function of N (conservative SFI). Some SOTA AI models are shown for reference.

$$n = \frac{N}{1 + e^2 \cdot \frac{N-1}{t^2 \cdot p*(1-p)}} \tag{27}$$

In Eq. (27), N corresponds to the actual size of the population and p to the probability of success, i.e., the likelihood that the predefined "success" occurs. Since $p \in [0, 1]$, when its value is unknown, it is commonly set to 0.5. This assumption corresponds to equal probabilities of success and failure and represents a conservative choice, as it maximizes the required sample size.

In the scope of FI experiments, N denotes the total number of possible faults in the system, while the sample n represents the subset of randomly injected faults used to estimate population characteristics, with $n \ll N$. An FI campaign consists of n repeated trials, where a success occurs when an injected fault leads to a critical failure.

B. SFI Methodologies

In the traditional SFI approach, also referred to as the one-step or conservative method, the total sample size (n) is determined before executing the fault injection campaigns, based on a predefined margin of error (e) and confidence level (t). Once e and t are defined, n is computed as in (27). Assuming a probability of success (p) equal to 0.5 provides a conservative upper bound on the required sample size. It ensures that the target margin of error is not underestimated, provided that the underlying statistical assumptions are satisfied. When carrying out SFI techniques, two relationships hold:

- The sample size is maximized when $p = 0.5$.
- The smaller the margin of error e, the larger the required sample size n.

However, for dependability analysis, selecting a small margin of error is crucial. A 5% margin of error implies that the statistical estimate lies within $\pm 5\%$ of the true value. While this may be acceptable when the failure rate is moderate to high, it becomes inadequate for safety-critical systems, where failure rates are typically very low [23]. For instance, a $\pm 5\%$ margin applied to an estimate of 0.16% implies that the true (unknown) value may lie within the range $[0, 5.16\%]$. A smaller margin of error must be used in a safety-critical scenario. However, reducing the margin of error significantly increases the number of faults that must be simulated. For extremely small margins of error (e = 0.001%), the required sample size remains close to 100% of the population across a wide range of N. Even for large-scale models such as VGG-16 (138M) or ViT-Large (307M), nearly the entire fault list must be injected, as shown in Fig. 4. Moreover, for a 99% confidence and p=0.5, the sample size saturates at $\sim$ 16k, $\sim$ 1.6M and $\sim$ 166M for e=1%, 0.1%, 0.01%, respectively.

However, by assuming equal probability of success and failure (p=50%), conservative SFI techniques inject more than the minimal number of injected faults [23].

Two alternatives have been explored:

1) Pre-estimation of p [16];
2) Iterative SFI solution [17], [24].

Pre-estimation of p: The first category falls under the one-step or conservative approach class. In [16], the Average Bit Flip Distance (ABFD) metric was proposed to pre-calculate the probability that a fault corrupting a precise bit position will produce a failure. These probabilities have been used in (27) to reduce the number of injected faults. As it will be shown, if the p estimation is accurate, the number of injected faults is minimized. Clearly, this approach is data-dependent and requires a preliminary analysis before running the fault injections. Assuming CNN weights are represented with I bits, a criticality metric $D_{\text{avg}}(i)$ is computed for each bit $i \in I$ as:

$$D_{\text{avg}}(i) = D_{0\to1}(i)\, f_0(i) + D_{1\to0}(i)\, f_1(i) \tag{28}$$

where $D_{0\to1}(i)$ and $D_{1\to0}(i)$ denote the average distance between golden and faulty weights caused by a bit-flip on bit i, and $f_0(i)$, $f_1(i)$ represent the frequency of bit i being 0 or 1 in the weight distribution.

$D_{\text{avg}}(i)$ captures the impact of flipping bit i. Since larger weight perturbations are more likely to cause mispredictions, we assume higher $D_{\text{avg}}(i)$ implies higher fault criticality. Therefore, the probability parameter $p(i)$ used for sample size computation is obtained by min-max normalizing $D_{\text{avg}}(i)$ in the range $[0, 0.5]$ (excluding outliers):

$$p(i) = \frac{(D_{\text{avg}}(i) - D_{\text{avg-min}})\, 0.5}{D_{\text{avg-max}} - D_{\text{avg-min}}} \tag{29}$$

where $D_{\text{avg-max}}$ and $D_{\text{avg-min}}$ are the maximum and minimum average distances across all bits. Once $p(i)$ is computed, the required sample size $n(i, l)$ is derived as in (27).

Authorized licensed use limited to: Tallinn University of Technology. Downloaded on November 10,2026 at 15:22:09 UTC from IEEE Xplore. Restrictions apply.

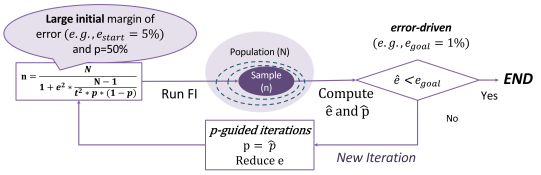


Fig. 5: Error-driven iterative SFI methodology

Iterative SFI solution: it is possible to benefit from initial statistical estimations available after a few experiments to further minimize the number of injected faults. It is not necessary to run any preliminary analysis (as in [16]). In the first iteration, a large margin of error is used (e.g., e=5%) and p=0.5. In this way, a first estimate of the failure rate of the system is obtained. Then, in subsequent iterations, the estimated failure rate is used as the updated p value, and the margin of error is gradually reduced until the target margin (e_{goal}) is reached. A schematic representation of the method is shown in Fig. 5. Starting from a large initial margin of error e_{start} and $p = 0.5$, the required sample size n is computed and fault injections are executed. The estimated failure probability $\hat{p}$ and margin of error $\hat{e}$ are then evaluated. If $\hat{e} < e_{\text{goal}}$, the process terminates; otherwise, p is updated to $\hat{p}$, the margin of error is reduced, and a new iteration is performed. At each new SFI iteration, the margin of error e is progressively reduced to reach the target e_{goal}. Instead of uniformly shrinking e, we propose an error-probability (E-P) curve that adapts the next margin of error according to the estimated failure rate $\hat{P}$. The objective is to approach e_{goal} rapidly for both non-critical and critical fault populations ($p \to 0$ or $p \to 1$), where fewer injections are required, while maintaining a balanced sample size. Specifically, when $\hat{P} \to 0$ or 1, the next margin of error tends to e_{goal}; when $\hat{P} \to 0.5$, it tends to $\hat{E}/3$. This behavior is modeled by a parabola passing through $(0, e_{\text{goal}})$, $(1, e_{\text{goal}})$, and $(0.5, \hat{E}/3)$, leading to:

$$e_{i+1} = \begin{cases} -k\hat{P}_i^2 + k\hat{P}_i + e_{\text{goal}}, & \text{if } \dfrac{\hat{E}_i}{3} > e_{\text{goal}} \\ e_{\text{goal}}, & \text{otherwise} \end{cases} \tag{30}$$

where

$$k = 4\left(\frac{\hat{E}_i}{3} - e_{\text{goal}}\right).$$

Compared to the state-of-the-art approach, which simply halves the margin of error at each iteration ($e_{i+1} = \hat{E}_i/2$), the proposed E-P curve avoids unnecessary fault injections and iterations. In particular, it reduces injections when $p \to 0$ or $p \to 1$, and limits excessive iterations when $p \to 0.5$.

C. Trade-off Analysis

Experimental analyses conducted on different DNN models (full results in [16], [17]) led us to the following insights:

TABLE III: Total number of FIs and iterations to achieve an estimate of SDC-1 faults in all layers with a 0.01% error margin.

SFI Methods	$e_{\text{goal}} = 0.01\%$		
	One-Step or Conservative		Iterative
	Data-unaware	Pre-estimaton of p	
ResNet20			
Total FIs	16,989,188	11,497,921	13,963,231
Max. Iterations	1	1	3.3
MobileNetV2			
Total FIs	131,913,403	70,383,855	65,977,468
Max. Iterations	1	1	4

TABLE IV: ResNet20: the larger the population of faults to be examined, the more advantageous it is to use an iterative approach to reduce the number of injections to be run.

LayerID	LayerType	Fault List Size (N)	Injected Faults [%] (e=0.01%, t=99%)	
			One-Step	Iterative
1	conv1	27,648	99.98	99.71
...	...	...	...	...
19	conv19	2,359,296	98.60	75.59
...	...	...	...	...

- When the parameter p is accurately pre-estimated, the number of injected faults is minimized. As shown in Table III, SFIs performed on ResNet20 illustrate that, with a proper pre-estimation of p, the total number of fault injections is lower compared to the iterative approach.
- When the acceptable error margin is small, the iterative fault injection approach becomes increasingly advantageous as the size of the fault list grows. The results in Table IV show that for small fault populations (e.g., conv1), the difference between the one-step and iterative approaches is negligible. However, as the fault list size increases (e.g., conv19), the iterative method significantly reduces the percentage of injected faults required to meet the target margin of error. This confirms that the iterative strategy becomes increasingly advantageous for large fault populations.

Overall, when no prior information is available for the DNN model under assessment, the iterative approach is particularly effective, as it guarantees convergence to the target FR estimate with a minimal number of FIs. This is achieved without requiring any preliminary estimation, relying solely on runtime monitoring of the injection process.

IV. FLAUERS: A NOVEL FAULT INJECTION TECHNIQUE FOR SYSTOLIC ARRAY ACCELERATED DNNS

A. Motivation

Given the complexity of Neural Networks (NNs) and the fact that they are oftentimes treated as black boxes, one of the main techniques employed to study the reliability of NNs relies on Fault Injection (FI). There are three families of FI methods: hardware-based, platform-based and simulation-based [10]. Given that the first two are very expensive, and in general not easily accessible, this work is focused on simulation-based

Authorized licensed use limited to: Tallinn University of Technology. Downloaded on November 10,2026 at 15:22:09 UTC from IEEE Xplore. Restrictions apply.

techniques. More specifically, we can distinguish between hardware-agnostic and hardware-aware simulations.

- **Hardware-Agnostic** In hardware-agnostic simulations, faults are injected into the NN model without considering the specifics of the underlying hardware. This approach is flexible and widely used for initial reliability assessments, but it may not capture hardware-specific vulnerabilities or provide accurate reliability metrics for real-world deployment [10], [11].
- **Hardware-Aware** Hardware-aware techniques take into account the specifics of the hardware platform and is more accurate than the other type. Faults are injected at lower abstraction levels, such as the instruction set architecture (ISA) or Register Transfer Level, which allows for more accurate reliability and resilience assessments. These methods are essential for safety-critical applications, as they provide insights into how faults propagate through the actual hardware and can guide the design of more robust systems [10].

As of today, there is a need to study the NNs while being executed (at least partly) on a faulty Systolic Array, to better assess the challenges and potential failures of a NN application. For this reason, we aimed at developing a hardware-aware simulation-based FI technique. In the literature, the only true way of doing this is by either using valid Hardware Description Language (HDL) models [8], [9], [25] or by using ad-hoc simulators like SCALE-sim [26] (which has been used in [12]) or S-SIM [27] (which has never used for FI to our knowledge). Given this context, we find that there is no software-based hardware-aware FI framework that can be efficiently used with state-of-the-art models. We created **flauers** (**f**aulty **l**ogic **a**ssessment **u**nit for **e**nhancing **r**eliability of **s**ystolic systems) to address this gap.

B. Background

The majority of SAs works as matrix multipliers. To this extent, we will introduce the necessary formalization to perform FI.

Given two matrices $A \in \mathbb{R}^{N_1,N_3}, B \in \mathbb{R}^{N_3,N_2}$ we are interested in the problem of computing $C \in \mathbb{R}^{N_1,N_2}$ s.t. $C = A \times B$. Each element c_{ij} of matrix C is computed as the sum of multiple products:

$$c_{ij} = \sum_{k=0}^{N3} a_{ik} \times b_{kj} \tag{31}$$

where a_{ij} corresponds to the element of A in the i-th row and the j-th column. We can construct a Regular Iterative Algorithm (RIA) [28] as follows:

$$\begin{aligned} c(i,j,k) &= c(i,j,k-1) + a(i,j-1,k) \times b(i-1,j,k) \\ a(i,j,k) &= a(i,j-1,k) \\ b(i,j,k) &= b(i-1,j,k) \\ &\text{initial conditions} \\ a(i,0,k) &= a_{ik}, \quad \forall i,k \\ b(0,j,k) &= b_{kj}, \quad \forall k,j \\ c(i,j,0) &= 0, \quad \forall i,j \end{aligned} \tag{32}$$

the index vector is $p = (i,j,k)$, with $1 \leq i \leq N_1$, $1 \leq j \leq N_2$, $1 \leq k \leq N_3$.

These equations are the basis for our FI framework. This is because there exist a powerful relationship between the iteration space (described by the values of p) and the physical space $z = (x,y,t)$, where x and y describe the coordinate of the Processing Element and t is the Clock Cycle (CC). The relationship is not unique and depends on the *shape* of the SA. To make things simple, we can have a linear relationship [29]:

$$s = Tp, \tag{33}$$

where T is a projection matrix that describes the shape of the SA.

C. Fault Injection

We will see how to inject values in a hardware-like way using the mathematical framework.

In order to identify a fault, some parameters are needed:

1) the PE, indicated with the physical position (x,y),
2) the register (to choose among a, b and c),
3) the fault type.

Given these parameters, we can use the inverse of Equation 33 to calculate the points $p = (i,j,k)$ that will are affected by the fault. These points are then collected in three sets: $\hat{S}_a, \hat{S}_b, \hat{S}_c$, each for a register. Also note that gathering the points is not a trivial task: the fault does forward in space and time. For this reason, it is necessary to take into account the dependencies of the values and correctly insert them into the sets $\hat{S}_i$. Once the points are gathered, it is possible to perform a matmul with some additional checks on $p = (i,j,k)$ With this, we can perform a normal matmul with some checks on $p = (i,j,k)$ to perform the injection. The process is described in Algorithm 1.

D. Software Implementation

To (hopefully) keep the pace with modern NN algorithms, flauers was implemented in python and additional interfaces were made available for PyTorch. Figure 6 shows the way all the components are interconnected together. Please note that, even if there is a "tensorflow interface" box in the figure, such component was never implemented as all the experiments were performed in PyTorch. These interfaces are needed to allow flauers to perform the injections on real data that is likely to predict the behavior of in-field deployed applications. To this extent, such interfaces have three main components: (i) model parsing, which is needed to find the compatible

Authorized licensed use limited to: Tallinn University of Technology. Downloaded on November 10,2026 at 15:22:09 UTC from IEEE Xplore. Restrictions apply.

Algorithm 1 matmul in flauers

procedure FLAUERSMATMUL($A, B, \hat{S}, \hat{F}$)
  for $i = 1..N1$ **do**
    for $j = 1..N2$ **do**
      $c = 0$
      for $k = 1..N3$ **do**
        $a = inject(A_{i,k})$ if $(i,j,k) \in \hat{S}_a$ else $A_{i,k}$
        $b = inject(B_{k,j})$ if $(i,j,k) \in \hat{S}_b$ else $B_{i,k}$
        $p = a \cdot b$
        $c \mathrel{+}= inject(p)$ if $(i,j,k) \in \hat{S}_c$ else p
      end for
      $C_{i,j} = c$
    end for
  end for
  return C
end procedure

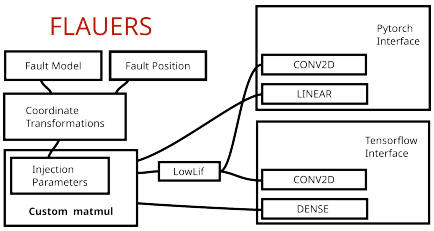


Fig. 6: Flauers software framework.

layers with SA acceleration, (ii) a Dense layer wrapper, which allows the computation to be effectively offloaded to flauers, and (iii) a Conv2D layer wrapper, which performs the Im2Col transformation before processing the convolution as a matmul. Please note that Im2Col is not the only option available. Due to the setting of the problem itself, many other alternatives exist [30]. The class which implements this general idea is called, in our tool, LowLif (from the operations of Lowering and Lifting). The fault models that are implemented are only two for the moment: stuck-at faults and bitflips. Nevertheless, it is possible to implement more sophisticated kinds of faults which might more closely resemble reality, but this is outside the scope of this text. Using this framework, it is possible to obtain an improvement of simulation time of about 10x with respect to RTL simulations, expanding the possibilities of what can be explored. The software is completely free and open source and available at the link https://github.com/jspkay/flauers.

E. Future Directions

We believe that developing a tool like flauers is an important step into the study of the reliability of NN accelerators. Using a framework like this, enables the users to explore a bigger space: before flauers the study of reliability was confined to either algorithm-only or hardware-only studies. Flauers opens the door for co-variability. Nevertheless, this is just a first step towards that direction. Many improvements can be introduced to the tool. Here we will list what we believe are the three most important features that are missing to the current state of the tool.

1) **Better library support** - we developed only pytorch interfaces for flauers. Although this is a good first step, we are far away from optimality. For example, even in PyTorch itself, we did not implement the quantized models interface, which would be of greater relevance, especially for edge devices. On top of that, TensorFlow interfaces should also be implemented, to open the door to the majority of existing models in the wild.
2) **Injectable components** - As of today, flauers only allows the injection of faults in the specific component of the matrix multiplier. Nevertheless, the same ideas can be applied to other components, most notably the LowLif module. Such implementation would further enhance *study-ability* of the systems.
3) **Fault models** - another important step is to insert fault models that closely reflect the behavior of the different elements inside the SA. To achieve that, it is possible to create statistical models of the error produced by a specific class of fault. These fault models could closely predict the application behavior for different technologies and topologies, while also enhancing functional testing of NN systems.

In conclusion, this was one small step for software, but one giant leap for hardware simulation.

V. CONCLUSIONS

The paper presented three complementary approaches to the reliability assessment of DNN inference on systolic arrays, addressing different aspects of this task. SAFPA demonstrates that analytical fault propagation analysis can replace costly microarchitecture-level simulation, achieving speed-ups of several orders of magnitude while supporting all three major SA dataflows. Its closed-form error equations enable rapid exploration of the vulnerability landscape across layers, dataflows, and array sizes, providing insights for fault-tolerant design. The iterative statistical fault injection methodology addresses the scalability bottleneck of fault injection campaigns by intelligently minimizing the required number of injections. The proposed error-probability curve adapts the margin reduction to the estimated failure rate, making the approach particularly effective for large fault populations where conservative methods would require near-exhaustive injection. The FLAUERS framework fills the gap between hardware-agnostic and RTL-level fault injection by providing an accessible, open-source tool that models SA-specific fault propagation within a standard deep learning workflow.

The tools are available as open-source software, supporting the research community and facilitating reproducibility. Future work includes extending these methodologies to emerging DNN architectures such as transformers and supporting quantized and mixed-precision inference, as well as addressing other fault models to account for technology-specific failure mechanisms.

Authorized licensed use limited to: Tallinn University of Technology. Downloaded on November 10,2026 at 15:22:09 UTC from IEEE Xplore. Restrictions apply.

ACKNOWLEDGMENT

This work was supported in part by the European Union under Grant Agreement No. 101160182 “TAICHIP”, and by the Estonian Research Council under grants PRG1467 “CRASHLESS” and CoE TK202 ”Universum”.

REFERENCES

[1] A. Ruospo, E. Sanchez, L. Matana Luza *et al.*, “A survey on deep learning resilience assessment methodologies,” *IEEE Computer Magazine*, vol. 56, no. 2, pp. 57–66, Feb. 2023.

[2] ISO, “ISO 26262: Road vehicles – functional safety,” International Organization for Standardization, 2018.

[3] ISO, “ISO/PAS 8800: Road vehicles – safety and artificial intelligence,” International Organization for Standardization, 2024.

[4] N. P. Jouppi, C. Young, N. Patil *et al.*, “In-datacenter performance analysis of a tensor processing unit,” in *ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, 2017, pp. 1–12.

[5] Y.-H. Chen, J. Emer, and V. Sze, “Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks,” in *ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, 2016, pp. 367–379.

[6] N. Cherezova, A. Jutman, and M. Jenihhin, “FORTALESA: Fault-tolerant reconfigurable systolic array for DNN inference,” *Microprocessors and Microsystems*, vol. 119, p. 105222, 12 2025.

[7] F. Libano, P. Rech, and J. Brunhaver, “Efficient error detection for matrix multiplication with systolic arrays on FPGAs,” *IEEE Transactions on Computers*, vol. 72, pp. 2390–2403, 8 2023.

[8] 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)*, 2021, pp. 1–6.

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

[10] M. H. Ahmadilivani, M. Taheri, J. Raik *et al.*, “A systematic literature review on hardware reliability assessment methods for deep neural networks,” *ACM Computing Surveys*, vol. 56, no. 6, pp. 1–39. [Online]. Available: https://dl.acm.org/doi/10.1145/3638242

[11] M. Taheri, M. Daneshtalab, J. Raik *et al.*, “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)*, 2024, pp. 19–24.

[12] J. Tan, Q. Wang, K. Yan *et al.*, “Saca-FI: A microarchitecture-level fault injection framework for reliability analysis of systolic array based CNN accelerator,” *Future Generation Computer Systems*, vol. 147, pp. 251–264, 10 2023.

[13] R. Billig Tonetto, C. Gava, P. Rech, and J. Brunhaver, “ENFOR-SA: End-to-end cross-layer transient fault injector for efficient and accurate DNN reliability assessment on systolic arrays,” *arXiv preprint arXiv:2602.00909*, 2026.

[14] C. Bolchini, L. Cassano, A. Miele, and A. Toschi, “Fast and accurate error simulation for CNNs against soft errors,” *IEEE Transactions on Computers*, vol. 72, pp. 984–997, 4 2023.

[15] 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*, 2009, pp. 502–506.

[16] A. Ruospo, G. Gavarini, C. de Sio *et al.*, “Assessing convolutional neural networks reliability through statistical fault injections,” in *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2023, pp. 1–6.

[17] A. Ruospo, M. Sonza Reorda, R. Mariani, and E. Sanchez, “An effective iterative statistical fault injection methodology for deep neural networks,” *IEEE Transactions on Computers*, vol. 74, no. 7, pp. 2431–2444, 2025.

[18] K. Chellapilla, S. Puri, and P. Simard, “High performance convolutional neural networks for document processing,” in *10th International Workshop on Frontiers in Handwriting Recognition*, 2006.

[19] J. J. Elliott, F. Mueller, M. K. Stoyanov, and C. G. Webster, “Quantifying the impact of single bit flips on floating point arithmetic,” Oak Ridge National Laboratory (ORNL), Oak Ridge, TN (United States), Tech. Rep., 08 2013.

[20] S. S. Mukherjee, C. Weaver, J. Emer *et al.*, “A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor,” *Annual International Symposium on Microarchitecture, MICRO*, vol. 2003-January, pp. 29–40, 2003.

[21] P. Ramachandran, P. Kudva, J. Kellington *et al.*, “Statistical fault injection,” in *2008 IEEE International Conference on Dependable Systems and Networks With FTCS and DCC (DSN)*, 2008.

[22] R. Johnson, I. Miller, and J. Freund, *Miller & Freund’s Probability and Statistics for Engineers*, ser. Pearson Modern Classics for Advanced Statistics Series. Pearson Education, 2018.

[23] I. Tuzov, D. de Andrés, and J.-C. Ruiz, “Accurate robustness assessment of hdl models through iterative statistical fault injection,” in *2018 14th European Dependable Computing Conference (EDCC)*, 2018, pp. 1–8.

[24] CAD & Reliability Group, Politecnico di Torino (Italy), “IterativeFI: Iterative statistical fault injection framework for deep neural networks,” https://github.com/cad-polito-it/IterativeSFI, 2026, gitHub Repository.

[25] U. K. Agarwal, A. Chan, A. Asgari, and K. Pattabiraman, “Towards reliability assessment of systolic arrays against stuck-at faults,” in *2023 53rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks - Supplemental Volume (DSN-S)*, pp. 230–236, ISSN: 2833-292X. [Online]. Available: https://ieeexplore.ieee.org/document/10206919/

[26] A. Samajdar, Y. Zhu, P. Whatmough *et al.*, “SCALE-sim: Systolic CNN accelerator simulator.” [Online]. Available: http://arxiv.org/abs/1811.02883

[27] Y. Li, M. Wen, R. Yang *et al.*, “S-SIM: A simulator for systolic array-based DNN accelerators with tile access awareness,” in *2022 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 2720–2724, ISSN: 2158-1525. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9937624

[28] S. K. Rao and T. Kailath, “Regular iterative algorithms and their implementation on processor arrays,” *Proceedings of the IEEE*, vol. 76, no. 3, pp. 259–269, 1988.

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

[30] A. Anderson, A. Vasudevan, C. Keane, and D. Gregg, “High-performance low-memory lowering: Gemm-based algorithms for dnn convolution,” in *2020 IEEE 32nd International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, 2020, pp. 99–106.

Authorized licensed use limited to: Tallinn University of Technology. Downloaded on November 10,2026 at 15:22:09 UTC from IEEE Xplore. Restrictions apply.

Appendix 5

V

N. Cherezova, M. Jenihhin, and A. Jutman. Scalable reliability assessment of vision transformers on systolic arrays via fault propagation analysis. In *IEEE 32nd International Symposium on On-Line Testing and Robust System Design (IOLTS)*, pages 1–7, 2026

2026 IEEE 32nd International Symposium on On-Line Testing and Robust System Design (IOLTS) | 979-8-3315-4685-4/26/$31.00 ©2026 IEEE | DOI: 10.1109/IOLTS69666.2026.11633827

Scalable Reliability Assessment of Vision Transformers on Systolic Arrays via Fault Propagation Analysis

Natalia Cherezova*, Artur Jutman*†, Maksim Jenihhin*
*Department of Computer Systems, Tallinn University of Technology, Tallinn, Estonia
†Testonica Lab, Tallinn, Estonia
{natalia.cherezova, artur.jutman, maksim.jenihhin}@taltech.ee

***Abstract*—Vision Transformers (ViTs), utilizing a self-attention mechanism to capture global context, are increasingly deployed in safety- and mission-critical applications, yet their vulnerability to hardware faults remains underexplored compared to Convolutional Neural Networks (CNNs). In this work, we establish a fault propagation analysis framework to enable scalable, hardware-aware reliability assessment of ViT inference on systolic arrays, avoiding the need for cycle-accurate simulation. We perform fine-grained, operation-level analysis on three representative ViT models (ViT-B16, DeiT-S, and Swin-T) and compare their vulnerability with ResNet-50. Our analysis reveals that, unlike CNNs, ViT reliability is governed by operation type rather than layer position, with the value matrix computation being exceptionally critical. We also show that architectural regularity directly impacts vulnerability distribution across layers. These results enable designing operation-specific fault tolerance schemes for ViTs on hardware accelerators.**

***Index Terms*—vision transformers, convolutional neural networks, systolic array, reliability.**

I. Introduction

Vision Transformers (ViTs), which outperform traditional Convolutional Neural Networks (CNNs) in computer vision tasks, have gained significant popularity recently. However, as deep learning is used more often in safety- and mission-critical applications, such as autonomous driving, space missions, and medical fields, the reliability aspect becomes increasingly important. Since CNNs were the predominant solutions for computer vision tasks before, their reliability has been thoroughly explored. The reliability aspects of ViTs, in comparison, are less explored.

The increasing computational complexity of deep learning models, especially transformers, necessitates specialized hardware accelerators. However, since nanoelectronic devices are susceptible to aging and soft errors caused by radiation, it is of utmost importance to understand how faults in hardware components propagate in the ViT models.

Several works have been published on the reliability of Vision Transformers. [1] explores the vulnerability of ViTs to the bit-flips in the outputs of the major arithmetic operations, i.e., multiplication and addition. [2] analyzes ViT sensitivity to the faults in weights. [3] evaluates the vulnerability of the Swin Transformer model to stuck-at faults in weights and bit-flips in the output neurons. They also propose a fault-tolerance solution based on the ranging of the outputs, inputs, and weights of the fully connected layers. However, those works perform analysis on the model level, without considering the hardware platform.

[4] performs a reliability analysis of ViTs considering faults in the GPU. The evaluation is based on the data collected through neutron beam experiments and software fault simulation. [5] and [6] also explore the reliability of ViTs on a GPU under neutron radiation and compare their reliability with that of CNNs. [6] additionally explores how runtime optimization techniques, data precision reduction, and quantization affect the reliability of the selected models. [7] presents a reliability evaluation of ViTs on Coral TPU under neutron radiation.

[8] presents a reliability analysis of the text embeddings considering different tasks and datasets. The analysis is performed on the model level as well.

Compared to the existing works, we propose a novel reliability assessment method for ViT inference on systolic arrays based on fault propagation analysis. This method is used to perform fine-grained reliability analysis of three common ViT models for image classification. We investigate their vulnerability to soft errors in systolic arrays, focusing on different models, layers, and operations. We also compare the reliability of ViTs with CNNs, namely ResNet-50, following previous works [1], [5], [6].

The contributions of the paper are:

- a scalable hardware-aware reliability assessment method for ViT inference on systolic arrays based on fault propagation analysis;
- a fine-grained reliability analysis of three representative ViT models (ViT-B16, DeiT-S, Swin-T) with comparison to CNNs, revealing: a) operation-dependent vulnerability rather than layer position dependent, i.e., a fundamental difference from CNNs; b) identification of the value matrix computation as especially critical; c) and the effect of ViTs architectural regularity on reliability.

The rest of the paper is organized in the following way. Section II presents the background on the ViT architecture and systolic arrays. Section III introduces a reliability assessment method for ViT inference on systolic arrays based on fault propagation analysis. Section IV describes the experimental setup. Section V presents an empirical analysis of ViT reliability, and Section VI concludes the paper.

979-8-3315-4685-4/26/$31.00 ©2026 IEEE
Authorized licensed use limited to: Tallinn University of Technology. Downloaded on November 10,2026 at 15:21:01 UTC from IEEE Xplore. Restrictions apply.

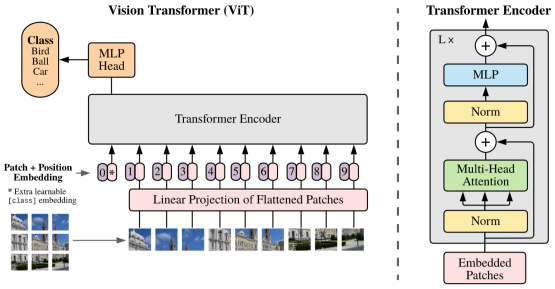


Fig. 1: Architecture of a Vision Transformer [9]

II. Background

A. Vision Transformers

Vision Transformers were proposed in 2021 by Dosovitskiy et al. [9]. The architecture of basic ViT is presented in Fig. 1. First, an input image is transformed into a sequence of 2D patches. The patches are flattened and mapped to D dimensions through a trainable linear projection. A classifier embedding is added to the resulting patch embeddings. The embeddings are then passed through a sequence of Transformer Encoder blocks. Each encoder block includes Multi-Head Attention (MHA) and Multi-Layer Perceptron (MLP), consisting of two fully-connected (FC) layers.

The MHA block defines where to put focus in the image and draws connections between patches. It starts with the calculation of **query** (Q), **key** (K) and **value** (V) vectors for each patch:

$$Q = XW_Q,\ K = XW_K,\ V = XW_V, \tag{1}$$

where X is the embedding matrix, W_Q, W_K, W_V are corresponding weights.

Then, the query and transposed key matrices are multiplied to get attention weight or **attention score** (S_{attn}):

$$S_{attn} = QK^T. \tag{2}$$

Attention value or a **context** (Z) is then calculated as

$$S_{sim} = \text{softmax}\left(\frac{S_{attn}}{\sqrt{d_k}}\right) \tag{3}$$

$$Z = S_{sim}V, \tag{4}$$

where S_{sim} is a similarity score, and d_k is the dimension of the Q, K and V vectors.

The idea behind MHA is that each head learns different relationships and provides a unique perspective on the input image. The attentions from all heads are concatenated and passed through a linear layer to get the **MHA output**:

$$O = ZW_Z. \tag{5}$$

The MHA output (O) is then passed to the MLP block, as mentioned earlier, consisting of two FC layers. Unlike CNNs, where FC layers are represented by vector-matrix multiplication, in ViTs, FC layers are represented by matrix-matrix multiplication.

The output of the final encoder layer corresponding to the added classifier embedding is passed to the MLP Head for the classification. FC layer of the head is a vector-matrix multiplication.

B. Systolic Arrays Architecture

A systolic array is a 2D grid of Processing Elements (PEs) that include a Multiply-Accumulate (MAC) unit and registers holding inputs and outputs (partial sums). Input data comes from the top and left side and propagates through the grid. Weights propagate from top to bottom, activations (inputs) propagate from left to right. Different types of systolic arrays include weight-stationary, input-stationary, and output-stationary, defined by the dataflow. The dataflow defines which values are stationary and which propagate through the array during calculation.

The accelerators for ViTs proposed in the literature [10], [11] are based on output-stationary systolic arrays, since they allow high input and output reuse. In the output-stationary systolic arrays, weights and inputs propagate through the array, and outputs are accumulated inside the PEs. For this reason, the analysis in this work is performed for the output-stationary systolic arrays.

III. Methodology

The proposed reliability assessment method is an extension of the method for CNNs presented in [12] for Vision Transformers. *This method combines fault injection with fault propagation analysis, which is used to calculate the resulting error in the operation output*, instead of performing time-consuming cycle-accurate simulations using a microarchitectural or RTL model of a systolic array. This allows for fast and scalable assessment. The method is several orders of magnitude faster than the microarchitecture-level fault injection [12].

The previous method [12] was developed for convolutional layers. However, all major arithmetic operations in ViTs are represented by matrix-matrix multiplication, whether it involves multiplying two previously calculated values or multiplying a previously calculated value by a weight. Fault propagation in matrix multiplication is different compared to convolution. Therefore, an extension to support ViTs was developed based on new error patterns and new equations for their calculation.

A. Fault Propagation in Systolic Arrays

Single transient faults (bit flips) in PE registers holding inputs (activations), weights, and partial sums (outputs) are considered for the analysis, following previous works [13]. Due to the regular structure of a systolic array and the propagation of the data, faults in different registers result in distinct error patterns in the output of the matrix multiplication. The error patterns were presented in [14]. A fault in a weight register (WREG) results in the **vertical line** error pattern, since a faulty weight propagates through the entire column and corrupts several pixels in the output. A fault in the input register (IREG) results in the **horizontal line** pattern, since faulty activation propagates through the entire row. Finally, a fault in the output register (OREG) holding partial sum results in the **point pattern**, since output values are stationary.

Authorized licensed use limited to: Tallinn University of Technology. Downloaded on November 10,2026 at 15:21:01 UTC from IEEE Xplore. Restrictions apply.

B. Fault Propagation Analysis

Each fault is defined by seven parameters given in Table I.

TABLE I: Transient fault parameters

Symbol	Description
f_{type}	Type of fault (IREG, WREG, OREG)
ts	Cycle of a systolic array execution
t_w	Weight tile
t_a	Activation tile
p_{row}	Row of the PE
p_{col}	Column of the PE
β_f	Bit index

A bit flip in the numerical value x can be expressed as an addition of the error term ε. In case of an integer value, this error term is defined as $\pm 2^{\beta_f}$. In the case of a floating point, the error term is different for the sign bit, exponent, and mantissa bits [15]. For this study, a 32-bit floating-point arithmetic is considered. Therefore, an error term is defined as

$$\varepsilon = \begin{cases} (-1)^{b_f} 2^{E} 2^{\beta_f - 24} & \text{if} \quad 0 \leq \beta_f \leq 23 \quad \text{(mantissa)} \\ (2^{(-1)^{b_f} 2^{\beta_f - 24}} - 1)x & \text{if} \quad 24 \leq \beta_f \leq 31 \quad \text{(exponent)} \\ -2x & \text{if} \quad \beta_f = 32 \quad \text{(sign)} \end{cases} \tag{6}$$

where β_f is an index of a faulty bit, b_f is the value of the β_f-th bit, and E is an exponent of numerical value x, expressed as

$$E = \sum_{i=24}^{31} b_i 2^{i-23} . \tag{7}$$

The result of the matrix multiplication is the sum of the element-wise multiplication between rows of the first input and columns of the second input. *A fault in one of the components results in the addition of error e to one or several values in the output.* Multiplying input matrix $X \in \mathbb{R}^{P \times S}$ and weight matrix $W \in \mathbb{R}^{S \times R}$, we get output matrix $Y \in \mathbb{R}^{P \times R}$.

Fault in input register (IREG). If the fault happens in the input register, it corrupts an activation value coming from the side and results in the horizontal line error pattern. Therefore, an error e_{ireg} is added to several output values in one row. The affected output values are defined by the row of the output matrix and the range of columns.

An error added to the output equals $e_{ireg} = w_{(i_f, j_f)} \cdot \varepsilon_x$, where $w_{(u_f, v_f)}$ is the weight multiplied with the faulty activation, and ε_x is an error term. The faulty activation is found by its position (i_f, j_f):

$$i_f = (t_a - 1) \cdot N + p_{row} \tag{8}$$

$$j_f = ts - p_{col} - p_{row} \tag{9}$$

where t_a is the targeted activation tile, N is the size of the systolic array, ts is the cycle of a systolic array execution, p_{row} and p_{col} denote the position of the targeted PE.

The weight is found by its position (u_f, v_f):

$$u_f = j_f \tag{10}$$

$$v_f = \left[v_f^{start}, v_f^{end} \right] \tag{11}$$

$$v_f^{start} = (t_w - 1) \cdot N + p_{col} + 1 \tag{12}$$

$$v_f^{end} = \begin{cases} t_w \cdot N & \text{if} \quad R \geqslant t_w \cdot N \\ Q & \text{if} \quad R < t_w \cdot N \end{cases} \tag{13}$$

where $v_{out,f}$ is a range of affected columns, starting with v_f^{start} and ending with v_f^{end}, t_w is the targeted weight tile, and R is the width of the output matrix.

The affected output row is found using Eq. (8) and the range of columns is found using Eq. (11)–(13).

Fault in weight register (WREG). In this case, a weight value coming from the top is corrupted, resulting in the vertical line error pattern. Therefore, an error e_{wreg} is added to the column of the output matrix. The affected output values are defined by the range of rows and one column.

An error added to the output in this case equals

$$e_{wreg} = x_{(i_f, j_f)} \cdot \varepsilon_w , \tag{14}$$

where $x(i_f, j_f)$ is the activation multiplied by the faulty weight. The activation is found by its position (i_f, j_f):

$$i_f = \left[i_f^{start}, i_f^{end} \right] \tag{15}$$

$$i_f^{start} = (t_a - 1) \cdot N + p_{row} \tag{16}$$

$$i_f^{end} = \begin{cases} t_a \cdot N & \text{if} \quad P \geqslant t_a \cdot N \\ P & \text{if} \quad P < t_a \cdot N \end{cases} \tag{17}$$

$$j_f = ts - p_{col} - p_{row} \tag{18}$$

where P is the height of the output matrix.

The faulty weight is found by its position (u_f, v_f):

$$u_f = (t_w - 1) \cdot N + p_{col} \tag{19}$$

$$v_f = j_f \tag{20}$$

The affected output rows are found using Eq. (15)–(17) and the column is found using Eq. (19).

Fault in the output register (OREG). A fault in the output register results in the point error pattern. An error e_{oreg} is added to a single value in the output matrix. Its position is defined by the row and column. An error added to the output equals $e_{oreg} = \varepsilon_y$, where ε_y is an error term added to the faulty partial sum. The row is found using Eq. (8), the column using Eq. (19).

C. Implementation

The proposed reliability assessment method combines fault injection with fault propagation analysis. Pytorch is used to run the inference of vision transformers, taking advantage of GPU speed up. The fault injection module gets the output of the targeted layer or operation and modifies only affected output values according to the equations presented in the previous subsection. The inference continues with the erroneous values. The implementation of the method will be made open-source.

IV. Experimental Setup

Three common vision transformer models were selected for the experiments, including the original ViT-B16 [9], DeiT-S [16], and Swin-T [17]. ResNet-50 was selected as an example of CNNs. The parameters of the tested ViT models are presented in Table II. ResNet-50 is trained on Imagenet1K and tested considering a 128×128 systolic array. The fault injection framework is implemented in Python using PyTorch. Experiments were performed on an NVIDIA A100 GPU.

Architectural vulnerability factor (AVF) was used as a reliability measure following previous works [13]. AVF is the

Authorized licensed use limited to: Tallinn University of Technology. Downloaded on November 10,2026 at 15:21:01 UTC from IEEE Xplore. Restrictions apply.

TABLE II: Parameters of the tested vision transformers

Model	Dataset	SA size	Input size	Patch size	Embedding size	# of layers	# of heads	MLP dimension
ViT-B16	CIFAR-100	32×32	224×224*	16×16	768	12	12	3072
ViT-B16	Imagenet1K	128×128	224×224	16×16	768	12	12	3072
DeiT-S	Imagenet1K	128×128	224×224	16×16	384	12	6	1536
Swin-T	Imagenet1K	128×128	224×224	4×4	96	(2,2,6,2)	(3,6,12,24)	384

* CIFAR-100 images were upscaled to 224×224.

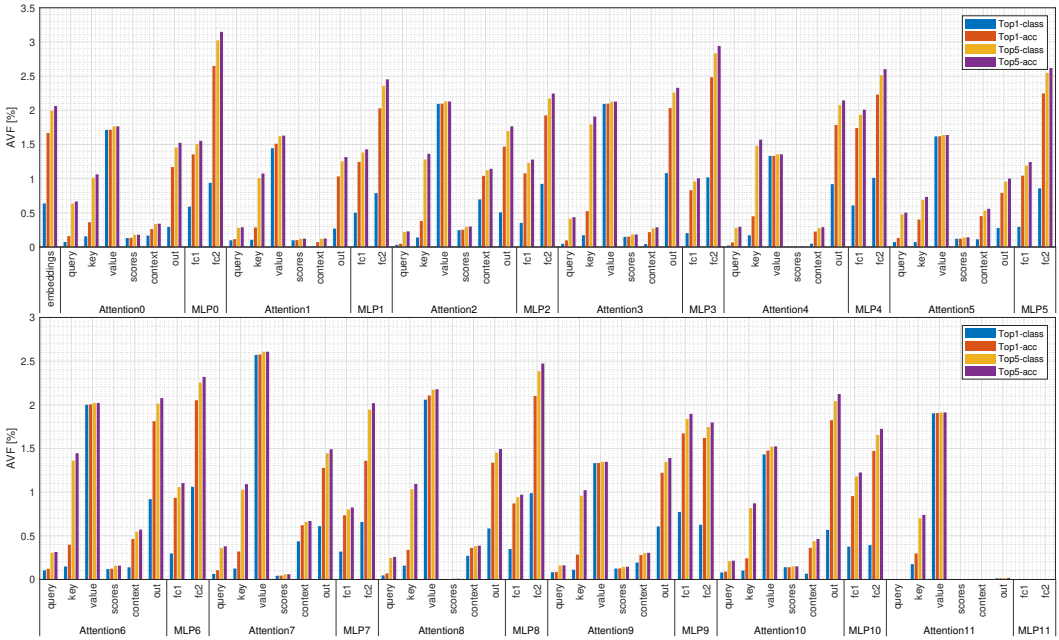


Fig. 2: Layer- and operation-wise vulnerability assessment of ViT-B16 considering transient faults (128×128 systolic array)

probability that a fault in the hardware structure causes an application output error [18]. It is calculated as the ratio of fault injection runs with output errors to the total number of fault injection runs [13]. Since the output of the selected models is the probability score of each class, the following output errors were considered [13]: Top1-class (the top-ranked class is different), Top1-acc (the probability score of the top-ranked class is different), Top5-class (at least one class in the top-5 is different), Top5-acc (the probability score of at least one class in the top-5 is different).

Layer-wise and operation-wise statistical fault injection was performed using the equation introduced in [19] to achieve 95% confidence and 5% error margin, ensuring more than 1000 faults for each layer and operation. For each fault injection, fault parameters (Table I) were set randomly, and a test dataset of 10 000 inputs was fed to the network. Faults were injected in each layer and operation that can be mapped to a systolic array, following the presented methodology (Section III).

V. Empirical analysis

A. Operation-dependent vulnerability pattern

Figures 2 and 3 present AVF values for each layer and operation of selected transformer models. Figure 4 presents AVF values for each convolutional layer in ResNet-50. For the sake of space, all four output error categories are presented only for the ViT-B16 model on a 128×128 systolic array. For the other models, only Top1-class AVF values are presented.

ResNet-50 (Fig. 4) shows high variations in vulnerability across the layers, common for CNNs. In ViTs, on the other hand, certain operations, namely value, context, MHA output, and both FC layers, consistently show high vulnerability. It can also be observed that the variations in the operations AVFs across layers (encoder blocks) are low, except for the Swin-T model. *It shows that while the reliability of CNNs is defined by the position of the layer, the reliability of ViTs is defined by the type of operation*. This means that, unlike reliability solutions for CNNs that proposed different levels of protection for different layers based on their vulnerability, the fault tolerance solution for ViTs should propose different levels of protection for different operations. Potentially by deploying several systolic arrays, one for performance, one for fault tolerance, since several operations in the MHA block can be performed in parallel, thus absorbing the timing overhead that can be introduced by fault tolerance.

B. Significant criticality of the value matrix

Figure 2 shows that, while for most operations the vulnerability factors for Top1-acc, Top5-class, and Top5-acc errors are

Authorized licensed use limited to: Tallinn University of Technology. Downloaded on November 10,2026 at 15:21:01 UTC from IEEE Xplore. Restrictions apply.

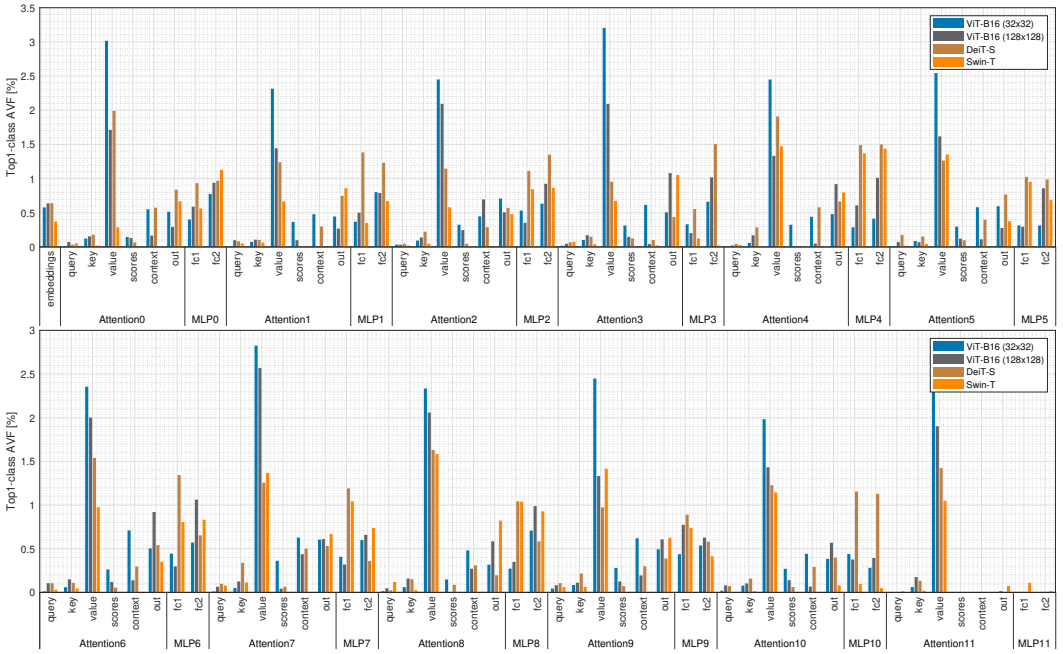


Fig. 3: Layer- and operation-wise vulnerability assessment of tested vision transformers considering transient faults

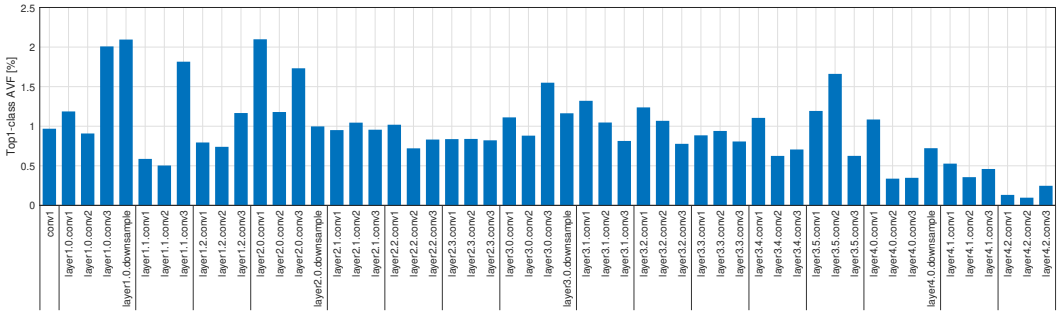


Fig. 4: Layer-wise vulnerability assessment of ResNet-50 considering transient faults (128×128 systolic array)

noticeably higher than those for Top1-class errors, for value calculations, they are always similar. It means that if a fault in value calculation affects the top-5 predictions, it almost always results in misclassification. For other operations, faults more often affect the probability score of the top-5 classes. The exceptional criticality of the value is confirmed by Fig. 5 showing the average Top1-class AVF for each operation in an encoder block. *The vulnerability of value is significantly higher than the vulnerability of other operations*. This can be explained by the fact that the faults in query and key computations propagate into the attention score (Eq. (2)) and then get attenuated by the softmax normalization (Eq. (3)), while faults in the value matrix pass through linearly (Eq. (4)). Moreover, while systolic array size and model size affect the vulnerability of other operations, the high criticality of the value matrix persists.

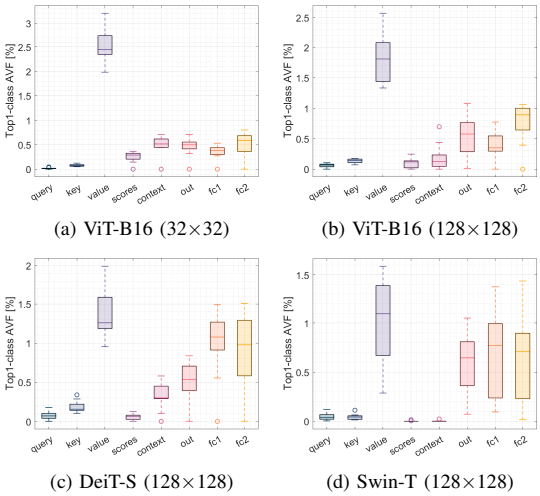
C. Reliability of embeddings

Figure 6 presents the average Top1-class AVF for embeddings and each encoder block in the networks. Contrary to some previous works [3], [7] that report patch embedding as the most critical component of vision transformers, it can be seen that embedding vulnerability is comparable to the average AVF of encoder blocks. Detailed results (Fig. 2 and 3) show that certain encoder operations are more critical.

D. Effect of architectural regularity on reliability

From Fig. 3 and 6, it can be seen that for the Swin-T model, the variation in AVF values across layers (encoder blocks) is high, resembling CNN behavior (e.g., ResNet-50 on Fig. 4). Models like ViT and DeiT have the same dimensions across the layers. Conversely, in Swin transformers, due to the smaller patches merging into bigger ones in the deeper layers and

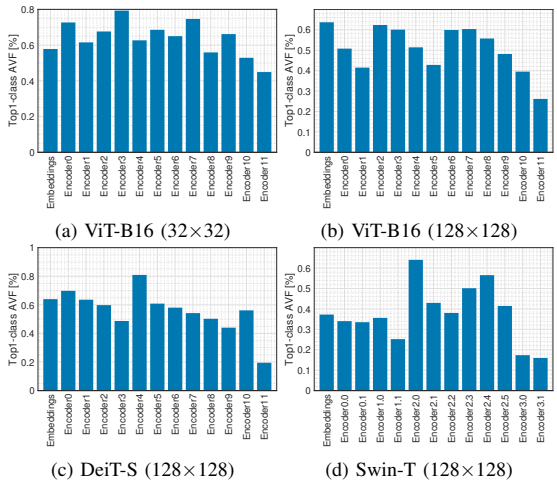
Authorized licensed use limited to: Tallinn University of Technology. Downloaded on November 10,2026 at 15:21:01 UTC from IEEE Xplore. Restrictions apply.



(a) ViT-B16 (32×32) (b) ViT-B16 (128×128)

(c) DeiT-S (128×128) (d) Swin-T (128×128)

Fig. 5: Layer- and operation-wise vulnerability comparison



(a) ViT-B16 (32×32) (b) ViT-B16 (128×128)

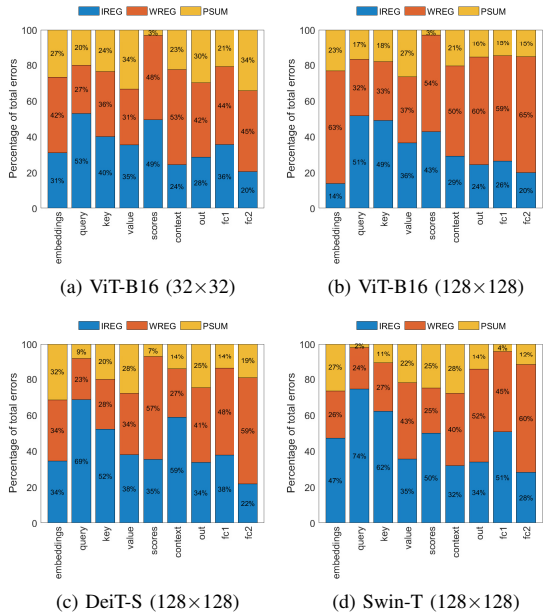
(c) DeiT-S (128×128) (d) Swin-T (128×128)

Fig. 6: Embeddings and encoder blocks vulnerability

to the increase in the number of heads in MHA, dimensions change across the layers, similar to CNNs. This change in size might explain higher variations in AVF values. *This shows that architectural regularity has an impact on model vulnerability across the layers*. Single-resolution models like ViT and DeiT show more uniform reliability across the layers. Hierarchical models like Swin show high variation across the layers, similar to CNNs.

E. Fault distribution

Figure 7 presents average error rates of PE registers for each operation in an encoder block. For the attention score, the query matrix goes through the input pipeline, and the transposed key matrix goes through the weight pipeline. For the context, the similarity score is considered as input, and the value matrix



(a) ViT-B16 (32×32) (b) ViT-B16 (128×128)

(c) DeiT-S (128×128) (d) Swin-T (128×128)

Fig. 7: Layer- and operation-wise fault distribution

as weights. It can be noticed that for query and key, faults in the input register, corrupting several values in one patch embedding, have a higher error rate. For the other operations, predominantly faults in the weight register, corrupting the same value in several patch embeddings, have a higher error rate. Noticeably, value has similar error rates for all registers.

VI. Conclusion

In this work, we present a reliability assessment method for ViT inference on systolic arrays based on fault propagation analysis. This method is used to perform fine-grained reliability analysis of three common ViT models for image classification, namely ViT-B16, DeiT-S, and Swin-T. We investigate their vulnerability to soft errors in systolic array registers, focusing on different models, layers, and operations. We also compare the reliability of ViTs with the ResNet-50 CNN. An empirical analysis shows that ViTs exhibit an operation-dependent vulnerability pattern, with significant criticality of value matrix calculation. It also shows the effect of architectural regularity on reliability, specifically that hierarchical transformer models like Swin have high vulnerability variation across the layers, resembling the behavior of CNNs.

Acknowledgment

This work was supported in part by the European Union under Horizon Europe Grant Agreement No. 101160182 “TAICHIP”, and by the Estonian Research Council under grants PRG1467 “CRASHLESS” and CoE TK202 “Foundations of the Universe”.

Authorized licensed use limited to: Tallinn University of Technology. Downloaded on November 10,2026 at 15:21:01 UTC from IEEE Xplore. Restrictions apply.

REFERENCES

[1] X. Xue, C. Liu, Y. Wang, B. Yang, T. Luo, L. Zhang, H. Li, and X. Li, "Soft error reliability analysis of vision transformers," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 31, no. 12, pp. 2126–2136, 2023.

[2] J. He, Y. Liu, C. Xu, X. Liao, and Y. Yang, "Fine-grained fault sensitivity analysis of vision transformers under soft errors," *Electronics*, vol. 14, p. 2418, 2025.

[3] G. Gavarini, A. Ruospo, and E. Sanchez, "Evaluation and mitigation of faults affecting Swin transformers," in *IEEE 29th International Symposium on On-Line Testing and Robust System Design (IOLTS)*, 2023.

[4] L. Roquet, F. F. D. Santos, P. Rech, M. Traiola, O. Sentieys, and A. Kritikakou, "Cross-layer reliability evaluation and efficient hardening of large vision transformers models," in *61st ACM/IEEE Design Automation Conference (DAC)*, 2024.

[5] F. F. Dos Santos, P. Rech, A. Kritikakou, and O. Sentieys, "Neutron-induced error rate of vision transformer models on GPUs," in *RADECS*, 2023, pp. 1–4.

[6] J. M. Badia, I. Martin-Salinas, G. Leon, A. Amor-Martin, L. Frias-Dominguez, J. A. Belloch, M. Garcia-Valderas, A. Lindoso, C. Cazzaniga, and L. Entrena, "Reliability of vision transformers and CNNs on edge AI systems under neutron radiation," *IEEE Transactions on Nuclear Science*, vol. 72, no. 8, pp. 2706–2716, 2025.

[7] B. Loureiro Coelho, P. Rafael Bodmann, N. Cavagnero, C. Frost, and P. Rech, "Vision transformer reliability evaluation on the Coral Edge TPU," *IEEE Transactions on Nuclear Science*, vol. 72, no. 4, 2025.

[8] S. Sapkal, U. Zahid, G. Gambardella, H.-G. Stratigopoulos, and B. Clerkin, "On the fault sensitivity of natural language embeddings computation," in *IEEE European Test Symposium (ETS)*, 2025.

[9] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," in *ICLR*, 2021.

[10] Y. Chen, T. Li, X. Chen, Z. Cai, and T. Su, "High-frequency systolic array-based transformer accelerator on field programmable gate arrays," *Electronics*, vol. 12, no. 4, 2023.

[11] Q. Song, Y. Dai, H. Lu, and G. Jin, "High-throughput systolic array-based accelerator for hybrid transformer-CNN networks," *Journal of King Saud University - Computer and Information Sciences*, vol. 36, no. 8, p. 102194, 2024.

[12] N. Cherezova, A. Jutman, and M. Jenihhin, "FORTALESA: Fault-tolerant reconfigurable systolic array for dnn inference," *Microprocessors and Microsystems*, vol. 119, p. 105222, 2025.

[13] J. Tan, Q. Wang, K. Yan, X. Wei, and X. Fu, "Saca-FI: A microarchitecture-level fault injection framework for reliability analysis of systolic array based CNN accelerator," *Future Generation Computer Systems*, vol. 147, pp. 251–264, 2023.

[14] P. Wu, Q. Guan, N. DeBardeleben, S. Blanchard, D. Tao, X. Liang, J. Chen, and Z. Chen, "Towards practical algorithm based fault tolerance in dense linear algebra," in *25th ACM International Symposium on High-Performance Parallel and Distributed Computing (HPDC)*, 5 2016, pp. 31–42.

[15] J. J. Elliott, F. Mueller, M. K. Stoyanov, and C. G. Webster, "Quantifying the impact of single bit flips on floating point arithmetic," Oak Ridge National Laboratory (ORNL), Oak Ridge, TN (United States), Tech. Rep., 08 2013.

[16] H. Touvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles, and H. Jegou, "Training data-efficient image transformers & distillation through attention," in *International Conference on Machine Learning*, 2021, pp. 10 347–10 357.

[17] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, "Swin transformer: Hierarchical vision transformer using shifted windows," in *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021, pp. 9992–10 002.

[18] S. S. Mukherjee, C. Weaver, J. Emer, S. K. Reinhardt, and T. Austin, "A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor," in *Annual International Symposium on Microarchitecture, MICRO*, 2003, pp. 29–40.

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

Authorized licensed use limited to: Tallinn University of Technology. Downloaded on November 10,2026 at 15:21:01 UTC from IEEE Xplore. Restrictions apply.

Appendix 6

VI

M. H. Ahmadilivani, N. Cherezova, M. Glaß, J.-D. Guerrero-Balaguera, M. Jenihhin, A. Kritikakou, R. L. Sierra, L. B. Poelhs, J. Raik, L. Roquet, F. F. dos Santos, F. A. da Silva, M. S. Reorda, A. Veronesi, E. C. V. Castillo, and J. E. R. Condia. Reliability assessment of deep neural networks and accelerators across design stages. In *IEEE 27th Latin American Test Symposium (LATS)*, pages 1–10, 2026

Reliability Assessment of Deep Neural Networks and Accelerators Across Design Stages

Mohammad Hasan Ahmadilivani*, Natalia Cherezova*, Michael Glaß§, Juan-David Guerrero-Balaguera†, Maksim Jenihhin*, Angeliki Kritikakou‖, Robert Limas Sierra†, Leticia Bolzani Poelhs¶, Jaan Raik*, Lucas Roquet‖, Fernando Fernandes dos Santos‖, Felipe Augusto da Silva‡, Matteo Sonza Reorda†, Alessandro Veronesi¶, Ernesto Cristopher Villegas Castillo§‡, Josie E. Rodriguez Condia†

**Tallinn University of Technology - Department of Computer Systems, Tallinn, Estonia

§Ulm University - Institute of Embedded Systems/Real-Time Systems, Ulm, Germany

†Politecnico di Torino - Department of Control and Computer Engineering (DAUIN), Turin, Italy

‖University of Rennes - INRIA, Rennes, France

¶Leibniz Institute for High Performance Microelectronics (IHP), Frankfurt Oder, Germany

‡Cadence Design Systems, Munich, Germany

***Abstract*—Artificial Intelligence (AI) is driving the introduction of advanced functionality and autonomy across several domains, including automotive, robotics, health care, and aerospace. Such advances are supported by the availability of algorithms and hardware accelerator platforms to efficiently implement them. However, transistor miniaturization raises reliability concerns during in-field operation and poses challenges in terms of computational intensity and performance for which conventional design and fault-characterization strategies are not well-suited. This work reviews three approaches for determining fault vulnerabilities and reliability challenges in hardware accelerators, devices, and software for Deep Neural Networks. The first strategy describes a cost-effective semi-analytical reliability assessment method for the design space exploration of reliability-performance trade-offs using reconfigurable systolic array engines. The second strategy analyzes and evaluates four abstraction levels for dot-product-unit-based accelerators and assesses the implications of selecting an affordable strategy based on the analysis needs. The third approach analyzes the critical role of correctly selecting representative inputs in large Vision Transformers (ViTs) for their effective reliability evaluation.**

***Index Terms*—Colored Petri Nets, Fault Injection, Hardware Accelerators, Reliability, Special Function Units, Tensor Cores.**

I. Introduction

The disruptive adoption of AI is supported by advanced and effective algorithms, as well as the availability of sophisticated platforms for its efficient deployment, including AI hardware based on *Systolic Arrays* (SAs), *Graphic Processing Units* (GPUs), and *Dot Product Unit* (DPU) architectures [1]. In fact, AI plays a crucial role in several application fields, such as robotics, autonomous machines, self-driving cars, and, more recently, in *Internet of Things* as *Artificial Intelligence of Things* or AIoT [2], [3].

In safety-critical environments, the strict reliability requirements make the analysis and assessment of non-functional properties both mandatory and essential throughout the development and deployment of AI-powered applications, such as *Deep Neural Networks* (DNNs), as mandated by industrial standards (e.g., ISO 26262). Unfortunately, the use of conventional fault assessment and evaluation strategies raises several technical challenges. In particular, transistor miniaturization enables the integration of increasingly complex functionalities into hardware while maintaining acceptable energy consumption. However, the high transistor density, combined with the data-intensive nature of AI-powered workloads, demands substantial computational power and large assessment times. Thus, the identification, exploration, and integration of innovative methods for assessing hardware accelerators and DNNs at the design and deployment stages are crucial to improve performance while preserving characterization accuracy.

This work explores 3 cutting-edge approaches to uncover the fault vulnerabilities and reliability challenges of hardware accelerators and software powering today's AI systems.

Section II presents a reliability assessment for the design space exploration of reliability-performance trade-offs in CNNs deployed on Reconfigurable Systolic Array (RSA) accelerators. With a comprehensive reliability and performance evaluation, the section proposes a reconfiguration strategy for a layer-wise reconfiguration of RSAs to balance the reliability and performance of an inference.

Section III analyses the effects on fault characterization accuracy and performance trade-offs across four fault assessment strategies applied on DPU-based AI accelerators for two number formats FP and Posit: *i)* low-level micro-architectural logic simulation, *ii)* emulation, *iii)* structural/architectural simulation, and through *iv)* Colored Generalized Stochastic Petri Nets (CGSPNs). The evaluation of several GEMM tiles on a DPU-based array engine indicates that fault effects vary according to the accelerator's shape and the hardware reuse for computing GEMMs. Furthermore, each fault assessment strategy plays a crucial role and might support different design stages. Our results indicate that structural and CGSPN strategies may be adapted for estimations in early designs, while emulation and logic simulation strategies are more suitable when highly accurate results are required.

Finally, Section IV analyzes the scalability limitations of

979-8-3195-4235-9/26/$31.00 ©2026 IEEE

2026 IEEE 27th Latin American Test Symposium (LATS) | 979-8-3195-4235-9/26/$31.00 ©2026 IEEE | DOI: 10.1109/LATS70329.2026.11480274

Authorized licensed use limited to: Tallinn University of Technology. Downloaded on November 10,2026 at 15:17:28 UTC from IEEE Xplore. Restrictions apply.

conventional radiation testing and fault simulation for reliability assessment, and their implications for input selection in large Vision Transformers (ViTs). Existing approaches typically rely on small, randomly sampled input sets. However, due to the dense nature of ViTs, such random sampling often fails to represent the full input space and can substantially underestimate failure rates. The experimental results show that input selection plays a critical role in reliability assessment for image-classification ViTs. In particular, confidence-aware input selection strategies can cause failure rates to vary by up to an order of magnitude in both neutron-beam experiments and software fault injection. Moreover, we show a reliability analysis with software fault simulation, which reveals a strong correlation between input sensitivity and model confidence, underscoring the need for more representative input selection methodologies in ViT reliability evaluation.

II. Reliability-Aware Reconfiguration for Reconfigurable Systolic Arrays

Systolic Arrays (SAs) are widely adopted for accelerating the inference of CNNs [4]. Reconfigurable SAs (RSAs) are presented to optimize the performance and energy consumption of the systems [5]–[7]. However, the reliability of reconfigurable SAs has not been previously studied in the literature [8], [9]. This section studies CNNs deployed on layer-wise reconfigurable SAs. The objective is to evaluate the reliability of CNNs' layers against a single fault in RSA registers, through a design space exploration over different SA sizes, along with their performance and utilization. Furthermore, a resizing strategy is proposed that identifies the best SA size for each layer, considering both reliability and performance.

A. Methodology

1) The Baseline RSA Architecture: The architecture of the RSA considered in this work is presented in Figure 1. The RSA is based on the output-stationary dataflow, in which weights propagate through the SA from top to bottom, activations propagate from left to right, and partial sums (outputs) are kept stationary in the Processing Elements (PEs). Before the execution of a layer, a set of PEs is activated by the controller, and only they perform the operations of that layer of a CNN. The unused PEs are clock-gated to optimize energy consumption. The outputs of the layer are captured from the determined subset of PEs. In this work, the RSA consists of 256×256 PEs, and it can be reconfigured into four sizes: 32×32, 64×64, 128×128, and 256×256. Before an inference, the system's configuration for each layer is determined based on offline profiling, and the RSA is reconfigured at run-time with a negligible performance overhead.

2) Reliability and Performance Analysis for the RSA: To analyze the reliability of the RSA, we perform an extensive Fault Injection (FI) experiment and obtain the Architecture Vulnerability Factor (AVF) for each CNN layer on each SA configuration. AVF represents the probability that a fault in the hardware results in an application output error [10]. In the case of CNNs, AVF refers to the probability of altering the golden

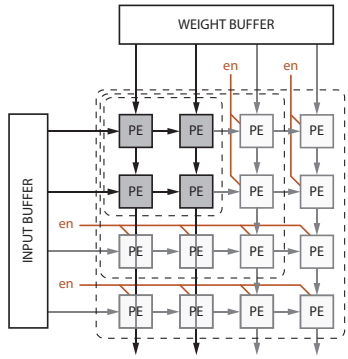


Fig. 1. Architecture of the reconfigurable systolic array

classification for an input by a fault. We consider single bitflips in the input, weight, and output registers of each active PE in each RSA configuration per inference. We perform FI for each register separately, and the AVF is the average of all FI campaigns.

The FI is performed using the reliability assessment method for CNN inference on systolic arrays based on fault propagation analysis presented in [11]. Instead of modeling a systolic array on the microarchitecture level and performing time-consuming cycle-accurate simulations, fault propagation analysis is used to calculate the resulting error in the layer output. This allows to noticeably speed up the reliability assessment. The FI is performed layer-wise based on single bitflips in the PE registers, with over 1000 injected faults per convolutional layer, ensuring 95% confidence and 5% error margin [12]. Faults are injected only in used PEs.

To analyze the performance, we profile the execution time (T) of each layer i on each SA configuration j as well as the utilization (U) of the PEs in each configuration by the CNN layer. To evaluate the performance, we consider the efficiency metric (E), as defined in Eq. (1).

$$E_{size_j}^{layer_i} = \frac{U_{size_j}^{layer_i}}{T_{size_j}^{layer_i}} \tag{1}$$

The efficiency metric combines the utilization and execution time of the SA configurations for one layer. A higher value means that an SA configuration can achieve a faster execution while using PEs as much as possible. To calculate efficiency, we normalize the execution time of all layers of a CNN to the longest execution time across all AS configurations. Also, the obtained E value is normalized to the maximum value of the layers of a CNN.

3) Resizing Strategies: To determine the RSA configuration, we first conduct an extensive layer-wise reliability and performance analysis of all aforementioned RSA configurations for each layer of the target CNNs. Based on the obtained AVF and FoM results, three strategies for layer-wise reconfiguration of the RSA are considered:

• *Constant Configuration*: We consider a constant configuration for the RSA based on the minimum (RSA32) and maximum (RSA256) available configurations, to demonstrate

Authorized licensed use limited to: Tallinn University of Technology. Downloaded on November 10,2026 at 15:17:28 UTC from IEEE Xplore. Restrictions apply.

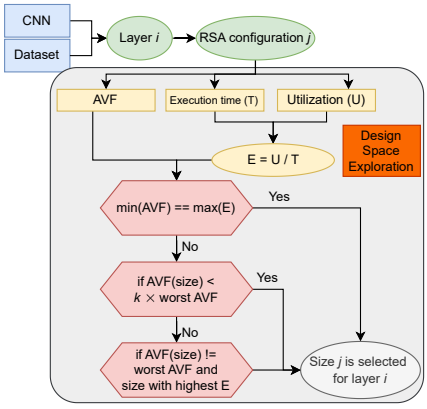


Fig. 2. Proposed algorithm to determine the RSA configuration for each layer by balancing reliability and performance.

TABLE I
SA SIZE SELECTED FOR EACH LAYER BY EACH STRATEGY FOR VGG-19

Layer	RSA32	RSA256	BR	BP	RP
1	32×32	256×256	32×32	64×64	128×128
2	32×32	256×256	64×64	64×64	64×64
3	32×32	256×256	32×32	128×128	128×128
4	32×32	256×256	128×128	128×128	128×128
5	32×32	256×256	256×256	256×256	256×256
6	32×32	256×256	256×256	256×256	256×256
7	32×32	256×256	128×128	256×256	256×256
8	32×32	256×256	32×32	256×256	256×256
9	32×32	256×256	32×32	256×256	256×256
10	32×32	256×256	32×32	256×256	256×256
11	32×32	256×256	32×32	256×256	128×128
12	32×32	256×256	64×64	256×256	256×256
13	32×32	256×256	32×32	256×256	128×128
14	32×32	256×256	32×32	256×256	256×256
15	32×32	256×256	256×256	256×256	256×256
16	32×32	256×256	64×64	256×256	256×256
Average	32×32	256×256	92×92	216×216	204×204

the advantages of the reconfigurability of RSA in terms of reliability and performance.

• *Best Reliability (BR)*: The RSA size for each layer is determined by the configuration, which implies minimum AVF.

• *Best Performance (BP)*: The RSA size for each layer is determined by the configuration, yielding the maximum performance efficiency according to Eq. (1).

• *Reliability-Performance Balance (RP)*: The RSA size for each layer is determined based on the proposed algorithm in Figure 2, to balance the reliability and performance of the SA. In this algorithm, the AVF and efficiency E for all configurations of the RSA for each layer are explored, and the RSA size for a layer is determined, which: 1) offers the best AVF and E, or 2) holds an AVF considerably lower than other sizes, or 3) contains the highest E as long as it does not carry the worst AVF.

In the experiments, we consider the $k = 0.6$ in the *RP* strategy, as a result of an exploration of the obtained AVF values for different layers and RSA configurations.

B. Experimental Results

In this subsection, we present the experimental results. First, we showcase a layer-wise comparison of reliability and performance between different RSA reconfiguration strategies for VGG-19. Afterward, we compare the obtained reliability and performance for three CNNs: VGG-19, ResNet-18, and ResNet-50, all trained on the ImageNet dataset. All networks use a 32-bit floating-point data type.

Figure 3 presents a layer-wise comparison of reliability and performance between different RSA reconfiguration strategies for VGG-19, respectively. Table I shows the RSA size selected for each layer by different strategies. The following observations can be highlighted:

• Reconfiguring the RSA for each layer provides an opportunity to optimize the reliability or performance of the CNNs deployment on systolic arrays. Comparing the results between the constant configurations (*RSA32* and *RSA256*) and the reconfigured configurations (*BR*, *BP*, and *PR*) shows gains in either reliability or performance.

• Based on the reconfiguration strategy, the RSA can provide the best reliability (*BR* strategy) or performance (*BP* strategy) in all layers. As observed, the AVF of all layers in the *BR* strategy is the minimum; however, their efficiency E is very low in many cases. Instead, in the *BP* strategy, efficiency is maximized across all layers; however, the AVF is high in many cases. This observation underscores the need to present the *RP* strategy.

• The proposed *RP* strategy yields configurations in which both reliability and performance are balanced. As observed, the AVF of *RP* in all layers is always equal to or less than that in the *BP* strategy, and the efficiency E in all layers is always equal to or more than that in the *BR* strategy.

• In terms of RSA size, it is observed that the *RP* strategy utilizes an average SA size of 204×204 PEs to balance the inference reliability and performance, which is smaller than that of *BP* and larger than that of *BR*.

Furthermore, we present the average results of AVF and normalized FoM for each CNN model to compare the impact of strategies. Figure 4 presents the results for average AVF and efficiency E, respectively, for three CNN models deployed on RSA with the five reconfiguration strategies. As observed, layer-wise reconfiguration strategies can effectively lead to a more reliable or performant deployment. On the other hand, in all cases, the *RP* strategy achieves a lower AVF (5.5% to 8.3% more reliable) than the *BP* strategy and higher performance efficiency than the *BR* strategy ($1.6\times$ up to $14.7\times$ more performant). These results highlight that *RP* can successfully balance reliability-performance in the CNN deployment on RSAs.

III. FAULT ASSESSMENT OF DPU-BASED AI HARDWARE: ACCURACY AND PERFORMANCE TRADE-OFFS IN FP AND POSIT FORMATS

This section analyzes and evaluates the advantages and constraints of four strategies for characterizing fault-rate accuracy and evaluating the performance of DPU-based array accelerators for *Floating Point* and *Posit* number formats. In particular,

Authorized licensed use limited to: Tallinn University of Technology. Downloaded on November 10,2026 at 15:17:28 UTC from IEEE Xplore. Restrictions apply.

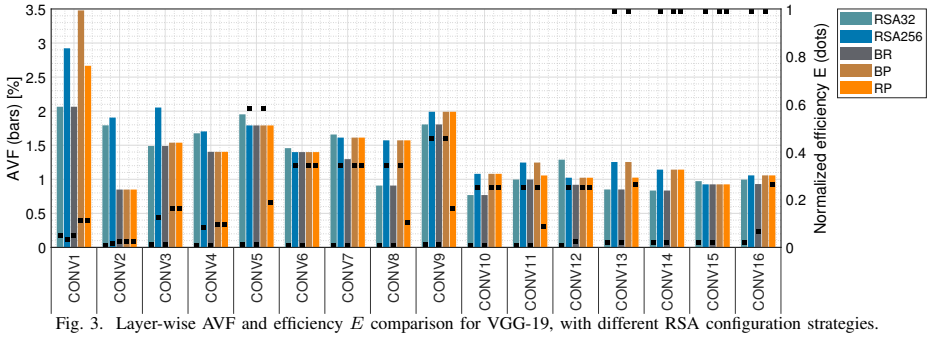


Fig. 3. Layer-wise AVF and efficiency E comparison for VGG-19, with different RSA configuration strategies.

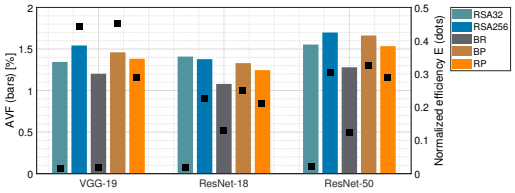


Fig. 4. Model-wise AVF and efficiency E comparison with different RSA configuration strategies.

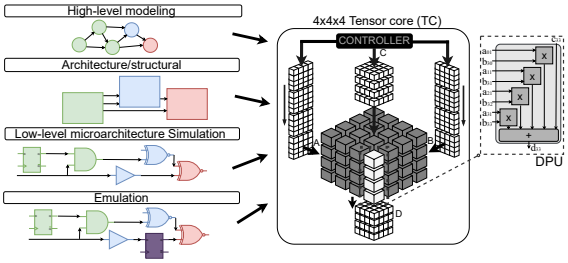


Fig. 5. A general scheme of the multi-abstraction analysis of DPU-based accelerators.

DPU-based-array cores are crucial elements commonly found in several platforms to train and deploy AI-based workloads, including *Tensor cores* (TCs) in GPUs [13], engines in NPUs [14], matrix core extensions in CPUs [15], [16], and on-chip accelerators in automotive devices [17]. Each implementation varies according to program flexibility targets, performance, and power budgets. In practice, some TCs can be configured for size and number format via software interfaces. Instead, other versions are fixed in hardware to improve performance.

In this work, we analyze the effects on fault characterization accuracy and performance trade-offs across four fault assessment strategies applied on DPU-based AI accelerators for two number formats, FP and Posit: *i)* low-level micro-architectural logic simulation, *ii)* emulation, *iii)* structural/architectural simulation, and through *iv)* Colored Generalized Stochastic Petri Nets (CGSPNs), as depicted in Figure 5.

Each abstraction level offers advantages and constraints for experimentally-based reliability analysis of soft-error effects in hardware accelerators. According to the main targets, several studies indicate that some strategies are more effective for guiding the development of fault countermeasures and hardening strategies [8], [18]–[21]. However, evaluating masking and propagation effects, as well as identifying vulnerable structures, might involve multiple evaluation strategies.

A. Main idea

We use four analysis and evaluation strategies to characterize fault-propagation impacts and, if possible, identify the most vulnerable structures in DPU-based accelerators (TCs) for two formats. In particular, we analyze the trade-offs among effectiveness, efficiency, and accuracy, as well as the challenges in identifying fault-vulnerability sources. For the evaluations, we employ equivalent DPU-based array accelerators (TCs) descriptions to assess the impact of soft errors on their computing part. In more detail, our experiments target the most susceptible structures to soft errors (*Single Event Upset* or SEUs) within the computing components of DPUs in TCs (i.e., pipeline registers and memory buffers).

B. Abstraction strategies

1) ***Simulation-based evaluation****:* relies on logic simulators and instrumentation tools that systematically modify a circuit's timing functionality to represent SEU effects. The crucial advantage of this abstraction is the high controllability for injecting faults into any low-level microarchitectural structure of the targeted circuit. Moreover, the extended observability enables focused fault-propagation analyses (i.e., internal-operation observability) and the direct identification of fault-susceptible structures. Our evaluation uses industrial-grade frameworks (Z01X by *Synopsis*) to analyze and determine the vulnerable structures in the TCs.

2) ***Emulation-based evaluation****:* uses SHADOWFI [22] to automatically integrate and instrument fault-injection structures into the gate-level descriptions of the TCs. SHADOWFI is an open-source framework that conceptualizes fault evaluation as a distributed computing task on hyperscale systems. In particular, SHADOWFI decomposes a fault-injection task into sub-tasks, which are distributed and executed across multiple computational nodes for simulation or emulation, as depicted in Figure 6. The simulation workflow distributes fault-injection tasks as multiple logic simulations on several nodes/clusters

Authorized licensed use limited to: Tallinn University of Technology. Downloaded on November 10,2026 at 15:17:28 UTC from IEEE Xplore. Restrictions apply.

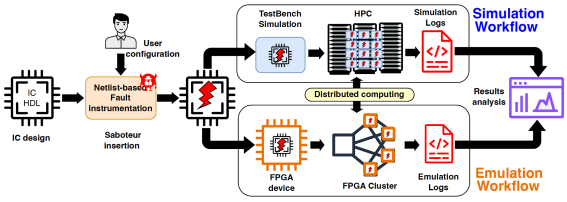


Fig. 6. General fault-injection workflow in SHADOWFI. [22]

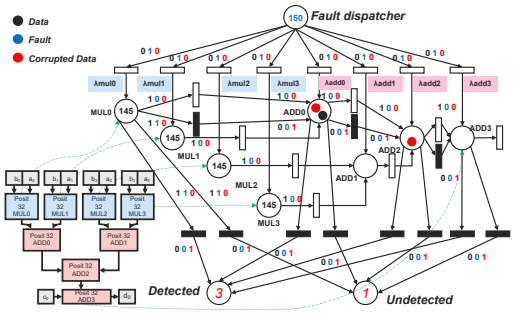


Fig. 7. CGSPN of the TC Tree Topology

within a high-performance computing system. Instead, the emulation workflow leverages FPGA reconfigurability to execute multiple fault injections across FPGA clusters. SHADOWFI employs a fault-emulation strategy via netlist instrumentation using saboteur circuits to make the fault characterization scalable across simulation and emulation workflows. The fault instrumentation first converts an input RTL into a netlist using generic cells via Yosys, and then inserts the saboteur circuits into the target components within the design. Finally, fault impacts are analyzed after evaluating the outputs from the FIs.

*3) **Structural evaluation**:* employs the EMBER framework [23], a C++-based, cycle-accurate RTL simulator that enables reliability assessment from the early design stages. EMBER combines a cycle-driven simulation kernel with RT-level accuracy, allowing efficient fault characterization. Its runtime parameter initialization supports rapid hardware configuration exploration without requiring recompilation. Additionally, EMBER provides built-in, extensible saboteur instrumentation for modeling single-event upsets (SEUs) at the circuit level.

In this work, the TCs are modeled at cycle-accurate RTL using a modular abstraction of their fundamental components. These include DPU arrays (composed of adders and multipliers), memory elements (buffers and accumulators), and support for two numerical formats: Posit32 and FP32. Each TC also generates execution traces that record cycle counts, memory traffic, and operand reuse patterns to facilitate reliability analysis. The transient faults are modeled as single random bit flips injected into operand buffers or accumulators during a *General Matrix Multiplication* (GEMM) tile execution. The Injection times and locations are uniformly sampled, and each fault-injection campaign introduces one upset per tile execution.

*4) **High-level modeling**:* relies on the CGSPN modeling strategy [24]–[27], which has shown promising results for the early analysis and evaluation of functional safety and reliability features of System-on-Chip (SoC) architectures. This strategy abstracts hardware-level granularity into a high-level graph structure and enables system characterization and fault-propagation analysis via compacted fault-injection events. The graph parameters in the model are extracted from the individual micro-architecture characterization of the fundamental structures in a SoC (e.g., Adders and multipliers for DPUs in TCs). In this work, the analysis of transient faults used timing features from the fundamental components of a SoC (e.g., ADD or MUL cores) to build the model [28].

TABLE II
TC FEATURES ACCORDING TO THEIR ABSTRACTION

Abstraction	Format	Size	Sequential cells	Execution time (s)
Emulation	FP32	388,800 cells	24,256	0.9
	Posit32	639,680 cells	20,352	0.9
Simulation	FP32	263,984 cells	45,248	1.7
	Posit32	358,309 cells	20,352	1.7
Structural	FP32	53,248 bytes	-	0.05
	Posit32	53,248 bytes	-	0.05
High-level	FP32	128 nodes	-	$< 50 \times 10^{-6}$
	Posit32	128 nodes	-	$< 50 \times 10^{-6}$

The methodology consists of the following steps: 1) Component selection: Choose components to abstract as CGSPN nodes (see Figure 7). 2) Interconnection mapping: Model links between components as edges. 3) Structural characterization: Count gates, flip-flops, and determine each component's FIT rate. 4) SET characterization: Run a single-time fault-injection campaign per component. 5) CGSPN configuration: Set deterministic transitions using structural data and SET results. 6) AVF computation: Compute TC-level AVF via CGSPN simulation. 7) Validation: Compare estimated AVF with exhaustive TC fault-injection results.

C. Experimental Results

For the experiments, we evaluated the fault vulnerabilities of 4×4×4 DPU-based TCs in two formats (*FP32* and *Posit32*) across the four abstraction levels. In detail, the description of the DPUs at RTL comprises 12-stage pipelines for the software- and emulation-based evaluations. For simulation and emulation experiments, the TCs were synthesized using the 15nm Nangate Open Cell Library. Instead, the architectural and high-level models are extracted from characterizing the DPU's fundamental components (3-stage pipeline ADD and MUL cores). The structural evaluation on EMBER [23] is based on the main operative data-path structures extracted from a typical TC core implementation on PyOpenTCU [29]. Furthermore, the CGSPN models used the functional structural organization of the TCs and extracted information from focused software-based evaluations to tune the representation parameters [26]. As inputs for each characterization, we used representative GEMM operations in ranges: ± 1.0 and ± 10.0.

In general, each TC description preserves the intended functionality of the accelerator. However, the overall size and execution performance differ among them, as depicted

Authorized licensed use limited to: Tallinn University of Technology. Downloaded on November 10,2026 at 15:17:28 UTC from IEEE Xplore. Restrictions apply.

TABLE III
AVF AND ESTIMATED AVF FOR THE FOUR EVALUATION ABSTRACTIONS ON 4X4X4 TCS

Evaluation abstraction	Number format	TC structures	Injected faults	AVF (%)
Simulation	Posit32	Overall	4,160	75.77 ± 2.3
		ADD	1,869	89.08
		MUL	2,227	66.77
	FP32	Overall	4,161	64.36 ± 2.3
		ADD	2,412	55.30
		MUL	1,748	76.31
Emulation (ShadowFI)	Posit32	Overall	162,816	26.5 ± 1.5
		ADD	73,728	29.2
		MUL	89,088	24.2
	FP32	Overall	194,048	5.3 ± 1.5
		ADD	100,864	5.2
		MUL	85,184	5.4
Structural (Ember)	Posit32	Overall	9,393	88.0 ± 3.0
	FP32	Overall	9,393	87.0 ± 8.0
High-level (CGSPN)	Posit32	Overall	150	34.0 ± 5.0
	FP32	Overall	150	32.7 ± 5.0

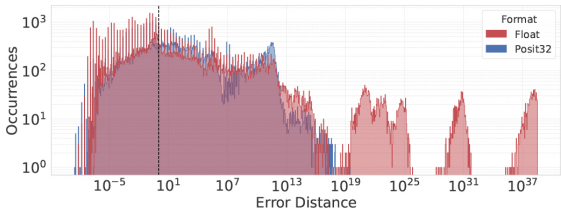


Fig. 8. Error distribution occurrences for the FP (*red*) and Posit (*blue*) Tensor Cores for the ± 1 input range.

in Table II. To evaluate the potential impacts on vulnerability analysis, we conducted several fault injection campaigns at each abstraction level to characterize overall impacts using the *Architectural Vulnerability Factor* (AVF). In more detail, Table III reports the AVF and their estimates for each abstraction, along with the AVF per TC's fundamental component, if possible. As observed, the preliminary AVF analysis indicates that Posit32 TC cores are more vulnerable than TC FP32 cores in both simulation and emulation. Clearly, differences in magnitude for both results are associated with the number of sequential structures on each abstraction. Moreover, an inverse fault-susceptibility trend is observed across the TCs: in Posit32 TCs, ADD cores are more susceptible to faults than MUL cores, while in FP32 TCs, MUL cores exhibit higher fault vulnerability than ADD cores.

In summary, simulation and emulation-based analysis allow the identification of vulnerable and critical structures within the cores. However, inconsistencies among strategies may arise, as reported in Table III, suggesting the need for complementary analyses. On the other hand, the AVF estimation on the structural TCs yielded results almost equivalent (around 87%), which can be explained by the description's focus on the cores. At these levels, memory structures, such as register files and buffers, are well identified. In contrast, the descriptions of internal structures (e.g., ADD and MUL cores) are limited to high-level operations, with minimal architectural detail.

TABLE IV
AVF AND ESTIMATED AVF (*e*AVF) FOR CGSPN ON FP & POSIT32

Architecture	FP	Posit	FP	Posit	FP	Posit	FP	Posit	FP	Posit
SET (ns)	200	300	500	600	2,405	900	10,225	1,200	24,050	1,500
Detected	1,768	6,879	2,850	5,550	3,927	18,619	4,492	15,059	4,590	18,492
Undetected	12,094	13,334	11,012	14,673	9,935	1,601	9,370	1,136	9,272	1,281
AVF (%)	12.8	34.0	20.6	27.4	28.3	92.1	32.4	93.0	33.1	93.7
eAVF (%)	11.92	32.74	20.1	27.04	26.82	90.1	31.72	92.16	31.51	92.9
AEE (%)	3.8	3.7	2.4	1.3	5.2	1.2	2.1	0.9	4.8	0.85

Despite the limitations of the structural analyses, these provide acceptable performance when evaluating the distributional impacts of faults, as depicted in Section III-C, which illustrates the error distribution of the observed propagated faults across operations in the TCs for both formats (FP32 and Posit32). We found that errors on FP32 and Posit32 TCs mostly occur within the range 10^{-7} to 10^{18}. However, large-magnitude error effects, which are more prone to critically corrupt applications, occurred in FP32 cores, indicating higher fault susceptibility than on Posit32 ones. In particular, large error effects are mostly represented as corruptions associated with the number format (i.e., corruption in the exponent fields of the FP format). In fact, the results indicate that FP's exponent field corruptions directly produce error magnitudes that are a power of two of the corrupted field in the exponent (e.g., 10^{24}, 10^{32}, and 10^{38}). Instead, the field's organization in the Posit format contributed to decreasing the magnitude of the observed errors.

For the high-level modeling, the CGSPN analysis enabled us to estimate the AVF of both the FP and Posit TC implementations using the methodology described earlier. The main challenge was preserving the TC's functional behavior while accurately capturing SET-induced fault effects and propagation. As shown in Table IV, modeling the TC with time-specific SET faults in a CGSPN yields an Absolute Estimation Error (AEE) below 5%, confirming sufficient accuracy for AVF estimation. Each CGSPN simulation required about five minutes per faulty component. Because of architectural symmetry, we only needed to inject faults into one Multiplication Unit and one of the two adders, resulting in four total simulations. This results in a total runtime of roughly 20 minutes—far faster than a full, exhaustive fault-injection campaign, which typically requires several hours. These results demonstrate that the proposed CGSPN methodology effectively supports SET-fault analysis.

D. Analysis of Inconsistencies

A complementary evaluation of *Multiple Event Upsets* (MEUs) was performed on the TCs to remove inconsistency sources and determine the most susceptible structures in FP32 and Posit32 TCs. In particular, we used SHADOWFI to evaluate both TCs across 8,000 experiments in which the targeted sequential cells and injection times were randomly selected. In detail, we evaluated the impact of MEUs injecting 2, 3, 4, 5, and 6 bit-flips per injection.

Table V reports the fault evaluations for MEU. As expected, the results show that the greater the number of corrupted bits, the greater the impact on TC operation in both FP32 and Posit32. However, Posit32 TC exhibits higher AVF sensitivity than FP32. In detail, when injecting 2 bit-flips, the TC FP32's AVF reached 10%, while the AVF for Posit32 reached 46%.

Authorized licensed use limited to: Tallinn University of Technology. Downloaded on November 10,2026 at 15:17:28 UTC from IEEE Xplore. Restrictions apply.

TABLE V
MEU AVF USING EMULATION ON BOTH TCS

TC format	Structures	MEU-2	MEU-3	MEU-4	MEU-5	MEU-6
FP32	overall	0.105	0.140	0.149	0.198	0.228
	MUL	0.109	0.129	0.153	0.170	0.188
	ADD	0.100	0.151	0.146	0.226	0.269
Posit32	overall	0.460	0.590	0.685	0.781	0.823
	MUL	0.437	0.551	0.652	0.736	0.789
	ADD	0.482	0.629	0.719	0.826	0.857

Nonetheless, when injecting 6 bit-flips, the AVF for Posit TC increases significantly (82%), whereas the AVF for PF TC barely reaches 22%. Regarding the fault-susceptibility of the internal cores, the MEU results show that, in most cases, for both TCs, ADD cores are slightly more vulnerable to fault corruption than MUL cores.

The structural placement of the ADD cores near the outputs and within the cone of influence of the primary outputs of a TC increases the likelihood that faults propagate directly to the outputs and manifest as observable errors. In contrast, faults in MUL cores may be masked by subsequent operations in the ADD cores within the TC, thereby reducing overall error propagation.

IV. INPUT SELECTION FOR DNN RADIATION TESTING

This section discusses the challenges of input selection for the reliability assessment of large DNNs. As reliability evaluation methods such as physical and software FI are inherently constrained by time and resources, careful input selection is necessary to obtain representative results.

A. The Challenge of Input Selection

Assessing the reliability of large DNNs through beam or fault simulation is becoming increasingly complex and often unfeasible due to the many combinations of dataset samples, model variants, and hardware accelerator optimizations. For example, testing the entire ImageNet validation set (50,000 samples divided into 1,000 classes) under realistic radiation conditions, and assuming we would test each sample for 1 hour of beam time, *it would take* ≈ 5.7 *years to gather enough statistical data for the ImageNet validation set*. Notably, this is a very optimistic assumption [30], [31], as testing complex devices might yield zero events even after a long period of beam testing [32]. A common strategy to manage the complexity of DNN reliability assessment is to evaluate only a small subset of random inputs [31], [33]. However, if this selection is not done carefully, the evaluation may be biased [34]. Different input subsets can yield drastically different failure rates. Some subsets may yield very few errors (optimistic scenario), while others may lead to significantly higher failure rates (pessimistic scenario). Developing improved, customized methods for beam testing DNNs is necessary to reduce the bias introduced by naive random selection.

Some works have explored input selection strategies to evaluate the robustness of DNNs, particularly in software testing and adversarial robustness [34]. These methods typically rely on criteria such as neuron coverage (i.e., which neurons are activated during input processing), surprise adequacy (distance between inputs), and uncertainty estimation from output probabilities [34]. Coverage- and distance-based techniques often have high computational costs and do not scale well to large models and datasets (e.g., transformers). In contrast, uncertainty metrics such as prediction variance or maximum probability are lightweight and effective as selection criteria [35]. We therefore use a confidence-based input selection methodology to bound DNN reliability between optimistic and pessimistic scenarios. We use the prediction probabilities that the models output in the classification task as a measure of confidence (i.e., the higher the $TOP{-}1$ probability, the higher the model's confidence in its prediction). Using the distribution of prediction confidences over the dataset, we partition inputs into *High-confidence* and *Low-confidence*. We hypothesize that *High-confidence* inputs are less likely to change their prediction under faults, whereas *Low-confidence* inputs are more prone to fault-induced misclassification.

Not all faults affect DNN outputs equally. A fault can perturb DNN execution without altering the correct classification. We refer to faults that change the classification as *Critical Silent Data Corruptions (SDCs)* (misclassifications).

B. Input Selection Approach

We propose an approach for selecting inputs for DNN radiation testing by classifying inputs based on the model's confidence. This method bounds random-input critical-failure results between a very pessimistic and a very optimistic critical SDC rate. We performed physical FIs under a neutron beam and software fault simulation (SWFS) to cross-validate it. Fig. 9 depicts our methodology divided into 3 steps.

1) Profiling: The first step of our method (step ⓐ) is to profile the dataset to gather useful data. Given a selected model, we run inference on each image of the dataset validation set. In this work, we selected the ImageNet validation dataset, which contains 50,000 images and 1,000 classes. Our approach can be extended to any vision dataset. From the validation set, we can extract whether the DNN correctly predicts the class of each input and the set of probabilities associated with each class for each input, including the $TOP1$ (final classification of the model) and $TOP2$ (second-highest classification probability). All the data is stored in a database for post-processing in the next phase.

2) Data Analysis and Input Classification: Using profiling data, we can extract different metrics to select the inputs (step ⓑ). First, we classify the input images into two classes: correct predictions and incorrect ones (the model's inference matches or differs from the ground truth). In this work, we focus only on correct predictions, as they provide the most valuable insights into the model's Critical SDCs rates. Secondly, we can compute the confidence of the correct prediction subset for each image. The confidence of input is given by the difference between the $TOP1$ probability and the $TOP2$ probability ($TOP1 - TOP2$). If the difference is close to 1, the model is very confident about its result. On the other hand, if the difference is close to 0, the model is not confident about

Authorized licensed use limited to: Tallinn University of Technology. Downloaded on November 10,2026 at 15:17:28 UTC from IEEE Xplore. Restrictions apply.

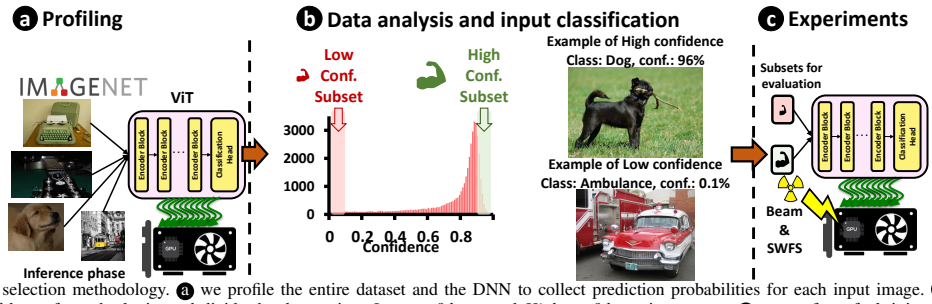


Fig. 9. Input selection methodology. (a) we profile the entire dataset and the DNN to collect prediction probabilities for each input image. (b) we compute the input confidence from the logits and divide the dataset into *Low-confidence* and *High-confidence* image sets. (c) we perform fault-injection experiments under a neutron beam and using SWFS to assess the complete dataset.

the difference between the two classes and might even be a random guess ($TOP1$ and $TOP2$ are very similar).

Figures 10b and 10c show the confidence distribution extracted for ViT and two examples of inputs extracted from the *High-confidence* and *Low-confidence* subsets. The horizontal axis shows the confidence, and the left vertical axis shows the observed frequency on the ImageNet validation dataset. We will discuss the SDC criticality results (right vertical axis) in the next section. The confidence distributions are very similar for both DNNs evaluated in this work on ImageNet. Both models are confident in the majority of inputs. We then extract two subsets from the full ImageNet dataset: the *lowest* and the *highest* confidence levels. We hypothesize that predicted confidence will affect the model's sensitivity to faults. For a fault to cause a Critical SDC (misclassification) on an input that the model is very confident in is much harder than for an image that the model has low confidence (it is easier to change the $TOP1$ probability). To illustrate how the complexity of an input affects the model's confidence, we show two real examples from the ImageNet validation set using the ViT model. 9b shows examples from the *High-confidence* set, where the ViT is 96% sure that a dog is in the image, and from the *Low-confidence* set, where the ViT is not sure whether it is an old car, an ambulance, or a fire truck, resulting in a confidence of 0.1%.

3) FI Experiments: Our analysis's final step (c) is the model reliability evaluation with 1) beam experiments to assess the effectiveness of our input selection method, and 2) SWFS to assess the complete dataset and cross-validate our method. In this step, we perform beam experiments to determine the failure rates of the different subsets (*High-/Low-confidence*) we extracted in step (b). In the last phase, we perform SWFS across the entire dataset to reproduce the impact of neutron-induced faults at the software level. This allows us to obtain a general correlation between the critical SDC rates and the confidence for all inputs. To ensure our fault simulations reflect realistic hardware behavior, we derive our fault model directly from beam experiments using a dedicated microbenchmarking procedure.

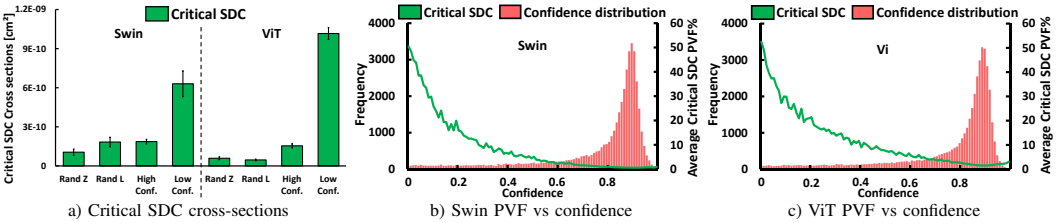
4) Evaluated DNNs and Inputs: We evaluated 2 ViT models from the HuggingFace library (v0.8.19) [36]: ViT-B-224, denoted as ViT [37], and Swin-B-W7-224, denoted as Swin [38]. ViT and Swin achieve accuracies of 84.50% and 85.20%, respectively. For the experiments, we used Python and PyTorch v2.0.0 to load the ViT and perform inferences on a batch of images from the ImageNet dataset [39].

We compare the proposed approach with the common random input selection used in previous works [31], [33]. The Random subsets were selected by setting two different random seeds. The first is with `seed = 0`, denoted Rand Z, and the second is with a large seed (`seed = 29052001`), denoted Rand L. For the neutron beam experiments, we tested four configurations for each model. We build batches of 32 images for each subset: 1) *Low-confidence* input sets are built with the 32 lowest confidence predictions; 2) *High-confidence* with the 32 highest confidence predictions; 3) *Inputs of random batches are picked among the whole ImageNet validation set*. It is worth noting that the *High-confidence* and *Low-confidence* sets include only correctly predicted inputs, as our goal is to identify inputs that may be more or less vulnerable under fault conditions. In contrast, the Random subsets include both correctly and incorrectly predicted inputs, following common practice in prior state-of-the-art DNN reliability studies [31], [33]. This distinction does not affect our conclusions, since a Critical SDC is defined as a fault-induced change in the model's $TOP1$ prediction relative to the fault-free execution, rather than relative to the ground-truth label.

5) Physical FI at ChipIr: We performed physical FI at the ChipIr facility of the Rutherford Appleton Laboratory, UK [40]. This facility has been designed to perform neutron-beam testing of electronic devices, providing an adequate test bench needed for the experiments. It notably allows multiple devices to be exposed simultaneously. ChipIR provides energy ranges similar to those in the atmosphere, with a neutron flux $10^9\times$ higher than terrestrial. On average, each configuration received a fluence of 3.33×10^{10} n/cm².

We performed neutron experiments on 3 NVIDIA RTX A2000 GPUs with the Ampere architecture. RTX A2000 has SECDED ECC to protect registers, caches, and GDDRs. We conducted all tests with ECC enabled. Our beam experiments focused only on GPU core errors (beam spot set to an $3.4cm \times 3.0cm$ area to avoid affecting other components).

Authorized licensed use limited to: Tallinn University of Technology. Downloaded on November 10,2026 at 15:17:28 UTC from IEEE Xplore. Restrictions apply.



a) Critical SDC cross-sections b) Swin PVF vs confidence c) ViT PVF vs confidence

Fig. 10. Critical SDC cross-sections and correlation between the confidence distribution and the average Critical SDC PVF%.

C. Software Fault Simulation

Our SWFS framework injects faults into the activations of a selected ViT layer. Because simple models such as single-bit flips are often unrealistic [20], we use fault models shown to represent how low-level hardware faults propagate through activation computations in large accelerators [41], [42].For each injection, we modify the targeted layer's activations by applying a selected relative error to the entire batch, and we repeat this over the full dataset to enable per-input correlations. For a given tensor, we corrupt a row/line or a block by multiplying each value by the chosen relative error.

We implement FI during model execution with a Python script to measure fault impact over the full dataset for each model. Injecting a single fault across all correctly predicted ImageNet validation inputs takes about 2 hours on average, which limits the campaign size. We target the Encoder Blocks of the ViT and Swin architectures, sampling 100 random faults and injecting them at three blocks of each ViT model (first, middle, last). From SWFS, we compute the Program Vulnerability Factor (PVF), i.e., the probability that a fault propagates to the application output as an SDC [43]. Overall, the campaign exceeded 1200 hours of computation.

D. Experimental Results

Beam Experiments: Fig.10a depicts the Critical SDC cross-sections (y-axis) for the assessed configurations (model and input subset) shown on the x-axis. For each configuration, the cross-section is computed as the number of observed Critical SDCs divided by the neutron fluence. This cross-section shows the probability of a Critical SDC per circuit area (the likelihood of a Critical SDC per area). Values are reported with 95% confidence intervals [44].

The *Low-confidence* subsets are the most vulnerable configurations, and generate the highest Critical SDC cross-sections. The average Critical SDC cross-section of *Low-confidence* sets is 8.23×10^{-10}. Contrarily, random and *High-confidence* configurations show an average Critical SDC cross-section of 1.7×10^{-10}. These results support our hypothesis, even a small error that propagates through the DNN can change the probability of the $TOP1$ class for inputs the model is not confident in, leading to a Critical SDC. Note that *High-confidence* set exhibits a similar Critical SDC rate to random sets. For the Swin model, the Critical SDC cross-section for the Random subsets is on average 1.45×10^{-10} and 1.87×10^{-10} for *High-confidence* subset. A similar result is found for ViT, the Random subsets have an average cross-section of 5.25×10^{-11} and the *High-confidence* has a cross-section of 1.54×10^{-10}. This further confirms the idea that selecting random inputs leads to cross-sections closer to an optimistic result. Finally, in our beam experiments, including initially misclassified inputs in the Random subsets did not affect the measured Critical SDC cross-sections, as no fault-induced changes in the $TOP1$ prediction were observed for these inputs relative to their fault-free execution.

Software Fault Simulation: To complement the beam experiments, we performed SWFS and focused on the occurrence of Critical SDCs across different input confidence levels. Figures 10b and 10c show the correlation between the input confidence and Critical SDC rates. The x-axis represents the input confidence values, from 0 to 1. We partition the full confidence range into 100 equal-sized bins. The left y-axis shows the frequency for each confidence bin across the dataset. The right y-axis corresponds to the average critical SDC PVF per confidence bin (i.e., the number of critical SDCs for a given bin divided by the number of injected faults).

Similarly to our beam experiment results, on average, Swin has shown to be more robust to critical faults, with lower Critical SDC rates, 2.70% on average, compared to 4.47% for ViT. The maximum Critical SDC rates are observed for the lowest-confidence bin (0–0.01), reaching 52.41% for ViT and 50.43% for Swin. At the other extreme, Swin also achieves the lowest Critical SDC rate, 0.60% for the 0.89–0.90 confidence bin, compared to 1.53% for ViT in the 0.90–0.91 bin. Overall, both DNNs show a strong inverse correlation between input confidence and Critical SDC PVF%. As confidence rises, the likelihood that a fault causes a misclassification drops sharply.

Discussion: In neutron beam experiments, *Low-confidence* subsets result in 7.79× more Critical SDCs than Random subsets on average, and can increase to 22.13× in the case of ViT with Rand L configuration. Conversely, the *High-confidence* subset only results in 2.19× more Critical SDCs than Random subsets. These results demonstrate that random input selection can severely underestimate DNN vulnerability. SWFS corroborate these findings. The *Low-confidence* subsets consistently show the highest Critical SDC rates, averaging 51.30% across models, compared to just 1.49% for *High-confidence*. Swin shows the largest disparity, 0.52% for *High-confidence* versus 51.75% for *Low-confidence*. Again, random configurations resemble the *High-confidence* case, further confirming their optimistic bias.

Authorized licensed use limited to: Tallinn University of Technology. Downloaded on November 10,2026 at 15:17:28 UTC from IEEE Xplore. Restrictions apply.

V. Conclusions

This work describes three approaches to understand the fault vulnerabilities and reliability challenges of hardware accelerators and software powering today's AI systems.

First, it presents a comprehensive design space exploration of reliability-performance trade-offs for Reconfigurable Systolic Array (RSA) CNN accelerators. In this regard, it presents a reconfiguration strategy (*RP*) for layer-wise reconfiguration of RSAs to balance reliability and performance during inference. It demonstrates that *RP* achieves 5.5% to 8.3% higher reliability than the best-performing reconfiguration strategy and $1.6\times$ to $14.7\times$ higher efficiency than the most reliable reconfiguration strategy.

Furthermore, the experiments across several abstraction levels indicate that each analysis level provides information that can guide design refinements or the development of fault-mitigation strategies. Although they examine the same circuit, the different abstractions capture distinct aspects of its structure and behavior, thereby offering complementary insights.

Finally, the third analysis, based on neutron beam and SWFS experiments, shows that model confidence strongly influences failure rates. Low-confidence inputs are significantly more vulnerable to hardware faults than higher-confidence inputs. This input-dependent sensitivity highlights the need for systematic input analysis prior to reliability assessment.

Acknowledgment

This paper is supported in part by EU Grants #101160182 TAICHIP and #101194287 NexTArc, the Estonian grant PUT PRG1467 CRASHLESS and CoE TK202 Universum.

References

[1] B. Dally, " Hardware for Deep Learning ," in *IEEE Hot Chips 35 Symp. (HCS)*, 2023, pp. 1–58.

[2] A. S. Rajasekaran, F. Al-Turjman *et al.*, *AIoT: Artificial Intelligence of Things*. CRC Press, 2025.

[3] A. Pandharipande *et al.*, "Sensing and machine learning for automotive perception: a review," *IEEE Sensors Journal*, vol. 23, no. 11, pp. 11 097–11 115, 2023.

[4] Y. Chen *et al.*, "A survey of accelerator architectures for deep neural networks," *Engineering*, vol. 6, no. 3, pp. 264–274, 2020.

[5] S. Yin *et al.*, "A high energy efficient reconfigurable hybrid neural network processor for deep learning applications," *IEEE Journal of Solid-State Circuits*, vol. 53, pp. 968–982, 4 2018.

[6] C. Peltekis *et al.*, "ArrayFlex: A systolic array architecture with configurable transparent pipelining," in *DATE*, 2023.

[7] C.-J. Lee *et al.*, "ReSA: reconfigurable systolic array for multiple tiny dnn tensors," *ACM T ARCHIT CODE OP*, vol. 37, p. 24, 9 2024.

[8] 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.

[9] ——, "Enhancing fault resilience of qnns by selective neuron splitting," in *IEEE 5th Int. Conf. on Artificial Intelligence Circuits and Systems (AICAS)*. IEEE, 2023, pp. 1–5.

[10] S. S. Mukherjee *et al.*, "A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor," *IEEE MICRO*, pp. 29–40, 2003.

[11] N. Cherezova *et al.*, "FORTALESA: Fault-tolerant reconfigurable systolic array for DNN inference," *Microprocessors and Microsystems*, vol. 119, p. 105222, 12 2025.

[12] R. Leveugle *et al.*, "Statistical fault injection: Quantified error and confidence," in *DATE*, 2009, pp. 502–506.

[13] J. Choquette *et al.*, "Nvidia a100 tensor core gpu: Performance and innovation," *IEEE Micro*, vol. 41, no. 2, pp. 29–35, 2021.

[14] E. Feng *et al.*, "snpu: Trusted execution environments on integrated npus," in *ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, 2024, pp. 708–723.

[15] H. Kim *et al.*, "Exploiting intel advanced matrix extensions (amx) for large language model inference," *IEEE Comput. Archit. Lett*, vol. 23, no. 1, pp. 117–120, 2024.

[16] E. Cui *et al.*, "Risc-v instruction set architecture extensions: A survey," *IEEE Access*, vol. 11, pp. 24 696–24 711, 2023.

[17] E. Talpes and athers, "Compute solution for tesla's full self-driving computer," *IEEE Micro*, vol. 40, no. 2, pp. 25–35, 2020.

[18] D. Gnad *et al.*, "Reliability and security of ai hardware," in *IEEE European Test Symp. (ETS)*, 2024, pp. 1–10.

[19] L. Yang *et al.*, "Gpu reliability assessment: insights across the abstraction layers," in *Int Conf. on Cluster Computing (CLUSTER)*, 2024, pp. 1–13.

[20] G. Papadimitriou and D. Gizopoulos, "Demystifying the system vulnerability stack: transient fault effects across the layers," in *ISCA*, 2021.

[21] O. Chatzopoulos *et al.*, "Measuring the unmeasurable: Demystifying silent data corruptions in ai accelerators through microarchitectural modeling," *IEEE Micro*, pp. 1–11, 2025.

[22] J.-D. Guerrero-Balaguera *et al.*, "Shadowfi: An open-source framework for fault evaluation of complex ic designs using hyperscale computing," *IEEE Access*, vol. 13, pp. 211 382–211 406, 2025.

[23] A. Veronesi *et al.*, "Ember: A cycle-based framework for early-stage reliability assessment in parametric rtl designs," in *34th Asian Test Symp. & The 9th Int. Test Conf. Asia (ATS/ITC-Asia 2025)*, 2025, pp. 1–6.

[24] E. C. V. Castillo *et al.*, "An efficient approach for stls development of automotive socs using colored petri nets," in *DDECS*, 2024, pp. 98–103.

[25] E. C. V. Castillo, F. A. Da Silva *et al.*, "Diagnostic coverage estimation for automotive socs based on colored stochastic petri nets," in *VLSI-SoC*, 2024, pp. 1–6.

[26] E. C. V. Castillo, J.-D. Guerrero-Balaguera *et al.*, "Early reliability estimation in hardware accelerators using improved colored petri nets," in *ITC*, 2025, pp. 498–501.

[27] E. C. V. Castillo *et al.*, "A predictive framework for fault vulnerabilities in edge hardware accelerators," in *LASCAS*, 2026.

[28] M. Cannon *et al.*, "Multiscale system modeling of single-event-induced faults in advanced node processors," *IEEE Trans. on Nuclear Science*, vol. 68, no. 5, pp. 980–990, 2021.

[29] R. L. Sierra *et al.*, "Analyzing the impact of different real number formats on the structural reliability of tcus in gpus," in *VLSI-SoC*, 2023, pp. 1–6.

[30] I. Rodriguez-Ferrandez *et al.*, "Proton evaluation of single event effects in the nvidia gpu orin som: Understanding radiation vulnerabilities beyond the soc," in *IEEE IOLTS*, 2024, pp. 149–155.

[31] P. Rech, "Artificial neural networks for space and safety-critical applications: Reliability issues and potential solutions," *IEEE Trans. Nucl. Sci.*, vol. 71, no. 4, pp. 377–404, 2024.

[32] H. Quinn *et al.*, "Measuring zero: Neutron testing of modern digital electronics," *IEEE Trans. Nucl. Sci.*, vol. 71, no. 4, pp. 670–679, 2024.

[33] J. M. Badia *et al.*, "Reliability of vision transformers and cnns on edge ai systems under neutron radiation," *IEEE Trans. Nucl. Sci.*, vol. 72, no. 8, pp. 2706–2716, 2025.

[34] S. Wang *et al.*, "A survey on test input selection and prioritization for deep neural networks," in *ISSSR*, 2024, pp. 232–243.

[35] W. Ma *et al.*, "Test selection for deep learning systems," *ACM Trans. Softw. Eng. Methodol.*, vol. 30, no. 2, pp. 1–22, 2021.

[36] R. Wightman, "Huggingface," huggingface.co/timm.

[37] A. Dosovitskiy *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," in *ICLR*, 2021, pp. 1–21.

[38] Z. Liu *et al.*, "Swin transformer: Hierarchical vision transformer using shifted windows," in *IEEE/CVF ICCV*, 2021, pp. 9992–10 002.

[39] J. Deng *et al.*, "ImageNet: A large-scale hierarchical image database," in *IEEE CVPR*, 2009, pp. 248–255.

[40] C. Cazzaniga *et al.*, "Fast neutron measurements for the characterization of the chipir beamline," *IEEE TNS*, vol. 71, no. 8, pp. 1520–1526, 2024.

[41] Y. Sun *et al.*, "Demystifying the resilience of large language model inference: An end-to-end perspective," in *SC*, 2025, p. 1127–1144.

[42] C. Bolchini *et al.*, "Fast and accurate error simulation for cnns against soft errors," *IEEE TC*, vol. 72, no. 4, pp. 984–997, 2022.

[43] V. Sridharan and D. Kaeli, "Eliminating microarchitectural dependency from architectural vulnerability," in *IEEE HPCA*, 2009, pp. 117–128.

[44] H. Quinn, "Challenges in testing complex systems," *IEEE Trans. Nucl. Sci.*, vol. 61, no. 2, pp. 766–786, 2014.

Authorized licensed use limited to: Tallinn University of Technology. Downloaded on November 10,2026 at 15:17:28 UTC from IEEE Xplore. Restrictions apply.

Appendix 7

VII

N. Cherezova, A. Jutman, and M. Jenihhin. FORTALESA: Fault-tolerant reconfigurable systolic array for DNN inference. *Microprocessors and Microsystems*, 119:105222, 12 2025

Microprocessors and Microsystems 119 (2025) 105222

Contents lists available at ScienceDirect

Microprocessors and Microsystems

journal homepage: www.elsevier.com/locate/micpro

FORTALESA: Fault-tolerant reconfigurable systolic array for DNN inference ☆

Natalia Cherezova [a,*], Artur Jutman [a,b], Maksim Jenihhin [a]

[a] Department of Computer Systems, Tallinn University of Technology, Tallinn, Estonia
[b] Testonica Lab, Tallinn, Estonia

ARTICLE INFO

Keywords:
Systolic array
Fault tolerance
Deep neural networks
Accelerator

ABSTRACT

The emergence of Deep Neural Networks (DNNs) in mission- and safety-critical applications brings their reliability to the front. High performance demands of DNNs require the use of specialized hardware accelerators. Systolic array architecture is widely used in DNN accelerators due to its parallelism and regular structure. This work presents a run-time reconfigurable systolic array architecture with three execution modes and four implementation options. All four implementations are evaluated in terms of resource utilization, throughput, and fault tolerance improvement. The proposed architecture is used for reliability enhancement of DNN inference on systolic array through heterogeneous mapping of different network layers to different execution modes. The approach is supported by a novel reliability assessment method based on fault propagation analysis. It is used for the exploration of the appropriate execution mode-layer mapping for DNN inference. The proposed architecture efficiently protects registers and MAC units of systolic array PEs from transient and permanent faults. The reconfigurability feature enables a speedup of up to 3×, depending on layer vulnerability. Furthermore, it requires 6× fewer resources compared to static redundancy and 2.5× fewer resources compared to the previously proposed solution for transient faults.

1. Introduction

Recent years have been marked by a tremendous increase in adoption of Machine Learning (ML) algorithms, specifically Deep Neural Networks (DNNs), and their employment in mission- and safety-critical application domains, such as automotive, avionics, and space. The computational complexity of DNN models requires the use of specialized hardware accelerators. Many DNN accelerators are based on systolic array architecture, including Google TPU [1], MIT Eyeriss [2], and Berkeley Gemmini [3]. However, as nanoelectronics used in hardware accelerators are prone to faults caused by radiation and aging, the concern about their reliability comes to the front [4].

Furthermore, with the current need for self-aware computing systems that can monitor their state and adapt to changing environments and operating conditions, an important property of hardware design is the ability to adjust or reconfigure to changing conditions for improved availability. To address those needs, we propose a run-time reconfigurable systolic array architecture that tackles the balance between reliability and performance through different execution modes. As a target application, we consider the inference of DNNs. Since different DNN components have different vulnerability levels, they might benefit from heterogeneous mapping to different execution modes of the proposed systolic array. We also introduce a novel reliability assessment method for systolic arrays. It is used to determine the appropriate execution mode-layer mapping and consequently to validate the effectiveness of the proposed architecture.

The contributions of the paper are

- a systolic array architecture with three execution modes (FORTALESA) for run-time reconfigurable redundancy and flexible reliability vs. performance trade-off;
- hardware implementation and evaluation of the proposed architecture;
- a novel reliability assessment method for DNN inference on a systolic array based on fault propagation analysis;
- a flexible DNN protection methodology balancing reliability and performance when executed on a systolic array.

For DNN protection, different network layers are mapped to appropriate execution modes of the systolic array based on their vulnerability. The proposed reliability assessment method is used for the exploration of the execution mode–layer mapping.

The proposed architecture efficiently protects both sequential (registers) and combinational components of systolic array processing elements. While the main objective is to address transient faults (soft

☆ This article is part of a Special issue entitled: 'DTTIS 2024' published in Microprocessors and Microsystems.

* Corresponding author.
E-mail addresses: natalia.cherezova@taltech.ee (N. Cherezova), artur@testonica.com (A. Jutman), maksim.jenihhin@taltech.ee (M. Jenihhin).

https://doi.org/10.1016/j.micpro.2025.105222
Received 25 June 2025; Received in revised form 19 September 2025; Accepted 23 October 2025
Available online 29 October 2025
0141-9331/© 2025 The Author(s). Published by Elsevier B.V. This is an open access article under the CC BY-NC license (http://creativecommons.org/licenses/by-nc/4.0/).

errors), the proposed architecture can handle permanent faults as well, allowing for graceful degradation. The reconfigurability feature of FORTALESA enables a speedup of up to 3×, depending on layer vulnerability, and requires 6× fewer resources compared to static redundancy and 2.5× fewer resources than the previously proposed solution for transient faults.

The rest of the paper is organized the following way. Section 2 gives an overview of related works. Section 3 presents the background on systolic array architecture. Section 4 introduces the architecture of the proposed reconfigurable systolic array. Section 5 explains the proposed reliability assessment method for systolic arrays based on fault propagation. Section 6 evaluates the effectiveness of the proposed methodology for DNN reliability, and Section 7 concludes the paper.

2. Related works

A reconfigurable redundancy approach is common in multicore processors, e.g., ARM's Triple Core Lock-Step (TCLS) system [5] and RISC-V based multicore processing cluster with on-demand redundancy for efficient performance vs. reliability trade-off [6]. Such solutions allow the use of individual cores separately to run different tasks for high performance or as a group to run the same task for high reliability. Execution mode can be reconfigured during run-time, thus allowing the computing system to adapt to changing conditions and task requirements.

While there are several ways to ensure the reliability of DNN inference, the most common approaches imply redundancy, fault-aware training, and algorithm-based fault tolerance (ABFT). Redundancy indicates replication of the components working in parallel (spatial redundancy) or re-execution of the same operation (temporal redundancy). Since redundancy incurs high overhead, the goal is to find a trade-off between reliability and performance by identifying and protecting more vulnerable components. To tackle this challenge, prior works have proposed selective replication of the layers [7,8], of the channels within a layer [9], of the weights [10] or neurons [11]. Another common approach requires re-training of the DNN with the added faults to improve the resilience of the network. Fault-aware training was used to address faults in FPGAs [12], timing [13] and computational [14] errors in accelerators. ABFT methods consider certain features of the application for fault detection and correction. Proposed solutions include checksums [15,16], bounded activation functions [17], restricted output ranges [18] and quantile shifts [19], and AN codes [20].

Systolic arrays were first introduced in the 1980s [21] and recently became popular for accelerating DNNs. As the reliability of DNN accelerators comes to the front, recent works have proposed methods for the assessment and enhancement of systolic arrays. [22] analyzed the effects of transient faults in the combinational logic of systolic arrays. [23] presented Saca-FI, a microarchitecture-level fault injection framework for systolic array-based CNN accelerators, for the analysis of transient faults in registers. Saca-FI is based on SCALE-Sim, a cycle-accurate simulator for DNN inference on systolic arrays [24]. [25] proposed an RTL-level fault injection framework for the analysis of stuck-at faults within multiply and accumulate (MAC) units. [26] presented a fault injection framework that models systolic array dataflow using Uniform Recurrent Equations. Unlike previous works, the proposed reliability assessment method is based on fault propagation analysis. Instead of injecting faults in the microarchitectural or RTL model of a systolic array, which is a very time-consuming task; errors are added directly to the DNN layer output using fault propagation analysis that allows to significantly speed up the fault injection process.

Reliability enhancement methods for systolic arrays, similarly to DNNs, include ABFT and network retraining. [27] presented a permanent fault correction technique by introducing a re-computing unit that recalculates the computations assigned to faulty processing elements. [28] addressed permanent faults through bypassing faulty units and assigning their computations to the redundant ones. Similarly, [29] used pruning of faulty processing elements for permanent fault correction. [30] proposed a fault-aware pruning plus retraining (FAP+T) technique to address manufacturing defects in the systolic array-based accelerators. It requires modification of the target DNN architecture through pruning and retraining to adapt it to a specific defect in the chip. [31] presented an ABFT technique for permanent fault detection in systolic arrays on FPGAs. Permanent faults in configuration memory are considered in that work. Another permanent fault detection approach assuming faults in registers and MAC units is presented in [32]. The approach is based on checksum calculations and was implemented on FPGA as well. [33] presented a fault tolerance solution for an FPGA-based systolic array accelerator. Faults in configuration memory affecting the systolic array datapath are detected using checksum values and corrected using dynamic partial reconfiguration of the FPGA. [34,35] presented an ABFT methodology specifically developed for sparse systolic arrays, considering their unique characteristics. While most works address permanent faults, [36] proposed an ABFT method for detecting timing faults due to reduced voltage. Several works proposed test architectures for permanent and transition-delay fault detection in low-power accelerators based on systolic arrays [37,38]. Compared to the aforementioned works that mostly address detection of permanent faults, the proposed reconfigurable systolic array architecture addresses both permanent and transient faults, enables reliability-performance trade-off, provides fault correction in both fault-tolerant execution modes, and does not require retraining of a DNN for each particular chip.

Reconfigurable DNN accelerators based on a systolic array have been proposed before: to improve energy efficiency [39], to decrease inference latency [40] and power consumption [41], to boost transformer models performance [42]. In contrast to the previous works, the proposed reconfigurable systolic array architecture improves reliability of the DNN inference while balancing performance.

3. Background

3.1. Systolic array architecture

Systolic array has a regular structure consisting of an array of Processing Elements (PEs). Each PE consists of a Multiply-Accumulate (MAC) unit and registers to hold the intermediate values. Memory buffers for activations and weights are located at the top and left sides of the systolic array respectively. Commonly, weights are fed vertically from top to bottom, thus each column is assigned to one output channel, and activations (inputs) are fed from left to right, hence each row processes values from the same sliding windows. Dataflow type defines the internal structure of PEs and the movement of data through the array. Dataflow types include weight-stationary (WS), input-stationary (IS), and output-stationary (OS) [43].

In the weight-stationary dataflow, the weight matrix is first pre-loaded into a systolic array by loading one row per cycle, then the input matrix propagates through the array from left to right. The outputs (results of MAC operation) propagate from top to bottom. In this scenario, weights are stationary and are kept in the systolic array during matrix multiplication execution. This dataflow maximizes the reuse of the weights.

In the input-stationary dataflow, activations are pre-loaded in the systolic array similarly to WS and kept in the systolic array during the matrix multiplication process. The data movement and structure of the systolic array are similar to WS dataflow.

In the output-stationary dataflow, outputs are kept in the PEs of the systolic array, while weights and activations are streaming through the systolic array. Weights are streaming from top to bottom, and activations (inputs) are streaming from left to right. Outputs are accumulated inside PEs. This dataflow type does not need a pre-loading stage.

An **output-stationary systolic array** is considered for this work. It is commonly used and allows high inputs and outputs reuse [44,45] and, on average, has higher performance [40]. Previous works have shown that it is more fault-tolerant [46].

3.2. Convolution on systolic array

The convolution operation is mapped to the systolic array by transforming it to matrix multiplication using im2col algorithm [47]. An input tensor is transformed into a $(H_{out} \cdot W_{out}) \times (H_k \cdot W_k \cdot C_{out})$ matrix and weights are transformed into a $(H_k \cdot W_k \cdot C_{in}) \times C_{out}$ matrix, where H_{out} and W_{out} are the height and width of the output tensor, H_k and W_k are the height and width of a kernel, C_{in} and C_{out} are the number of channels in the input and output tensor correspondingly. When inputs and weights are represented this way, each column of the output-stationary systolic array calculates values for one output channel, and each row calculates values of the same pixel across all output channels.

If the input matrices exceed the size of the systolic array, matrix multiplication is performed in tiles corresponding to the size of the systolic array grid.

The latency L of a matrix multiplication on a systolic array is calculated as

$$L_{SA} = M + N - 1 + N - 1 = M + 2N - 2\,, \tag{1}$$

where $N \times N$ is the size of the systolic array and $N \times M$ is the size of the matrices [24].

The number of tiles in each matrix is calculated as

$$T_a = \left\lceil \frac{P}{N} \right\rceil, \tag{2}$$

$$T_w = \left\lceil \frac{K}{N} \right\rceil, \tag{3}$$

where T_a and T_w are the number of tiles in activations and weights correspondingly, P is the size of the reshaped activations, and K is the number of output channels and the size of the weights matrix. The total number of steps needed to calculate the output equals $S = T_a \cdot T_w$, and the total latency of matrix multiplication on the systolic array is therefore

$$L_{total} = S \cdot L_{SA} = \left\lceil \frac{P}{N} \right\rceil \cdot \left\lceil \frac{K}{N} \right\rceil \cdot (M + 2N - 2). \tag{4}$$

4. Fault-tolerant reconfigurable systolic array

We propose FORTALESA, a systolic array with reconfigurable redundancy that provides flexible reliability–performance trade-off for critical tasks requiring fault tolerance and non-critical ones that benefit from increased parallelism.

The proposed architecture supports three **execution modes**:

- no redundancy for increased performance;
- dual-redundancy grouping (DRG) for reliability–performance trade-off;
- triple-redundancy grouping (TRG) for full protection.

The selected execution mode is configured through a control signal that controls multiplexers. The control signal comes from the host processor based on the task requirement. Since different layers of DNNs have different vulnerability, it is possible to execute different layers in different modes, thus achieving an efficient trade-off between performance and reliability.

The proposed architecture addresses faults in registers and arithmetic units of systolic array PEs, since they lead to silent data corruption without indication of an error, unlike faults in control logic that usually result in functional interrupts that signal that there was a fault and require a reset.

DRG execution mode. In the DRG execution mode, two neighboring PEs form a group (Fig. 1). They receive the same inputs, one of them acts as the main PE, the other as a shadow. Main PE compares two partial sums (OREG) and adjusts its partial sum value without re-execution using one of the possible fault correction techniques: (a) averaging of two values (DRGA) or (b) setting faulty bits to zero (DRG0). While dual redundancy does not enable full protection, these correction techniques decrease the effect of the fault on the output values, thus enabling a reliability–performance trade-off. The choice of the fault correction technique is a design-time parameter that defines two implementation options for the DRG execution mode.

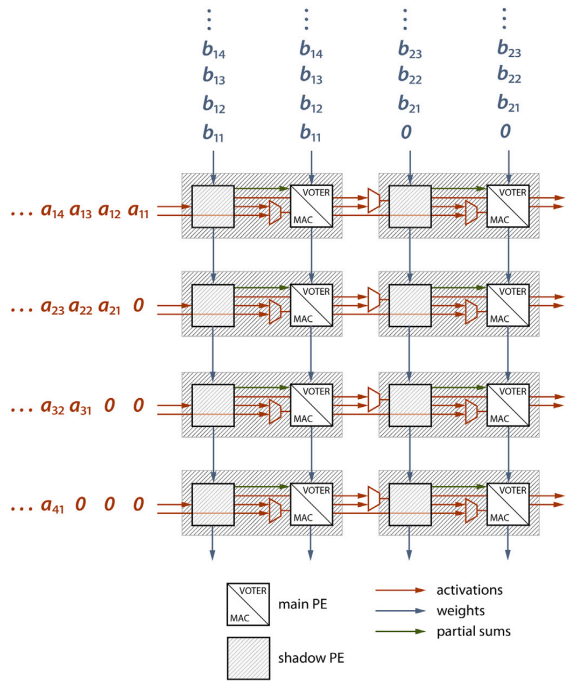


Fig. 1. Arrangement of the systolic array in the dual-redundancy grouping execution mode (DRGA or DRG0).

Additional connections are introduced to support redundant execution modes. To ensure that faulty IREG or WREG values do not propagate through the array, each PE passes the activation to the next PE of the same type, skipping one in between. While faults in buffers and interconnects are not in the focus of this work, such interleaved connections can help protect against those faults as well. During the fault-tolerant execution modes, there are two copies of input data traversing through the systolic array. The shadow PE also passes the value of the OREG (partial sum) to the main PE for comparison. Main PE has additional functionality for fault correction that works in parallel with the MAC unit. Since the updated partial sum is available on the next clock cycle, the latency of the systolic array working in DRG execution mode equals the latency of the $N \times N/2$ systolic array plus 1:

$$L_{SA}^{drg} = M + N - 1 + \frac{N}{2} - 1 + 1 = M + \frac{3N}{2} - 1\,. \tag{5}$$

The total latency is then

$$L_{total}^{drg} = \left\lceil \frac{P}{N} \right\rceil \cdot \left\lceil \frac{2K}{N} \right\rceil \cdot \left(M + \frac{3N}{2} - 1 \right). \tag{6}$$

Since in the redundancy execution modes, several PEs calculate the same values, the term **effective size** of the systolic array is introduced. It defines the number of groups calculating unique values and the size of the output matrix. The effective size of the systolic array in the DRG execution mode is $N \times \frac{N}{2}$.

TRG execution mode. For the TRG execution mode, two implementations are possible: with groups of three or four PEs. The implementation option with groups of three PEs (TRG3) is shown in Fig. 2 and the

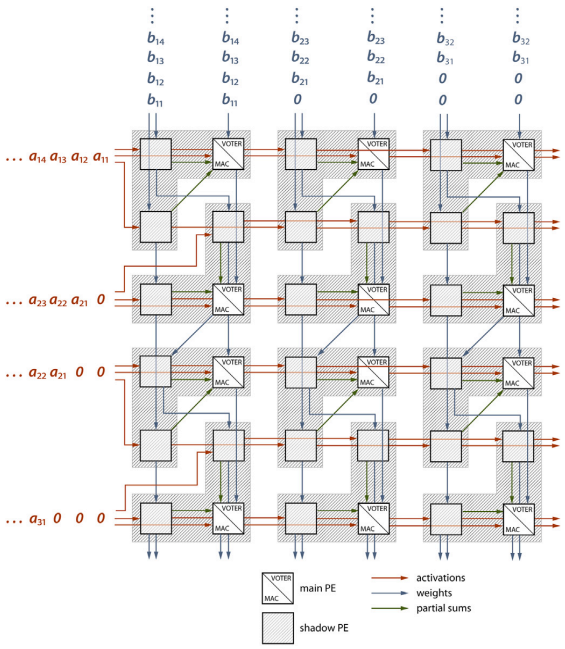


Fig. 2. Arrangement of the systolic array in the triple-redundancy grouping execution mode, implementation option with groups of three PEs (TRG3).

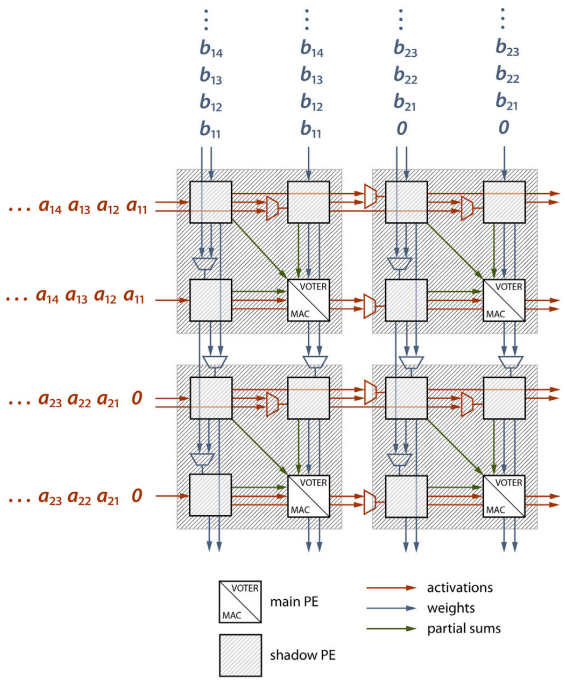


Fig. 3. Arrangement of the systolic array in the triple-redundancy grouping execution mode, implementation option with groups of four PEs (TRG4).

Table 1
Execution modes (run-time) and implementation options (design-time).

Execution mode	Implementation options	Effective size
Performance (PM)	Baseline SA	$N \times N$
DRG	DRGA DRG0	$\frac{N}{2} \times N$
TRG	TRG3 TRG4	$\frac{2N}{3} \times \frac{N}{2}$ $\frac{N}{2} \times \frac{N}{2}$

implementation option with groups of four PEs (TRG4) in Fig. 3. The group size is the design-time parameter.

In the TRG3 implementation, the voter in the main PE works in parallel with the MAC unit. In the TRG4 implementation, the MAC unit in the main PE is not used in the TRG execution mode, the main PE only compares the partial sums calculated by three shadow PEs.

The effective size of the systolic array, i.e., the size of the output matrix, for TRG3 implementation is $\frac{2N}{3} \times \frac{N}{2}$. The effective sizes of the matrices that can be multiplied are $\frac{2N}{3} \times M$ and $M \times \frac{N}{2}$. The latency of the TRG3 implementation is

$$L_{SA}^{trg3} = M + \frac{2N}{3} - 1 + \frac{N}{2} - 1 + 1 = M + \frac{7N}{6} - 1. \tag{7}$$

The total latency then

$$L_{total}^{trg3} = \left\lceil \frac{3P}{2N} \right\rceil \cdot \left\lceil \frac{2K}{N} \right\rceil \cdot \left(M + \frac{7N}{6} - 1 \right). \tag{8}$$

The effective size of the systolic array for the TRG4 implementation is $\frac{N}{2} \times \frac{N}{2}$. The effective sizes of the matrices that can be multiplied are $\frac{N}{2} \times M$ and $M \times \frac{N}{2}$. The latency of the TRG4 implementation is, therefore,

$$L_{SA}^{trg4} = M + \frac{N}{2} - 1 + \frac{N}{2} - 1 + 1 = M + N - 1. \tag{9}$$

The total latency then

$$L_{total}^{trg4} = \left\lceil \frac{2P}{N} \right\rceil \cdot \left\lceil \frac{2K}{N} \right\rceil \cdot (M + N - 1). \tag{10}$$

Since both execution modes have design-time parameters, four implementation options are possible: PM-DRG0-TRG3, PM-DRG0-TRG4, PM-DRGA-TRG3, and PM-DRGA-TRG4. Their efficiency is evaluated in Section 6. The overview of the execution modes and implementation options is given in Table 1.

The architecture of FORTALESA is based on the output-stationary dataflow; however, other dataflows can be used to build reconfigurable systolic arrays following FORTALESA methodology. Processing elements can be grouped in a similar way, with similar interleaved connections. During the pre-loading stage, PEs from one group should receive the same input or weight. The faults would be corrected by applying a selected correction technique to the output value.

5. Reliability assessment method for systolic arrays

This section presents the proposed reliability assessment method for systolic arrays. The proposed reliability assessment method combines fault injection with fault propagation analysis. Instead of modeling a systolic array on the microarchitecture level and performing time-consuming cycle-accurate simulations, fault propagation analysis is used to calculate the resulting error in the layer output. This allows to noticeable speed up the reliability assessment of DNN inference on a systolic array. The method allows to analyze both transient and permanent faults in the OS systolic arrays.

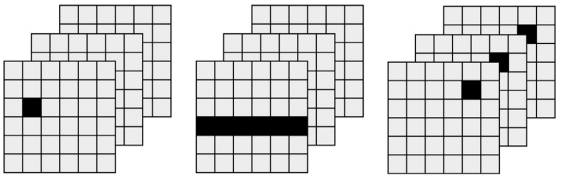


Fig. 4. Error patterns in the output tensor: (a) single point, (b) line, (c) bullet.

Table 2
Transient fault parameters.

Symbol	Description
f_{type}	Type of fault (IREG, WREG, OREG, MULT)
ts	Cycle of a systolic array execution
t_w	Weight tile
t_a	Activation tile
p_{row}	Row of the PE
p_{col}	Column of the PE
β_f	Bit index

5.1. Fault propagation in systolic arrays

Considering the structure of a single processing element, faults can occur in registers holding intermediate values or in a multiplier. Considering the regular structure of a systolic array and data movement, faults in different components produce different error patterns in the output tensor. Fig. 4 presents the possible patterns for the convolution operation on the output-stationary systolic array. Fault patterns are derived analytically and experimentally following examples from previous works for similar cases [31,48]. A bit flip in a weight register (WREG) results in the propagation of faulty weight through the whole column affecting several output values in one channel. This ends up in the *line pattern*. A bit flip in an input register (IREG) results in the propagation of faulty activation through the whole row affecting one value in several output channels. This ends up in the *bullet pattern*. A bit flip in the multiplier or an output register keeping partial sum (OREG) affects one value in one output channel, resulting in a *point pattern*.

5.2. Fault propagation analysis for transient faults

Transient faults in registers and MAC units are considered for the analysis. Each fault is defined by seven parameters given in Table 2.

An output of the convolutional layer can be expressed as

$$y_{(u,v)} = \sum_{c}^{C_{in}} \sum_{i}^{H_k} \sum_{j}^{W_k} w_{(c,i,j)} \cdot x_{(c,u+i,v+j)} + b, \tag{11}$$

where $y_{(u,v)}$ is the output value with index (u, v), C_{in} is the number of input channels, H_k and W_k are the height and width of a kernel, $w_{(c,i,j)}$ is an individual weight, $x_{(c,u+i,v+j)}$ is an activation (input), and b is a bias.

A bit flip in an input can be expressed as an addition of the error term ε to this input. The error term for a signed integer is expressed as

$$\varepsilon = 2^{\beta_f} \cdot \gamma, \tag{12}$$

where β_f is the index of a faulty bit and γ is a sign coefficient, which is derived by the following equation

$$\gamma = \begin{cases} -1 & \text{if} \quad x_f^{(\beta_f)} = 1 \wedge \beta_f \neq \beta_{sign} \\ 1 & \text{if} \quad x_f^{(\beta_f)} = 1 \wedge \beta_f = \beta_{sign} \\ -1 & \text{if} \quad x_f^{(\beta_f)} = 0 \wedge \beta_f = \beta_{sign} \\ 1 & \text{if} \quad x_f^{(\beta_f)} = 0 \wedge \beta_f \neq \beta_{sign} \end{cases} \tag{13}$$

where $x_f^{(\beta_f)}$ is the faulty bit of the affected input and β_{sign} is the index of a sign bit.

A fault in a multiplier or register holding activation, weight, or partial sum results in an addition of an error e to an output of the convolutional layer. This error e depends on the error term ε added to the faulty value and can be calculated for each type of fault from parameters given in Table 2.

Fault in the input register of a systolic array. If the fault happens in the input register, it corrupts an activation, then an error added to the output of the convolutional layer equals

$$e_{ireg} = w_{(c_f,i_f,j_f)} \cdot \varepsilon, \tag{14}$$

where $w_{(c_f,i_f,j_f)}$ is the weight multiplied with the faulty activation. This weight is found by its position (c_f, i_f, j_f):

$$c_f = \left\lfloor \frac{ts - p_{col} - p_{row}}{H_k \cdot W_k} \right\rfloor \tag{15}$$

$$k_f = (ts - p_{col} - p_{row}) \mod (H_k \cdot W_k) \tag{16}$$

$$i_f = \left\lfloor \frac{k_f}{W_k} \right\rfloor \tag{17}$$

$$j_f = k_f \mod W_k \tag{18}$$

where ts is the targeted cycle of a systolic array execution, p_{row} and p_{col} are the position of the targeted PE.

Fault results in the bullet error pattern. Therefore, the following parameters define affected output values: output channels $c_{out,f}$ and position in the feature map (u_f, v_f). Affected output channels are calculated as follows:

$$c_{out,f} = \left[c_{out,f}^{start}, c_{out,f}^{end} \right] \tag{19}$$

$$c_{out,f}^{start} = (t_w - 1) \cdot N + p_{col} + 1 \tag{20}$$

$$c_{out,f}^{end} = \begin{cases} t_w \cdot N & \text{if} \quad C_{out} \geqslant t_w \cdot N \\ C_{out} & \text{if} \quad C_{out} < t_w \cdot N \end{cases} \tag{21}$$

where $c_{out,f}$ is a range of affected output channels, starting with $c_{out,f}^{start}$ and ending with $c_{out,f}^{end}$, t_w is the targeted weight tile, p_{row} is the row of the targeted PE, and C_{out} is the total number of output channels.

The position (u_f, v_f) is found in the following way:

$$\square_f = (t_a - 1) \cdot N + p_{row} \tag{22}$$

$$u_f = \left\lfloor \frac{\square_f}{W_{out}} \right\rfloor \tag{23}$$

$$v_f = \square_f \mod W_{out} \tag{24}$$

where $\square_f$ is the affected sliding window and W_{out} is the width of the output feature map.

An error e_{ireg} calculated using Eq. (14) is added to the affected output values.

Fault in the weight register of a systolic array. An error added to the output of the convolutional layer in this case equals

$$e_{wreg} = x_{(c_f,i_f,j_f)} \cdot \varepsilon. \tag{25}$$

The sign of the ε is defined by the affected weight following Eq. (13).

Fault in the weight register results in the line error pattern, therefore, the following parameters define affected output values: output channel $c_{out,f}$ and a range of positions starting with $(u_f^{start}, v_f^{start})$ and ending with (u_f^{end}, v_f^{end}). The affected output channel is calculated as follows:

$$c_{out,f} = (t_w - 1) \cdot N + p_{col}. \tag{26}$$

Positions are calculated from the affected sliding windows following Eqs. (23) and (24):

$$\square_f^{start} = (t_a - 1) \cdot N + p_{row} \tag{27}$$

$$\square_f^{end} = \begin{cases} t_a \cdot N & \text{if} \quad W_{out} \cdot H_{out} \geqslant t_a \cdot N \\ W_{out} \cdot H_{out} & \text{if} \quad W_{out} \cdot H_{out} < t_a \cdot N \end{cases} \tag{28}$$

Table 3
Permanent fault parameters.

Symbol	Description
f_{type}	Type of fault (IREG, WREG, OREG, MULT)
p_{row}	Row of the PE
p_{col}	Column of the PE
β_f	Bit index

Faults in the output register and the multiplier. Since those faults result in a single point error pattern, an error added to the output equals

$$e_{oreg} = e_{mult} = 2^{\beta_f} . \tag{29}$$

The output channel $c_{out,f}$ is found using Eq. (26), and the position is found using Eqs. (23) and (24).

5.3. Fault propagation analysis for permanent faults

Permanent faults are present during the whole execution and therefore defined by four parameters given in Table 3.

Permanent faults result in several error patterns, one for each step. Therefore, the affected output values are defined by sets of parameters. E.g., in case of permanent faults in the input registers, the affected output values are defined by a set of output channels

$$\left\{ (c^{start}_{out,f}, c^{end}_{out,f})_i \mid i \in \{1, \dots, T_w\} \right\} \tag{30}$$

and a set of positions

$$\left\{ (u_f, v_f)_i \mid i \in \{1, \dots, T_a\} \right\} . \tag{31}$$

A set of output channels is found by substituting t_w with an iterator in Eqs. (20) and (21):

$$c^{start}_{out,f(i)} = (i-1) \cdot N + p_{col} + 1 \;\forall\; i \in \{1, \dots, T_w\} \tag{32}$$

$$c^{end}_{out,f(i)} = \begin{cases} i \cdot N & \text{if} \quad C_{out} \geqslant i \cdot N \\ C_{out} & \text{if} \quad C_{out} < i \cdot N \end{cases} \;\forall\; i \in \{1, \dots, T_w\} \tag{33}$$

which gives a pair $(c^{start}_{out,f}, c^{end}_{out,f})$ for every weight tile.

A set of positions is found by substituting t_a with an iterator in Eq. (22) for the affected sliding windows and then using Eqs. (23) and (24) to calculate position from the sliding window:

$$\square_{f(i)} = (i-1) \cdot N + p_{row} \;\forall\; i \in \{1, \dots, T_a\} \tag{34}$$

$$u_{f(i)} = \left\lfloor \frac{\square_{f(i)}}{W_{out}} \right\rfloor \tag{35}$$

$$v_{f(i)} = \square_{f(i)} \mod W_{out} \tag{36}$$

which gives a position for every input tile. This way, every bullet pattern is characterized by a Cartesian product of those two sets.

An error added to the output for each bullet pattern is the cumulative value

$$e^{p}_{ireg} = \sum_{c}^{C_{in}} \sum_{i}^{H_k} \sum_{j}^{W_k} w_{(c,i,j)} \cdot \varepsilon_{(c,u_f+i,v_f+j)} \tag{37}$$

where $\varepsilon_{(c,u_f+i,v_f+j)}$ is the error term added to input $x_{(c,u_f+i,v_f+j)}$. Since permanent faults are modeled as stuck-at-0 or stuck-at-1 faults, an error term is expressed as

$$\varepsilon = \begin{cases} 2^{\beta_f} \cdot \gamma & \text{if} \quad x_f^{(\beta_f)} \neq s \\ 0 & \text{if} \quad x_f^{(\beta_f)} = s \end{cases} \tag{38}$$

where s is the stuck-at state (0 or 1).

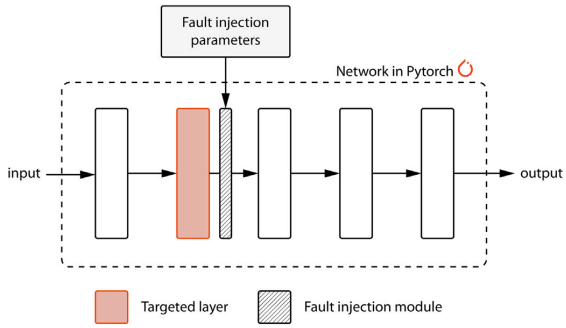


Fig. 5. Workflow of the fault injection process.

Table 4
Comparison of fault injection methods.

Work	Fault model	Output	Dataflow	Runtime
Saca-FI	Transient and permanent faults in registers	AVF	WS, IS, OS	47 s
Ours	Transient and permanent faults in registers and multipliers	AVF	OS[a]	0.0028 s

[a] The proposed method can be easily extended to other dataflows as well.

5.4. Implementation

The workflow of the proposed reliability assessment method combining fault injection with fault propagation analysis is presented in Fig. 5. Pytorch is used to run the inference of the network, leveraging GPU speed up. The fault injection module takes the output of the targeted layer and modifies only affected output values according to the aforementioned formulas. The simulation then continues with the erroneous values. The implementation of the proposed reliability assessment method will be made open-source.

6. Evaluation

6.1. Efficiency of reliability assessment method

The proposed reliability assessment method for systolic arrays is compared with the representative fault injection framework that models systolic array microarchitecture, Saca-FI [23]. While previous works rely on a cycle-accurate simulation of systolic array, which is a very time-consuming task, the proposed method is based on fault propagation analysis. Table 4 presents the comparison of two fault injection methods. Runtime denotes the processing time of one input with fault injection. For runtime comparison, the selected model is VGG-16 evaluated on the ILSVRC-2012 dataset, systolic array size is 256 × 256, fault is injected in the first convolutional layer. Both methods give AVF (Architecture Vulnerability Factor) as an output, the probability that a fault in the hardware structure causes an application output error [49]. While the functionality of the methods is similar, the proposed method is several orders of magnitude faster than the microarchitecture-level fault injection.

6.2. FORTALESA hardware implementation parameters

The proposed architecture of the systolic array is described in SystemVerilog and synthesized using Cadence Genus 2021 and 45 nm Nangate PDK. A comparison of the baseline systolic array without reconfiguration ability and four implementation options of FORTALESA is presented in Tables 5 and 6. Two sizes of the systolic array were selected: 48 × 48 and 132 × 132 to demonstrate the scalability of

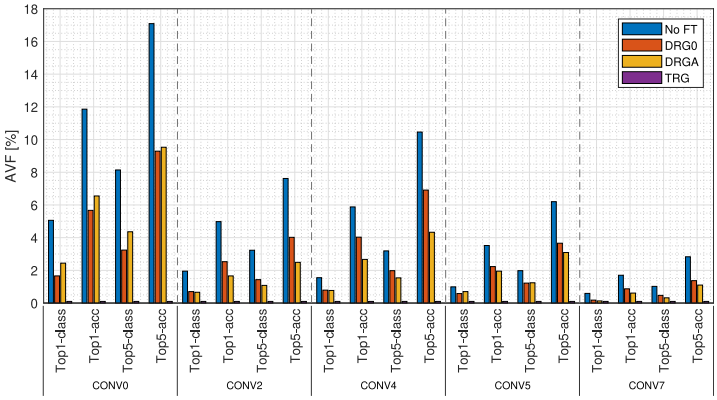


Fig. 6. Layer-wise reliability assessment of AlexNet considering transient faults.

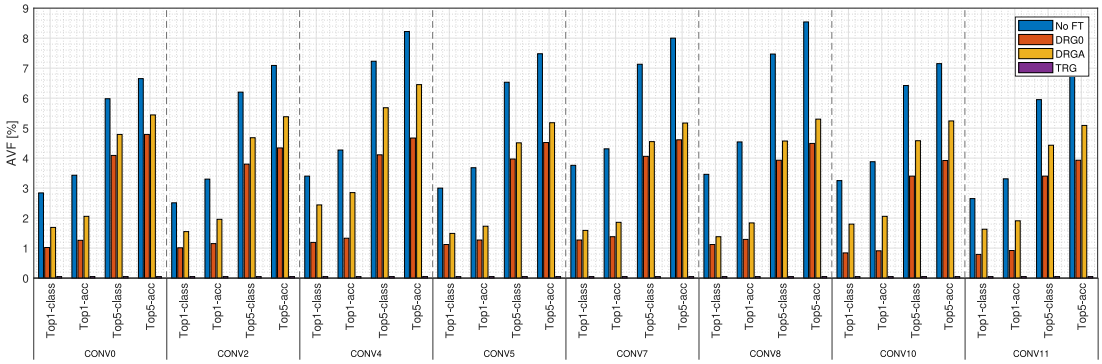


Fig. 7. Layer-wise reliability assessment of VGG-11 considering transient faults.

Table 5
Hardware implementation parameters for 48 × 48 systolic array.

Implementations	Area, mm^2	Power, W	Max frequency, MHz
Baseline SA	1.726	0.158	402
PM-DRG0-TRG3	1.937	0.177	357
PM-DRG0-TRG4	1.929	0.176	372
PM-DRGA-TRG3	2.129	0.193	303
PM-DRGA-TRG4	2.091	0.190	302

Table 6
Hardware implementation parameters for 132 × 132 systolic array.

Implementations	Area, mm^2	Power, W	Max frequency, MHz
Baseline SA	12.533	3.625	285
PM-DRG0-TRG3	14.243	4.053	252
PM-DRG0-TRG4	14.284	4.029	249
PM-DRGA-TRG3	15.826	4.552	230
PM-DRGA-TRG4	15.605	4.489	262

the design. Input and weight registers are 8-bit long, and the output register is 32-bit long. Due to the regular structure of the proposed architecture, the designs scale almost linearly proportionate to the increase in the number of processing elements. All implementations introduce around 5% overhead due to additional interconnects and MUXes, around 6%–7% due to correction mechanisms for DRG0 versions, and around 17%–18% for DRGA versions.

It can be noticed that frequency might drop considerably for the implementations with the averaging correction scheme for the DRG execution mode (DRGA). Critical path of these implementation variants includes an additional adder in the main PEs, thus increasing the datapath and lowering the frequency.

6.3. Reliability analysis

Experimental setup. Three common CNN models were used for reliability evaluation, including AlexNet, trained on the CIFAR-10 dataset, and VGG-11 and ResNet-18, trained on the ILSVRC-2012 dataset. Models and datasets were selected to represent different network scales. For the fault injection experiments, networks were quantized to an 8-bit integer format. Experiments were performed on NVIDIA GeForce RTX 3090 24G GPU.

Architectural vulnerability factor (AVF) was used as a reliability measure. AVF is the probability that a fault in the hardware structure causes an application output error [49]. Since the output of the selected models is the probability score of each class, the following output errors of the DNNs were considered [23]:

- Top1-class: the top-ranked class is different from the golden run.
- Top1-acc: the probability score of the top-ranked class is different. The top-ranked class may be different. Top1-acc includes the Top1-class errors.
- Top5-class: at least one class in the top-5 is different, including different order of the top-5 classes.

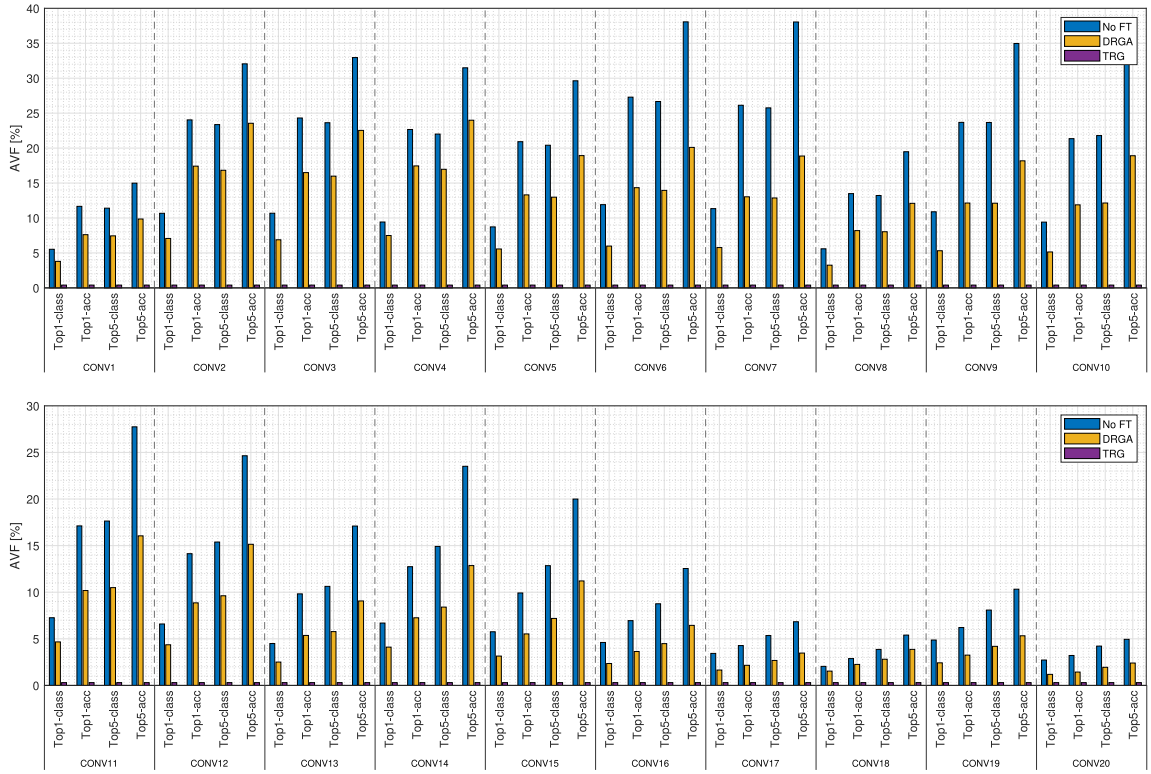


Fig. 8. Layer-wise reliability assessment of ResNet-18 considering transient faults.

- Top5-acc: the probability score of at least one class in the top-5 is different. The top-5 classes may be different. Top5-acc includes Top1-class, Top1-acc and Top5-class errors.

For transient fault analysis, layer-wise statistical fault injection was done following the equation introduced in [50] to achieve 95% confidence and 5% error margin. For each fault injection, fault parameters (Table 2) were set randomly and a test dataset of 10.000 inputs was fed to the network. Faults were injected in each convolution layer using the proposed methodology (Section 5). For the TRG execution mode, it is assumed that all faults are corrected, and there are no output errors. Fully connected layers were not considered for fault injection as they occupy only one row of the systolic array and their vulnerability factor, therefore, is very low [23].

Experimental results. Figs. 6, 7, and 8 show AVF values of each layer for selected networks considering different execution modes and configurations of the 48 × 48 systolic array and transient faults. For ResNet-18, only DRGA is considered since DRG0 correction scheme was not very efficient for this model.

Results show the varying vulnerability of different layers and the masking effect of DRG execution mode implementations. It can be seen that the DRG execution mode decreases the vulnerability of certain layers almost twice. Those results can be used to decide how to assign different layers of DNNs to different execution modes of the systolic array.

Fig. 9 shows AVF values for the reliability assessment of permanent faults for AlexNet. The analysis is done for stuck-at-1 faults since they are proven to be more critical [23]. Unlike transient faults, analysis is done for the whole network (all convolutional layers) as permanent faults persist throughout the network execution.

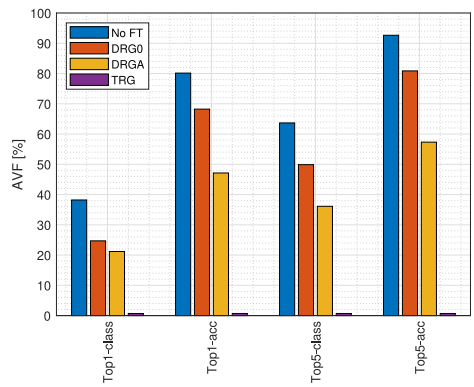
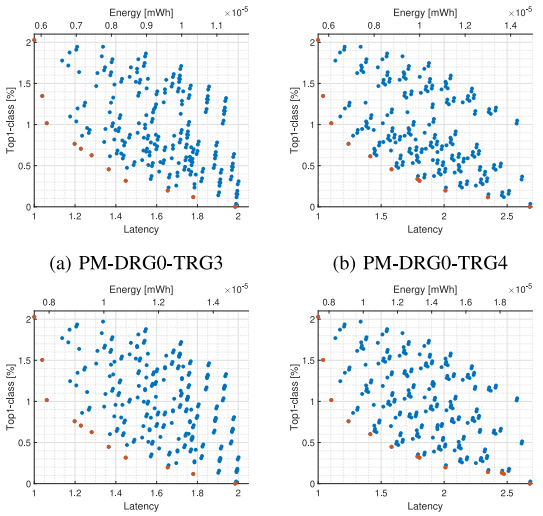


Fig. 9. Reliability assessment of AlexNet considering permanent faults (stuck-at-1).

6.4. Performance vs. reliability trade-off

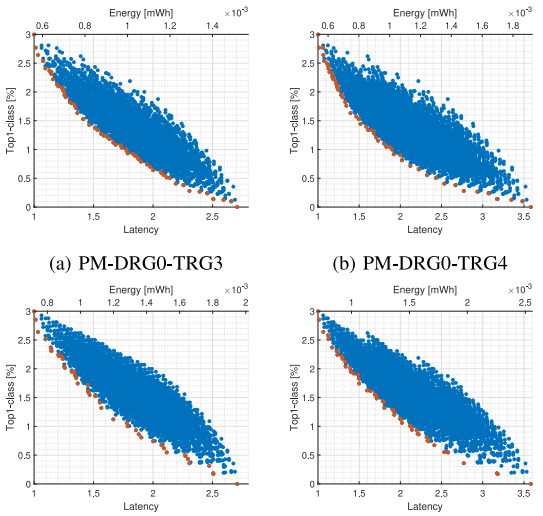
A flexible performance vs. reliability trade-off can be achieved by executing different layers of the DNN using different modes of the reconfigurable systolic array. Figs. 10, 11 and 12 plot reliability versus execution latency of the networks and energy consumption for all the possible combinations of modes. Latency is normalized to the execution of the network using performance mode for every layer (no fault tolerance). Energy is given in milliwatt-hour (mWh). For reliability,



(a) PM-DRG0-TRG3 (b) PM-DRG0-TRG4

(c) PM-DRGA-TRG3 (d) PM-DRGA-TRG4

Fig. 10. Reliability vs. latency and energy for AlexNet. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)



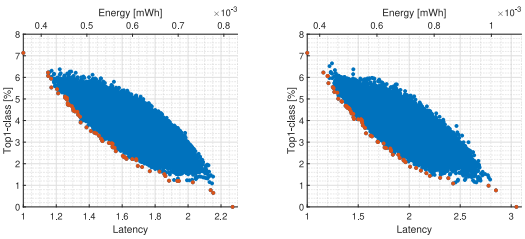
(a) PM-DRG0-TRG3 (b) PM-DRG0-TRG4

(c) PM-DRGA-TRG3 (d) PM-DRGA-TRG4

Fig. 11. Reliability vs. latency and energy for VGG-11. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Top1-class errors are considered. The Pareto front is marked with red color. Four implementations of FORTALESA are considered for each network since different implementations of DRG execution mode give slightly different reliability results, and different implementations of TRG execution mode affect the latency.

The left-most point of the Pareto front denotes the configuration where all layers are executed using performance mode. The execution latency is small, but there is no fault tolerance. The right-most point



(a) PM-DRGA-TRG3 (b) PM-DRGA-TRG4

Fig. 12. Reliability vs. latency and energy for ResNet-18. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

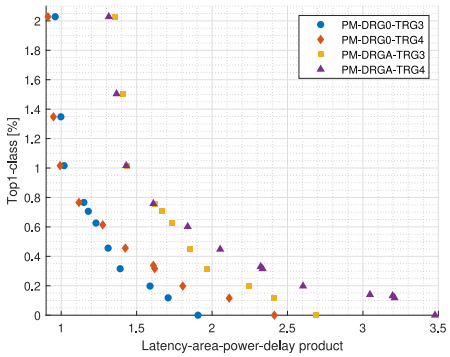


Fig. 13. Comparison of FORTALESA implementation options considering reliability for AlexNet.

denotes the configuration where all layers are executed using TRG mode. The execution latency is high, but all faults are corrected. The points in between denote configurations with a combination of execution modes, including DRG mode. The figures show that DRG execution mode allows for a flexible reliability–performance trade-off.

To compare different implementations of FORTALESA, the AVF of the Pareto front points was plotted against a product of latency, power, area, and delay (Figs. 13 and 14). It can be seen that while systolic arrays with the TRG4 implementation have smaller area and power, and higher frequency parameters, taking execution latency into consideration version with the TRG3 implementation shows better overall results.

These figures present a design space exploration with the highlighted Pareto front, allowing designers to select a trade-off that fits the needs and requirements of their system.

6.5. Comparison with static redundancy

The comparison of the proposed run-time reconfigurable systolic array architecture with the static TMR approaches is presented in Fig. 15. A power-area product is plotted against the maximum possible throughput of the architecture (since the throughput of FORTALESA depends on the execution mode). Throughput is calculated as the number of MAC operations performed by the systolic array in one clock cycle multiplied by the frequency. Different cases are considered for static TMR: triplication of registers only, triplication of registers and MAC units, and triplication of the whole array. As well as different sizes: 48×48 and 24×32 as this is the efficient size of the 48×48 systolic array operating in a TRG execution mode (TRG3 implementation option).

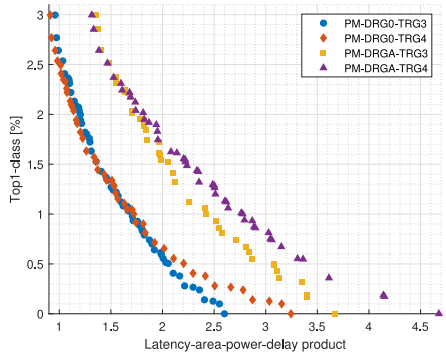


Fig. 14. Comparison of FORTALESA implementation options considering reliability for VGG-11.

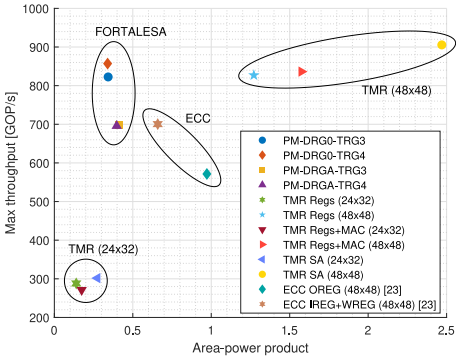


Fig. 15. Comparison of FORTALESA implementation options with static TMR and selective ECC [23].

As can be seen from the figure, the 24 × 32 systolic array with static TMR has a lower power-area product than the proposed architecture but has a fixed low throughput. On the other hand, the 48 × 48 systolic array with static TMR has high throughput, but the power-area product is significantly higher than the proposed architecture. FORTALESA, on average, requires 6× fewer resources than static TMR.

The proposed architecture is also compared with the selective ECC proposed in [23]. While it protects only selected registers, it requires, on average, 2.5× more resources than FORTALESA, which protects all registers and MAC units.

6.6. Comparison with the state-of-the-art

The comparison of the proposed architecture with other methods to enhance fault tolerance of systolic arrays is presented in Table 7. The proposed architecture provides a comprehensive solution that protects both registers and MAC units in the PEs, and since it utilizes a form of spatial redundancy, it covers both transient and permanent faults. Faults are corrected in both fault-tolerant modes without interrupting inference execution for recovery. Additionally, the proposed architecture provides support for reconfigurability, addressing the dynamic requirements of AI applications.

7. Conclusion

In this work, we proposed a run-time reconfigurable fault-tolerant systolic array architecture FORTALESA with three execution modes and four implementation options. All four implementation options were evaluated in terms of resource utilization, throughput, and fault tolerance improvement. The proposed architecture is used for reliability enhancement of DNN inference on a systolic array through heterogeneous mapping of different network layers to different execution modes. We also introduced a reliability assessment method based on fault propagation analysis. It is used to efficiently evaluate the vulnerability of DNN models and select the appropriate execution mode–layer mapping for DNN inference. The proposed architecture efficiently protects both registers and MAC units of systolic array PEs from transient and permanent faults without interrupting inference execution. The reconfigurability feature of FORTALESA enables a speedup of up to 3×, depending on layer vulnerability, and requires 6× fewer resources compared to static redundancy and 2.5× fewer resources than the previously proposed solution for transient faults. As future work, an ASIC (Application-Specific Integrated Circuit) based on FORTALESA will be developed for further testing, e.g., at radiation facilities. The design of the chip will be made open-source.

Table 7
Comparison with the state-of-the-art.

Work	Covered faults		Protected parts		Fault
	Permanent	Transient	Registers	MAC	correction
[27]	✓	✗	✗	✓	✓
[30]	✓	✗	✗	✓	✓[a]
[31]	✓	✗	✗	✓[b]	✗
[23]	✗	✓	✓	✗	✓[c]
[32]	✓	✗	✓	✓	✗
[36]	✗	✓[d]	✗	✓	✗
Our	✓	✓	✓	✓	✓

[a] Requires retraining of the DNN for each faulty chip.
[b] Faults in configurational memory of FPGA.
[c] Only single-bit fault correction (ECC).
[d] Timing faults from low voltage.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was supported in part by the Estonian Research Council grant PUT PRG1467 “CRASHLESS”, CoE TK202 “Universum” and by EU Grant Project 101160182 “TAICHIP”.

Data availability

No data was used for the research described in the article.

References

[1] N.P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, P.-l. Cantin, C. Chao, C. Clark, J. Coriell, M. Daley, M. Dau, J. Dean, B. Gelb, T.V. Ghaemmaghami, R. Gottipati, W. Gulland, R. Hagmann, C.R. Ho, D. Hogberg, J. Hu, R. Hundt, D. Hurt, J. Ibarz, A. Jaffey, A. Jaworski, A. Kaplan, H. Khaitan, D. Killebrew, A. Koch, N. Kumar, S. Lacy, J. Laudon, J. Law, D. Le, C. Leary, Z. Liu, K. Lucke, A. Lundin, G. MacKean, A. Maggiore, M. Mahony, K. Miller, R. Nagarajan, R. Narayanaswami, R. Ni, K. Nix, T. Norrie, M. Omernick, N. Penukonda, A. Phelps, J. Ross, M. Ross, A. Salek, E. Samadiani, C. Severn, G. Sizikov, M. Snelham, J. Souter, D. Steinberg, A. Swing, M. Tan, G. Thorson, B. Tian, H. Toma, E. Tuttle, V. Vasudevan, R. Walter, W. Wang, E. Wilcox, D.H. Yoon, In-Datacenter performance analysis of a tensor processing unit, in: 44th Annual International Symposium on Computer Architecture, 2017, pp. 1–12.

[2] Y.-H. Chen, T. Krishna, J. Emer, V. Sze, Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks, in: IEEE International Solid-State Circuits Conference, ISSCC 2016, Digest of Technical Papers, 2016, pp. 262–263.

[3] H. Genc, S. Kim, A. Amid, A. Haj-Ali, V. Iyer, P. Prakash, J. Zhao, D. Grubb, H. Liew, H. Mao, A. Ou, C. Schmidt, S. Steffl, J. Wright, I. Stoica, J. Ragan-Kelley, K. Asanovic, B. Nikolic, Y.S. Shao, Gemmini: Enabling systematic deep-learning architecture evaluation via Full-Stack integration, in: 58th Annual Design Automation Conference, DAC, 2021.

[4] M.H. Ahmadilivani, M. Taheri, J. Raik, M. Daneshtalab, M. Jenihhin, A systematic literature review on hardware reliability assessment methods for deep neural networks, ACM Comput. Surv. 56 (6) (2024).

[5] X. Iturbe, B. Venu, E. Ozer, J.-L. Poupat, G. Gimenez, H.-U. Zurek, The arm triple core Lock-Step (TCLS) processor, ACM Trans. Comput. Syst. 36 (2018) 1–30.

[6] M. Rogenmoser, N. Wistoff, P. Vogel, F. Gurkaynak, L. Benini, On-Demand redundancy grouping: Selectable Soft-Error tolerance for a multicore cluster, in: IEEE Computer Society Annual Symposium on VLSI, ISVLSI, 2022, pp. 398–401.

[7] F. Libano, B. Wilson, J. Anderson, M.J. Wirthlin, C. Cazzaniga, C. Frost, P. Rech, Selective hardening for neural networks in FPGAs, IEEE Trans. Nucl. Sci. 66 (2019) 216–222.

[8] C. Bolchini, L. Cassano, A. Miele, A. Nazzari, Selective hardening of CNNs based on layer vulnerability estimation, in: IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems, DFT, 2022.

[9] T.G. Bertoa, G. Gambardella, N.J. Fraser, M. Blott, J. McAllister, Fault-Tolerant neural network accelerators with selective TMR, IEEE Des. Test 40 (2023) 67–74.

[10] N. Khoshavi, A. Roohi, C. Broyles, S. Sargolzaei, Y. Bi, D.Z. Pan, SHIELDeNN: Online accelerated framework for fault-tolerant deep neural network architectures, in: Design Automation Conference, 2020.

[11] M.H. Ahmadilivani, M. Taheri, J. Raik, M. Daneshtalab, M. Jenihhin, Enhancing fault resilience of QNNs by selective neuron splitting, in: IEEE 5th International Conference on Artificial Intelligence Circuits and Systems, AICAS, 2023, pp. 1–5.

[12] U. Zahid, G. Gambardella, N.J. Fraser, M. Blott, K. Vissers, FAT: Training neural networks for reliable inference under hardware faults, in: International Test Conference, 2020.

[13] J. Deng, Y. Fang, Z. Du, Y. Wang, H. Li, O. Temam, P. Ienne, D. Novo, X. Li, Y. Chen, C. Wu, Retraining-based timing error mitigation for hardware neural networks, in: Design, Automation and Test in Europe Conference, DATE, 2015, pp. 593–596.

[14] D. Xu, K. Xing, C. Liu, Y. Wang, Y. Dai, L. Cheng, H. Li, L. Zhang, Resilient neural network training for accelerators with computing errors, in: IEEE 30th International Conference on Application-Specific Systems, Architectures and Processors, ASAP, 2019, pp. 99–102.

[15] E. Ozen, A. Orailoglu, 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, Digest of Technical Papers, ICCAD, 2020.

[16] E. Ozen, A. Orailoglu, Boosting Bit-Error resilience of DNN accelerators through median feature selection, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 39 (2020) 3250–3262.

[17] L.-H. Hoang, M.A. Hanif, M. Shafique, FT-ClipAct: Resilience analysis of deep neural networks and improving their fault tolerance using clipped activation, in: Design, Automation and Test in Europe, 2020.

[18] Z. Chen, G. Li, K. Pattabiraman, 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, 2021, pp. 1–13.

[19] F. Geissler, S. Qutub, M. Paulitsch, K. Pattabiraman, A Low-Cost strategic monitoring approach for scalable and interpretable error detection in deep neural networks, in: Computer Safety, Reliability, and Security: 42nd International Conference, SAFECOMP, 2023, pp. 75–88.

[20] B.F. Goldstein, V.C. Ferreira, S. Srinivasan, D. Das, A.S. Nery, S. Kundu, F.M. Franca, A lightweight error-resiliency mechanism for deep neural networks, in: International Symposium on Quality Electronic Design, ISQED, 2021, pp. 311–316.

[21] H.T. Kung, Why systolic architectures? Computer 15 (1) (1982) 37–46.

[22] E. Vacca, S. Azimi, L. Sterpone, A comprehensive analysis of transient errors on systolic arrays, in: 26th International Symposium on Design and Diagnostics of Electronic Circuits and Systems, DDECS, 2023, pp. 175–180.

[23] J. Tan, Q. Wang, K. Yan, X. Wei, X. Fu, Saca-FI: A microarchitecture-level fault injection framework for reliability analysis of systolic array based CNN accelerator, Future Gener. Comput. Syst. 147 (2023) 251–264.

[24] A. Samajdar, J.M. Joseph, Y. Zhu, P. Whatmough, M. Mattina, T. Krishna, A systematic methodology for characterizing scalability of DNN accelerators using SCALE-Sim, in: IEEE International Symposium on Performance Analysis of Systems and Software, ISPASS, 2020, pp. 58–68.

[25] U.K. Agarwal, A. Chan, A. Asgari, K. Pattabiraman, Towards reliability assessment of systolic arrays against Stuck-at Faults, in: 53rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks - Supplemental Volume, DSN-S, 2023, pp. 230–236.

[26] M. Taheri, M. Daneshtalab, J. Raik, M. Jenihhin, S. Pappalardo, P. Jimenez, B. Deveautour, A. Bosio, SAFFIRA: a framework for assessing the reliability of Systolic-Array-Based DNN accelerators, in: 27th International Symposium on Design and Diagnostics of Electronic Circuits and Systems, DDECS, 2024, pp. 19–24.

[27] Y. Zhao, K. Wang, A. Louri, FSA: An efficient Fault-tolerant systolic array-based DNN accelerator architecture, in: IEEE International Conference on Computer Design: VLSI in Computers and Processors, 2022, pp. 545–552.

[28] H. Lee, J. Park, S. Kang, An Area-Efficient systolic array redundancy architecture for reliable AI accelerator, IEEE Trans. Very Large Scale Integr. (VLSI) Syst. 32 (2024) 1950–1954.

[29] H. Lee, J. Kim, J. Park, S. Kang, STRAIT: Self-Test and Self-Recovery for AI accelerator, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 42 (2023) 3092–3104.

[30] J.J. Zhang, K. Basu, S. Garg, Fault-Tolerant systolic array based accelerators for deep neural network execution, IEEE Des. Test 36 (2019) 44–53.

[31] F. Libano, P. Rech, J. Brunhaver, Efficient error detection for matrix multiplication with systolic arrays on FPGAs, IEEE Trans. Comput. 72 (2023) 2390–2403.

[32] E. Vacca, G. Ajmone, L. Sterpone, RunSAFER: A novel runtime fault detection approach for systolic array accelerators, in: The 41st IEEE International Conference on Computer Design, 2023.

[33] G. Cora, E. Vacca, C.D. Sio, S. Azimi, L. Sterpone, RePAIR: Reconfigurable platform for AI resilience within RISC-V ecosystem, Lecture Notes in Comput. Sci. 15594 LNCS (2025) 71–87.

[34] C. Peltekis, D. Filippas, G. Dimitrakopoulos, Error checking for sparse systolic tensor arrays, in: 2024 IEEE 6th International Conference on AI Circuits and Systems, AICAS, 2024, pp. 552–556.

[35] C. Peltekis, C. Nicopoulos, G. Dimitrakopoulos, Periodic online testing for sparse systolic tensor arrays, in: 2025 14th International Conference on Modern Circuits and Systems Technologies, MOCAST, IEEE, 2025, pp. 1–6.

[36] M. Safarpour, R. Inanlou, O. Silvén, Algorithm level error detection in low voltage systolic array, IEEE Trans. Circuits Syst. II: Express Briefs 69 (2022) 569–573.

[37] M. Ibtesam, U.S. Solangi, J. Kim, M.A. Ansari, S. Park, Highly efficient test architecture for Low-Power AI accelerators, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 41 (2022) 2728–2738.

[38] H. Lee, J. Lee, S. Kang, A robust test architecture for Low-Power AI accelerators, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 44 (2025) 1581–1594.

[39] S. Yin, P. Ouyang, S. Tang, F. Tu, X. Li, S. Zheng, T. Lu, J. Gu, L. Liu, S. Wei, A high energy efficient reconfigurable hybrid neural network processor for deep learning applications, IEEE J. Solid-State Circuits 53 (2018) 968–982.

[40] C.-J. Lee, T.T. Yeh, ReSA: Reconfigurable systolic array for multiple tiny DNN tensors, ACM Trans. Archit. Code Optim. 37 (2024) 24.

[41] C. Peltekis, D. Filippas, G. Dimitrakopoulos, C. Nicopoulos, D. Pnevmatikatos, ArrayFlex: A systolic array architecture with configurable transparent pipelining, in: Design, Automation and Test in Europe Conference, DATE, 2023.

[42] T. Zhao, S. Miao, S. Lu, J. Cao, J. Qiu, X. Shi, K. Wang, L. He, Towards a reconfigurable systolic array with Multi-Level packing for transformers, in: Architecture and System Support for Transformer Models (ASSYST@ ISCA 2023), 2023.

[43] Y.-H. Chen, J. Emer, V. Sze, Eyeriss: A spatial architecture for Energy-Efficient dataflow for convolutional neural networks, in: ACM/IEEE 43rd Annual International Symposium on Computer Architecture, ISCA, 2016, pp. 367–379.

[44] R. Xu, S. Ma, Y. Wang, Y. Guo, HeSA: Heterogeneous systolic array architecture for compact CNNs hardware accelerators, in: Design, Automation and Test in Europe, DATE, 2021, pp. 657–662.

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

[46] S. Burel, A. Evans, L. Anghel, MOZART+: Masking outputs with zeros for improved architectural robustness and testing of DNN accelerators, IEEE Trans. Device Mater. Reliab. 22 (2) (2022) 120–128.

[47] K. Chellapilla, S. Puri, P. Simard, High performance convolutional neural networks for document processing, in: 10th International Workshop on Frontiers in Handwriting Recognition, 2006.

[48] C. Bolchini, L. Cassano, A. Miele, A. Toschi, Fast and accurate error simulation for CNNs against soft errors, IEEE Trans. Comput. 72 (2023) 984–997.

[49] S.S. Mukherjee, C. Weaver, J. Emer, S.K. Reinhardt, T. Austin, A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor, in: Annual International Symposium on Microarchitecture, MICRO, Vol. 2003-January, 2003, pp. 29–40.

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



Natalia Cherezova received her M.Sc. degree in computer systems engineering from Tallinn University of Technology, Tallinn, Estonia, in 2021. She is currently working towards her Ph.D. degree at Tallinn University of Technology. Her research interests include hardware reliability, digital systems design, and DNN accelerators.



Artur Jutman is a founder and Managing Director of Testonica Lab since 2005. He got a PhD degree in computer engineering from TalTech, Estonia in 2004. His professional focus is on design for testability and reliability as well as embedded instrumentation for system health monitoring and test. Dr. Jutman has coordinated several EU-funded research consortia, participated in organizing conferences and workshops across Europe.



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 from the same university in 2008. His research interests include reliable and efficient hardware for AI acceleration, EDA tools, as well as test, verification and debug topics. He has published more than 200 research papers, supervised several PhD students and postdocs, and served on the executive and program committees of numerous IEEE conferences (DATE, ETS, DDECS, LATS, NORCAS, etc.). Prof. Jenihhin coordinates European collaborative research projects TIRAMISU and TAICHIP, as well as national projects on energy efficiency and reliability of edge-AI chips and on cross-layer self-health awareness of autonomous systems.

Curriculum Vitae

1. Personal data

Name	Natalia Cherezova
Date and place of birth	14 December 1989 St. Petersburg, Russia
Nationality	Russian

2. Contact information

Address	Tallinn University of Technology, School of Information Technologies, Department of Computer Systems, Ehitajate tee 5, 19086 Tallinn, Estonia
E-mail	natalia.cherezova@ttu.ee

3. Education

2021–2026	Tallinn University of Technology, School of Information Technologies, Information and Communication Technology, PhD studies
2019–2021	Tallinn University of Technology, School of Information Technologies, Computer Systems Engineering, MSc *cum laude*
2016–2019	Tallinn University of Technology, School of Engineering, Integrated Engineering, BSc *cum laude*

4. Language competence

Russian	native
English	fluent
Japanese	pre-intermediate
Estonian	beginner

5. Computer skills

- Operating systems: Windows, Linux
- Document preparation: Microsoft Word, LaTeX
- Programming languages: C, C++, C#, Python, Matlab, Haskel, TCL
- Hardware description languages: VHDL, Verilog, SystemVerilog
- EDA tools: Vivado, Yosys, OpenRoad, KLayout, Cadence Genus, Cadence Innovus, QuestaSim

6. Defended theses

- 2021, HLS-Based Optimization of Tau Triggering Algorithms for LHC@CERN Application, MSc, supervisors Dr. Artur Jutman, Dr. Dmitri Mihhailov, Tallinn University of Technology, School of Information Technologies
- 2019, Digital Twin: Synchronization Between Virtual and Real World Equipment — ABB Robot Use Case, BSc, supervisor Dr. Vladimir Kuts, Tallinn University of Technology, School of Engineering

7. Fields of research

- Hardware reliability
- Digital systems design and optimization
- Hardware accelerators design

Elulookirjeldus

1. Isikuandmed

Nimi	Natalia Cherezova
Sünniaeg ja -koht	14.12.1989, Peterburi, Venemaa
Kodakondsus	Vene

2. Kontaktandmed

Aadress	Tallinna Tehnikaülikool, Arvutisüsteemide Instituut, Ehitajate tee 5, 19086 Tallinn, Estonia
E-post	natalia.cherezova@taltech.ee

3. Haridus

2021–2026	Tallinna Tehnikaülikool, Infotehnoloogia teaduskond, Info- ja kommunikatsioonitehnoloogia, doktoriõpe
2019–2021	Tallinna Tehnikaülikool, Infotehnoloogia teaduskond, Arvutisüsteemid, MSc *cum laude*
2016–2019	Tallinna Tehnikaülikool, Inseneriteaduskond, Integreeritud tehnoloogiad, BSc *cum laude*

4. Keelteoskus

vene keel	emakeel
inglise keel	kõrgtase
jaapani keel	eelkesktaseme
eesti keel	algaja

5. Arvuti oskused

- Operatsioonisüsteemid: Windows, Linux
- Kontoritarkvara: Microsoft Word, LaTeX
- Programmeerimiskeeled: C, C++, C#, Python, Matlab, Haskel, TCL
- Ristvarakirjelduskeeled: VHDL, Verilog, SystemVerilog
- Tööristad: Vivado, Yosys, OpenRoad, KLayout, Cadence Genus, Cadence Innovus, QuestaSim

6. Kaitstud lõputööd

- 2021, Kõrgtasemesünteesil põhinev tau-vallandumise algoritmide optimeerimine CERNi Suurele Hadronite Põrgutile, MSc, juhendajad Dr. Artur Jutman, Dr. Dmitri Mihhailov, Tallinna Tehnikaülikool, Infotehnoloogia teaduskond
- 2019, Digikaksikud: reaalsete ning virtuaalsete maiilmade seadmete sünkroniseerimine — ABB tööstusroboti näide, BSc, juhendaja Dr. Vladimir Kuts, Tallinna Tehnikaülikool, Inseneriteaduskond

ISSN 2585-6901
ISBN 978-9916-80-599-2