
Doctoral Dissertation
Doctoral Program in Computer and Control Engineering ($38^{th}$ cycle)

# New techniques for the Reliability Evaluation of AI-oriented Hardware Accelerators

**Robert Alexander Limas Sierra**

* * * * *

**Supervisors**
Matteo Sonza Reorda
Josie Esteban Rodríguez Condia

Politecnico di Torino
May, 2026

I hereby declare that the contents and organization of this dissertation constitute my own original work and do not compromise in any way the rights of third parties, including those relating to the security of personal data.

. . . . . .

Robert Alexander Limas Sierra

Turin, May, 2026

# Summary

Artificial Intelligence (AI) has become a cornerstone of modern computing, enabling applications in different domains such as autonomous vehicles, robotics, healthcare, finance, and High-Performance Computing (HPC). These domains demand high throughput, energy efficiency, and low latency, which has driven the widespread adoption of specialized AI-oriented hardware accelerators. In practice, these platforms include devices such as Graphics Processing Units (GPUs), Deep Learning Accelerators (DLAs), and Neural Processing Units (NPUs), often integrating dedicated tensor-acceleration engines to speed up matrix operations. They execute Machine Learning (ML) and Deep Learning (DL) workloads through massive parallelism, using architectures composed of many arithmetic units performing Multiply–Accumulate (MAC) operations.

As these accelerators move beyond data centers and into safety-critical domains—such as autonomous driving, aerospace, and advanced robotics—their reliability becomes a fundamental design concern. When used in HPC and data centers, their reliability is also a major concern, since the number of devices involved requires an extremely low per-device failure rate to guarantee an acceptable overall failure rate at the system level. AI-oriented accelerators are increasingly fabricated in deeply scaled semiconductor technologies that, while improving performance and power efficiency, also increase susceptibility to hardware faults. Radiation-induced upsets, process variations, and device aging can cause transient or permanent faults in logic and memory structures. Depending on where they occur and how they propagate, such faults may lead to Silent Data Corruptions (SDCs), where results are corrupted without detection, or Detected Unrecoverable Errors (DUEs), which may trigger exceptions, hangs, or application/system crashes. In safety-critical deployments and HPC environments, these outcomes can translate into mispredictions, accuracy degradation, or catastrophic failures.

The doctoral research presented in this thesis addresses these challenges by introducing new techniques for the reliability evaluation of AI-oriented hardware accelerators. The goal is twofold: (*i*) to develop methodologies for accurate and scalable reliability assessment across multiple abstraction levels, and (*ii*) to design lightweight mitigation strategies, co-optimized across hardware and software, that enhance fault resilience under realistic constraints.

Compared to the state of the art—often split between hardware-agnostic perturbation approaches (scalable but low fidelity) and low-level fault injection (accurate but expensive and difficult to generalize)—this thesis contributes a unified cross-layer flow that links circuit-level fault mechanisms to accelerator-level propagation and application-level impact, while remaining tractable for large workloads and adaptable across numerical formats.

At the arithmetic level, the thesis performs fine-grain reliability characterization of Floating-Point (FP) and Posit arithmetic cores implementing addition, multiplication, and MAC operations. Through detailed Fault Injection Campaigns (FICs) at gate and structural levels, the study identifies the substructures most responsible for error propagation and quantifies the magnitude and distribution of the resulting numerical corruptions. Rather than treating arithmetic units as black boxes or reporting only aggregate error rates, the proposed characterization connects observed error severity to specific internal blocks (e.g., exponent/regime handling, normalization, rounding). This enables feasible and format-aware mitigation decisions. The resulting vulnerability profiles explain why some FP designs exhibit rare but extreme outliers, whereas Posit designs may show higher activation rates with more bounded deviations.

At the accelerator level, the thesis investigates the reliability of tensor-core-style compute engines that accelerate General Matrix Multiplication (GEMM) operations, which are fundamental kernels in DNNs and CNNs. Faults are injected into internal Dot-Product Units (DPUs) and local memories to study how dataflow organization, workload mapping, and numerical format affect fault propagation and error accumulation. Beyond reporting error rates, the thesis shows that tensor-core faults generate deterministic spatial corruption signatures tied to warp-to-DPU mapping and MMA (matrix–multiply–accumulate) scheduling. This information is typically lost in application-level injection or overly abstract architectural models, yet it is essential to explain observed outcomes and to enable targeted runtime strategies that operate on tile fragments rather than duplicating entire kernels.

To reduce the computational cost of traditional FICs, the thesis proposes scalable evaluation frameworks combining analytical error models and an HPC-based execution environment. First, statistical fault models are developed to represent the impact of hardware faults as compact patterns of output corruptions in matrix multiplication results. Unlike fast simulators based on hardware-agnostic bit-flip assumptions, the proposed models are derived from low-level characterization at the gate level and preserve both magnitude statistics and the spatial corruption patterns observed in tensor-core executions, enabling higher fidelity at a fraction of the simulation cost. Second, an HPC-based distributed environment is introduced to accelerate large FIC campaigns through containerized execution across multiple compute nodes, enabling analyses that would otherwise be impractical on a single workstation.

Beyond evaluation, the thesis explores practical mitigation strategies. On the

hardware side, selective hardening is proposed as an efficient alternative to full redundancy. Instead of applying uniform replication, e.g., full Triple Modular Redundancy (TMR), across entire datapaths—often prohibitive for accelerator-grade arithmetic—the thesis leverages fine-grain vulnerability maps to protect only the blocks that dominate SDC and catastrophic outliers. This targeted approach improves reliability under tight overhead constraints and provides a methodology for selecting where redundancy, e.g., Dual Modular Redundancy (DMR), TMR, or self-checking, yields the highest reliability gain per area and power cost.

On the software side, a fault-tolerant GEMM mechanism is introduced for tensor-core accelerators. Unlike classical kernel duplication and generic software redundancy, which often incur multi-× slowdowns, the proposed mechanism exploits tensor-core tiling and redundant execution at fragment granularity. It provides continuous detection with low overhead and activates fine-grain reconstruction only upon anomaly detection. This hardware-aware design yields predictable overhead while enabling correction of localized persistent faults without requiring hardware changes.

The key contributions of this thesis can be summarized as follows:

- Development of a unified cross-layer reliability evaluation methodology for AI-oriented accelerators that bridges circuit-level fault mechanisms, tensor-core architectural propagation, and application-level impact.
- Fine-grain fault analysis of FP and Posit arithmetic cores that maps error severity and activation to specific internal substructures, enabling feasible and format-aware hardening decisions beyond black-box reliability metrics.
- Architectural reliability assessment of tensor-core-style accelerators that identifies deterministic spatial fault signatures induced by scheduling and DPU mapping, capturing effects not represented by hardware-agnostic tensor perturbations.
- Hardware-aware analytical error models that preserve both magnitude statistics and spatial corruption footprints while reducing evaluation cost compared to full architectural fault injection.
- Mitigation strategies with bounded overhead: selective redundancy guided by vulnerability profiles, and tensor-core-aware GEMM detection/correction that avoids full kernel duplication.

Overall, the thesis advocates a cross-layer approach to reliability in AI-oriented hardware accelerators, integrating low-level fault modeling, architectural analysis, and hardware/software mitigation strategies. The key advance is the combined emphasis on accuracy (structurally grounded fault mechanisms), scalability (analytical modeling and HPC-accelerated campaigns), and deployability (selective hardening

and software-only mitigation with predictable overhead), providing practical tools and design guidelines for dependable AI computing in critical environments.

# Acknowledgements


- First and foremost, I would like to express my deepest gratitude to my advisors, **Prof. Matteo Sonza Reorda** and **Prof. Josie Esteban Rodríguez Condia**. Their guidance, encouragement, and constant support have been instrumental throughout my doctoral journey. Matteo, thank you for your patience, insight, and for always challenging me to improve and think critically. Josie, thank you for your trust, kindness, and for showing me how to balance scientific rigor with creativity. Working under your supervision has been both a privilege and an inspiration.

- I would also like to sincerely thank my colleagues and friends **Juan** and **Nima** for the countless discussions, brainstorming sessions, and shared efforts that made even the most complex challenges enjoyable. Your collaboration, technical insights, and good humor have been invaluable, both inside and outside the lab. The long days of coding, debugging, and paper writing were made lighter thanks to your friendship and teamwork.

- To my **family**, especially my mother and father — none of this would have been possible without your love, sacrifices, and unwavering belief in me. You have always been my foundation, no matter how far from home I have been.

- To my **friends**, who have never let me face any difficulty alone, thank you for your continuous encouragement, laughter, and companionship. You reminded me that there is life beyond research and that true friendship endures even through the busiest times.

  **A todas las personas que me han acompañado en este camino, mis más profundas y sinceras gracias rotundas:** buenas son las risas que quedan en la memoria, recuerdos que iluminan incluso los días grises, un gesto sincero puede cambiarlo todo, jornadas largas que valieron la pena por la compañía, instantes que se vuelven eternos sin pretenderlo, ternura en las miradas que acompañan en silencio, amistad que permanece aunque el tiempo avance; algunas palabras no necesitan explicación.

‹‹‹

# Contents

# List of Tables

# List of Figures

# Acronyms

**ABFT** Algorithm-based Fault Tolerance. 45, 58

**AI** Artificial Intelligence. iii, v, vi, ix, xiii, 1–9, 11–17, 19, 21–23, 25, 32, 40, 43–45, 70, 71, 73, 74

**AVF** Architectural Vulnerability Factor. 17, 45

**BTI** Bias Temperature Instability. 14, 19, 43

**CMOS** Complementary Metal-Oxide-Semiconductor. 14, 15

**CNN** Convolutional Neural Network. iv, 1, 24, 39, 72

**CPU** Central Processing Unit. 10

**DL** Deep Learning. iii, 1, 3

**DLA** Deep Learning Accelerator. iii, 2, 10, 22

**DMR** Dual Modular Redundancy. v, xv, xvi, 5, 43, 44, 47, 49, 53, 54, 72

**DNN** Deep Neural Network. iv, 1, 9, 10, 23, 25

**DPU** Dot-Product Unit. iv, v, ix, xiii, xiv, 2, 4, 6, 7, 10, 11, 16, 19, 22, 25, 32, 33, 36–40, 57–63, 65, 69, 70, 72, 73

**DUE** Detected Unrecoverable Error. iii, xiv, 13, 14, 17, 22, 23, 28, 33–35, 38–40, 43, 47, 51, 72

**ECC** Error-Correcting Codes. 44, 65

**EM** Electromigration. 14, 19, 24, 41

**FAPR** Fault Activation and Propagation. ix, xiii, 17–19, 22, 28–30

**FI** Fault Injection. ix, 4–6, 16, 19, 22, 25, 40, 47

**FIC** Fault Injection Campaign. iv, 4

**FIT** Failures In Time. 2, 16

**FP** Floating-Point. iv, v, xiii, 5, 12, 25, 26, 43, 71–73

**FR** Fault Rate. 28, 47, 71

**GEMM** General Matrix Multiplication. iv, v, xiii, xvi, xvii, 2, 4–10, 18, 19, 22, 25, 32, 37, 38, 57–59, 61, 64–69, 72, 73

**GPU** Graphics Processing Unit. iii, xii, 2, 7, 9, 10, 15, 16, 19, 22–24, 29, 32, 44, 57–59, 64, 65, 67, 74

**HMMA** Half-precision Matrix Multiply-Accumulate. xiv, 11, 12, 19, 32, 33, 36, 37, 40, 57, 58, 60, 63–66, 68–70, 72, 73

**HPC** High-Performance Computing. iii–v, 5, 6, 8, 9, 58, 73

**IN** Input fault class (faults injected at DPU input signals). xiv, 22, 32–35, 38, 40, 72

**ISA** Instruction Set Architecture. 16, 23

**MAC** Multiply–Accumulate. iii, iv, xiii, 2, 5, 9–11, 26–28, 71, 74

**MAE** Mean Absolute Error. ix, xv, 18, 19, 22, 38, 39

**MFRO** Mean Fault Rate per Operation. 29, 47

**MFRS** Mean Fault Rate per Stimuli. ix, xiii, 17–19, 22, 28–30

**ML** Machine Learning. iii, 1, 21, 23

**MSB** Most Significant Bit. 12

**NPU** Neural Processing Unit. iii, 2, 3, 10, 19, 22

**OUT** Output fault class (faults at DPU output adder signals). xiv, 22, 32–36, 38, 40, 72

**PR** Product Register fault class (faults in DPU internal product registers). xiv, 22, 32–35, 38, 40, 72

**QMR** Quadruple Modular Redundancy. xvi, 61–63, 73

**RTL** Register-Transfer Level. x, 4, 7, 16, 21, 24, 25, 27, 29, 31

**S-CR** Self-Check and Repair. xii, xv, xvi, 43, 47, 49, 50, 52–55

**SAF** Stuck-at fault. 14, 18, 28, 33

**SDC** Silent Data Corruption. iii, v, ix, xii, xiv, 3, 5, 7, 15, 17, 19, 22, 23, 28, 29, 31, 33–35, 37–40, 43, 44, 46, 47, 49, 51–53, 55, 71–73

**SDE** Silent Data Error. ix, 3, 14, 17, 19

**SER** Soft error rate. 16

**SEU** Single Event Upset. 2, 15, 19, 22

**SIMT** Single instruction, multiple threads. 9

**SM** Streaming Multiprocessor. xvi, xvii, 10, 32, 33, 57, 58, 60, 61, 66, 67, 69, 73

**SoC** System-on-Chip. 10

**TCU** Tensor Core Unit. ix, xi, xiii, xiv, xvi, 2, 3, 6, 9–12, 15, 16, 19, 22, 25, 32–37, 39–41, 57–63, 65, 68–70, 72, 73

**TDDB** Time-Dependent Dielectric Breakdown. 14, 19, 43

**TMR** Triple Modular Redundancy. v, xv, xvi, 5, 43, 44, 46, 47, 49, 50, 52–55, 72–74

**TPU** Tensor Processing Unit. 10, 19

**WMMA** Warp Matrix Multiply-Accumulate. 64

# Chapter 1

# Introduction

New technologies are deeply transforming daily life by improving quality of service while increasing productivity and efficiency across society. Examples range from environmentally friendly transportation systems to highly automated manufacturing lines, where advanced control and monitoring reduce human effort and optimize the production of goods and services.

In the automotive domain, current trends focus on introducing new features that significantly increase the number and complexity of on-board embedded systems and processors [1]. These systems aim to optimize energy consumption, enhance the user experience through infotainment services, and provide autonomous or semi-autonomous driving capabilities [2]. In industrial environments, modern factories increasingly deploy autonomous or collaborative robots that operate alongside humans, automating hazardous tasks and boosting production throughput [3].

Both domains are representative examples of *safety-critical applications*, where any functional failure may lead to severe consequences, including serious injuries or fatalities, significant property damage, or extensive environmental impact [4]. As a result, the complex electronic devices now integrated into such systems must meet stringent safety, reliability, and security requirements to ensure correct operation throughout their lifetimes [5, 6].

A key enabler of this technological shift is Artificial Intelligence (AI) driven by modern Machine Learning (ML) and Deep Learning (DL) algorithms. In many safety-critical applications, workloads such as Deep Neural Networks (DNNs) and, in particular, Convolutional Neural Networks (CNNs) process massive volumes of sensor data under tight constraints in terms of latency, energy, and cost [7, 8]. To satisfy these requirements, modern computing platforms increasingly rely on heterogeneous architectures in which general-purpose processors are coupled with *AI-oriented hardware accelerators*, i.e., specialized engines optimized for ML and DL workloads [9].

These accelerators encompass a broad family of architectures, including Graphics Processing Units (GPUs), Deep Learning Accelerators (DLAs), Neural Processing Units (NPUs), and custom tensor engines. A common characteristic is the presence of highly parallel compute arrays tightly coupled with a carefully designed memory hierarchy [10]. In particular, many state-of-the-art GPUs integrate Tensor Core Units (TCUs) or similar tensor engines, which are matrix-oriented blocks implementing parallel dot-product operations through arrays of Dot-Product Units (DPUs) built around Multiply–Accumulate (MAC) datapaths [11, 12]. These structures are designed to accelerate General Matrix Multiplication (GEMM) kernels, to which convolutions, fully connected layers, and attention mechanisms are often reduced by compilers and libraries, such as cuBLAS (CUDA Basic Linear Algebra Subprograms) [11]. As a result, TCU-equipped GPUs and other AI accelerators are now deployed not only in data centers but also in embedded and safety-critical systems such as autonomous driving platforms, aerospace payloads, and advanced medical devices [6, 7].

To meet demanding throughput and energy-efficiency targets, these accelerators increasingly exploit deeply scaled semiconductor technologies, aggressive clock frequencies, reduced-voltage operation, dense integration, compact numerical formats, and mixed-precision execution [13]. However, the same trends exacerbate susceptibility to hardware faults. Several studies [14, 15, 16, 17, 18] show that modern devices implemented in cutting-edge technologies are prone to multiple types of faults both at early operational stages and, more frequently, during their operational life.

Such faults can stem from two broad classes of causes. On the one hand, *internal* causes include manufacturing defects that escape end-of-production testing, as well as device degradation due to long-term operation or idle aging effects (e.g., electromigration, time-dependent dielectric breakdown, or bias temperature instability) [19, 20, 21, 22]. These mechanisms typically give rise to intermittent or permanent faults. On the other hand, *external* causes are associated with environmental conditions, such as exposure to high-energy particles or electromagnetic interference, that temporarily or permanently alter the device's electrical characteristics [23, 24]. In this context, radiation-induced Single Event Upsets (SEUs) and related phenomena can corrupt the content of storage elements or logic nodes, leading to transient faults that may propagate as *soft errors* whenever the affected elements are exercised by the running application [25].

The Failures In Time (FIT) rate of advanced technologies is generally higher than that of previous nodes, especially for highly integrated devices such as modern processors and accelerators. Qualitative trends, often illustrated through bathtub-like curves, indicate that while the infant mortality period may remain almost constant, the useful-life phase tends to shorten, and aging-related failures become more prominent earlier in the lifetime [26, 27]. Consequently, ensuring long-term reliability in modern devices increasingly requires *in-field* testing and monitoring

techniques that complement traditional end-of-production tests [28, 21]. This need is particularly acute for safety-critical systems built upon AI-oriented accelerators, whose correct operation must be guaranteed throughout the mission profile of the application.

In the safety-critical domain, compliance with industrial standards such as ISO 26262 and IEC 61508 is mandatory to ensure a sufficient level of functional safety [5]. These standards prescribe a systematic process for hazard analysis, safety goal definition, and the allocation of safety requirements to hardware and software components. In practice, AI-oriented accelerators used in such systems must not only deliver high performance and low energy consumption, but also provide adequate diagnostic coverage and robustness against both permanent and transient faults [6, 7]. This combination of constraints creates a challenging design space, where performance, cost, and safety must be jointly optimized.

Beyond safety-critical domains, AI accelerators are also widely deployed in data centers and edge devices, where reliability directly impacts quality of service, availability, and total cost of ownership [29, 30]. Large-scale infrastructures have reported significant rates of Silent Data Corruption (SDC) and Silent Data Error (SDE) events in advanced processors and memories deployed at scale [29]. As AI workloads become pervasive in such environments, understanding and mitigating the reliability issues of AI accelerators is essential not only for safety but also for dependability and maintainability [13, 31].

## 1.1 Main Motivation

In practice, the development of modern AI-enabled, safety-critical applications requires three key ingredients: (*i*) high-performance operation and power efficiency, (*ii*) affordable costs, and (*iii*) guaranteed safety and reliability [32]. To satisfy the first two requirements, manufacturers and designers adopt advanced technologies and architectural specialization, integrating large numbers of compute units, memory structures, and dedicated accelerators (e.g., TCUs, systolic arrays, and NPU cores) on the same chip [10, 11]. This specialization is tightly coupled with the use of compact numerical formats and mixed-precision arithmetic, which significantly increase throughput and energy efficiency for DL workloads [13, 12].

However, these same choices aggravate reliability challenges. As feature sizes shrink, devices become more sensitive to process variations, environmental disturbances, and aging effects, increasing the likelihood of both transient and permanent faults [21, 27]. In addition, AI-oriented accelerators often operate close to their margins through aggressive voltage/frequency scaling and thermally dense integration, reducing noise margins and increasing susceptibility to timing and soft-error effects [25, 24]. The impact of these faults is often *silent*: numerical results may be corrupted without detection [29, 30], potentially degrading inference accuracy or

producing hazardous decisions in closed-loop safety-critical systems [8, 7].

A further challenge is that AI accelerators are not monolithic compute engines. Their reliability depends on the interaction between multiple layers of abstraction: arithmetic formats and datapaths, microarchitectural scheduling, dataflow and tiling, and application-level tolerance. Modern tensor engines execute GEMM as a sequence of tiled operations, repeatedly scheduling HMMA-style instructions across fixed DPU groups [11, 12]. As shown later in this thesis, faults within a single arithmetic substructure or DPU do not necessarily produce random perturbations; instead, they can generate deterministic error patterns whose spatial footprint depends on the mapping between warps, DPUs, and accumulation stages [33, 34]. This coupling implies that reliability evaluation must be both *hardware-aware* and *workload-aware*: the same physical fault may be masked or amplified depending on operand distributions, tile reuse, scheduling policies, and numerical format [13, 35].

These characteristics make reliability evaluation inherently challenging. Classical approaches rely on Fault Injection (FI), injecting faults into a model of the target device and observing their impact on representative workloads. The literature distinguishes three main strategies: beam experiments exposing a device to radiation, software-based error injection at the instruction level, and architectural or gate-level simulation [34, 36]. At low levels such as gate-level or Register-Transfer Level (RTL), FI can accurately capture structural masking, activation conditions, and the internal mechanisms through which exponent/regime paths, normalization, and rounding amplify numerical errors [35, 37]. However, exhaustive FIC at these levels becomes prohibitively expensive for large arithmetic cores and for the vast fault space of tensor engines. Even architectural fault injection at the accelerator level can require long runtimes when each injected fault must be exercised across realistic GEMM schedules and operand tiles [38, 39].

Conversely, high-level fault injection (e.g., perturbing weights or activations in software) offers excellent scalability but neglects key architectural and circuit-level details, including pipeline timing, operand alignment, buffer reuse, and format-specific sensitivity [8, 40]. A key open question in the field is whether software-based vulnerability evaluation methods provide representative results that can reliably predict real failure rates [41]. Similarly, fast error-simulation approaches that rely on generic bit-flip assumptions can misrepresent the structured error footprints induced by tensor-core execution and cannot capture the format-dependent behavior that distinguishes IEEE 754 floating-point from alternative formats such as Posit [42, 43]. In industrial settings, vendor-provided error abstractions may improve usability but can be opaque and device-specific, limiting interpretability and research reproducibility [44].

Accurate reliability evaluation alone is insufficient in safety-critical and high-availability applications: effective and affordable *mitigation* strategies must also be devised. In-field testing and software-based self-test techniques represent one

important mitigation direction: by periodically exercising the hardware during operation, latent faults can be detected before they cause silent failures in application workloads [28]. Classical hardware techniques such as Dual Modular Redundancy (DMR) and Triple Modular Redundancy (TMR) provide strong protection but introduce high area and power overheads that are often incompatible with accelerator efficiency constraints [45]. Selective hardening, where only the most fault-sensitive substructures are reinforced, is a more cost-effective alternative but requires fine-grain insight into which blocks dominate SDC and catastrophic numerical outliers [37]. Complementarily, software-based strategies can improve resilience without hardware modifications by exploiting warp-level redundancy, tile-level comparisons, and deterministic scheduling properties of tensor engines [46, 39], but these techniques must be carefully designed to avoid excessive overhead [47].

These considerations motivate the need for *new techniques* that (*i*) provide accurate yet tractable reliability evaluation tailored to AI-oriented accelerators across multiple abstraction levels, and (*ii*) leverage these evaluations to design lightweight mitigation strategies at hardware and software levels. This thesis addresses these needs through a unified cross-layer methodology centered on GEMM-dominated tensor workloads, combining fine-grain characterization, architectural analysis of tensor engines, scalable error modeling, and HPC-accelerated campaigns, and cost-effective mitigation.

## 1.2 Contribution

This thesis addresses the above challenges by developing and applying methodologies for reliability evaluation of AI-oriented hardware accelerators, and by using the resulting insights to guide lightweight mitigation strategies. It follows a twofold objective: (*i*) to devise evaluation techniques that enable accurate yet tractable reliability analyses across multiple abstraction levels tailored to AI-oriented accelerators, and (*ii*) to exploit these techniques to identify and validate hardware- and software-based mitigation strategies under realistic overhead constraints.

Concretely, the thesis makes the following main contributions:

- **Fine-grain reliability characterization and evaluation of arithmetic cores.** It reports results from an extensive FI-based evaluation of IEEE 754 FP and Posit arithmetic units implementing the most common operations in AI accelerators (addition, multiplication, and Multiply–Accumulate). By injecting permanent faults at the gate and structural levels using industrial tools [35], it quantifies fault activation, masking, and numerical error amplification across formats and internal pipeline stages. The analysis identifies a small set of dominant vulnerability sources (e.g., the *S&E* and *RnD* stages for FP, the *D&C* and *EN* stages for Posit) and distills these observations

into reusable vulnerability profiles, enabling structured and fair comparisons among arithmetic formats and implementations [37].

*Further details about the proposed methodology and results are available in the following publications:*

- *Analyzing the Reliability of TCUs Through Micro-architecture and Structural Evaluations for Two Real Number Formats*, Springer VLSI-SoC 2023 (conference paper) [48].
- *Investigating and Mitigating Critical Faults in Floating-Point and Posit Arithmetic Hardware*, IEEE Transactions on Emerging Topics in Computing, 2025 (journal paper) [37].

- **Structural reliability analysis of tensor-core-style accelerators for AI workloads.** Focusing on TCUs and tensor engines, it investigates how permanent faults affect mixed-precision matrix multiplication across multiple numerical formats. Using cycle-accurate or instruction-aware architectural models [33], it correlates fault locations within DPUs with both numerical deviation statistics and deterministic spatial corruption patterns on $16 \times 16$ GEMM tiles. The results show that scheduling policies, operand reuse, and warp-to-DPU mapping fundamentally shape fault propagation [11, 34].

  *Further details about the proposed methodology and results are available in the following publications:*

  - *Analyzing the Impact of Different Real Number Formats on the Structural Reliability of TCUs in GPUs*, IEEE VLSI-SoC 2023 (conference paper) [33].
  - *Analyzing the Reliability of TCUs Through Micro-architecture and Structural Evaluations for Two Real Number Formats*, Springer VLSI-SoC 2023 (conference paper) [48].
  - *Investigating and Mitigating Critical Faults in Floating-Point and Posit Arithmetic Hardware*, IEEE Transactions on Emerging Topics in Computing, 2025 (journal paper) [37].

- **Scalable reliability evaluation via error modeling and HPC.** To overcome the prohibitive cost of naive FI on AI workloads, it introduces complementary acceleration methodologies. It develops hardware-aware analytical error models that abstract tensor-engine faults into sparse spatial corruption masks combined with statistically grounded error generators [43], and an HPC-oriented execution framework enabling large-scale campaigns [49].

  *Further details about the proposed methodology and results are available in the following publications:*

  - *Effective Application-level Error Modeling of Permanent Faults on AI Accelerators*, IEEE IOLTS 2024 (conference paper) [43].
  - *Optimizing the Analysis and Evaluation of Logic Simulation Workloads in HPC Systems*, IEEE AICT 2023 (conference paper) [49].

- **Reliability-driven, hardware-based selective hardening of arithmetic units.** It investigates selective hardening mechanisms applied only to critical pipeline stages of arithmetic units, targeting the *S&E/D&C* and *RnD/EN* stages that dominate large-magnitude SDCs, and quantifying the trade-offs among resilience improvement, area, power, and timing overheads [37].

  *Further details about the proposed methodology and results are available in the following publications:*

  - *Investigating and Mitigating Critical Faults in Floating-Point and Posit Arithmetic Hardware*, IEEE Transactions on Emerging Topics in Computing, 2025 (journal paper) [37].

- **Software-based fault tolerance for tensor-core matrix multiplication on AI accelerators.** It proposes a software-only mechanism to detect and mitigate permanent and transient faults in tensor-core-based GEMM operations, enabling online correction with modest overhead on real GPU platforms [46]. The approach exploits the deterministic warp-to-DPU mapping and spatial error signatures identified in the structural analysis to achieve localized detection and conditional reconstruction with bounded overhead [39, 47].

  *Further details about the proposed methodology and results are available in the following publications:*

  - *A Software Fault-Tolerant Mechanism for Matrix Multiplication on Tensor Cores*, IEEE Transactions on Computers, 2026 (journal paper, submitted) [46].

**Research artifacts and reproducibility.** All experimental results presented in this thesis are supported by a suite of publicly available, modular research artifacts developed as part of this work. These artifacts include RTL and architectural models of tensor-core-style accelerators, instruction-aware simulators, scheduling and fault-propagation analysis frameworks, error-modeling toolchains, and software-level fault-tolerance prototypes for GPUs. The complete set of tools and documentation is available at:

- `https://github.com/robalexlimas/RTL_TCU.git`
- `https://github.com/TheColombianTeam/PyOpenTCU.git`

- https://github.com/TheColombianTeam/GPUScheduling.git
- https://github.com/robalexlimas/tensorcore-gemm-fault-tolerance.git

Overall, the thesis advocates a cross-layer perspective on reliability for AI-oriented hardware accelerators: from arithmetic formats and core micro-architectures, through tensor-engine scheduling and GEMM dataflows, up to software-level protection. By combining detailed characterization, accelerated evaluation, and both hardware- and software-level mitigation guided by these evaluations, the proposed work provides practical guidelines and tools for designing robust AI accelerators that can operate reliably in safety-critical and high-availability environments [31, 50].

The remainder of this thesis is organized as follows. Chapter 2 introduces the necessary background on AI accelerators, numerical formats, fault models, and reliability metrics. Chapter 3 presents fine-grain reliability characterization of arithmetic cores, structural reliability analysis of tensor-core-style accelerators, and scalable evaluation through analytical error modeling and HPC-based frameworks. Chapter 4 discusses reliability-driven hardware-level mitigation strategies, focusing on selective hardening mechanisms for arithmetic units. Chapter 5 introduces software-based fault-tolerance techniques for tensor-core matrix multiplication. Finally, Chapter 6 concludes the thesis, summarizing the main findings, highlighting the key contributions, and outlining future research directions in reliability evaluation and mitigation of AI-oriented hardware accelerators.

# Chapter 2

# Background

This chapter provides the necessary technical background to understand the reliability evaluation and mitigation methodologies developed in this thesis. It introduces the main architectural features of Artificial Intelligence (AI)-oriented hardware accelerators, the numerical representations employed in their arithmetic units, the fault models used for reliability analysis, and the standard reliability metrics adopted throughout the work.

## 2.1 AI-Oriented Hardware Accelerators

### 2.1.1 Overview of AI Acceleration Architectures

Modern AI workloads are dominated by computationally intensive kernels such as General Matrix Multiplication (GEMM) and convolution, which form the backbone of Deep Neural Networks (DNNs) — including convolutional, recurrent, and transformer architectures [51, 52]. These operations involve billions of multiply-accumulate (MAC) iterations and require extremely high arithmetic throughput that cannot be matched by conventional general-purpose processors [9]. To satisfy these demands, modern computing platforms rely on specialized *AI-oriented accelerators* designed to exploit massive data parallelism, operand reuse, and the regular computation patterns of linear algebra [52, 10].

The main families of AI accelerators include:

- **Graphics Processing Units (GPUs):** Originally designed for parallel graphics rendering [53], GPUs have evolved into massively parallel accelerators with thousands of cores executing the Single instruction, multiple threads (SIMT) paradigm [54, 55]. Programming frameworks such as CUDA and OpenCL make them versatile tools for both AI and High-Performance Computing (HPC) tasks. Modern GPUs additionally integrate Tensor Core

Unit (TCU)s, which execute mixed-precision GEMM operations through arrays of Dot-Product Unit (DPU)s [11, 12]. A single Streaming Multiprocessor (SM) in the Volta architecture contains four TCUs, each capable of one $4 \times 4$ matrix multiply-accumulate per clock cycle [11, 56].

- **Deep Learning Accelerators (DLAs):** Vendor-specific accelerators such as NVIDIA's DLA or Google's Tensor Processing Unit (TPU) target DNN inference and training workloads with systolic-array architectures and large on-chip weight buffers for high energy efficiency [10]. Systolic arrays arrange processing elements in a two-dimensional grid so that data flows rhythmically through the fabric, enabling dense matrix multiplications with minimal off-chip memory traffic [10, 52]. Google's first TPU achieved 92 TOPS with a $256 \times 256$ systolic array of 8-bit multipliers while consuming less than half the power of a contemporary GPU [10].
- **Neural Processing Units (NPUs):** Cores designed for DNN inference at the edge, with dedicated arithmetic units optimized for low-precision operations and stringent power budgets [9]. These units target embedded deployment in automotive, robotics, and IoT applications where latency and energy efficiency are paramount [52].
- **Custom Tensor Engines:** Modern System-on-Chips (SoCs) integrate custom tensor or matrix engines optimized for dense linear algebra with support for emerging numeric formats (e.g., FP8, BF16, Posit16) [9, 13]. These engines typically co-exist with GPU or CPU cores in heterogeneous architectures [52].

Despite their implementation diversity, these accelerators share a fundamental design principle: massively parallel compute arrays organized into hierarchical tiled structures, coupled with a carefully designed memory hierarchy to maximize data reuse and minimize costly off-chip communication [52, 10]. The energy cost of data movement can dominate over arithmetic, so maximizing operand reuse within on-chip buffers is a primary architectural objective [52]. Figure 2.1 conceptually illustrates this structure.

### 2.1.2 Tensor Core Unit (TCU) and Dot-Product Unit (DPU) Architecture

TCUs are in-chip matrix-acceleration engines introduced in NVIDIA's Volta architecture that execute warp-level mixed-precision matrix multiply-accumulate operations through four layers of parallelism: warps, thread groups, DPU arrays, and individual MAC units [11, 12]. Architecturally, a TCU comprises a $4 \times 4$ array of DPUs: each DPU computes one multiply-accumulate operation per cycle on

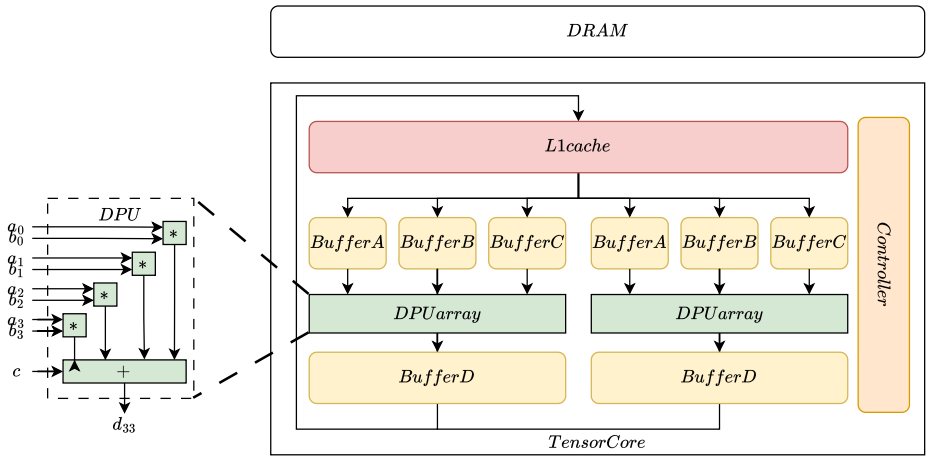



Figure 2.1: Conceptual organization of a tensor-core-style AI accelerator, illustrating the parallel compute array of DPUs, the on-chip accumulation buffers, and the hierarchical memory system. Adapted from [11, 57].

four input operands drawn from a pair of $4 \times 4$ matrix fragments $\mathbf{A}$ and $\mathbf{B}$ and an accumulator fragment $\mathbf{C}$ [11].

The fundamental operation performed by a TCU is:

$$\mathbf{D} = \mathbf{A} \times \mathbf{B} + \mathbf{C}, \tag{2.1}$$

where $\mathbf{A}$ and $\mathbf{B}$ are input matrices encoded at reduced precision (FP16 or Posit16), and $\mathbf{C}$ and $\mathbf{D}$ are the accumulator and output matrices at higher precision (FP32 or Posit32) [11, 13]. Large-scale matrix multiplications are decomposed via tiling into $16 \times 16$ fragments and executed as a sequence of Half-precision Matrix Multiply-Accumulate (HMMA) instructions, each consuming a $4 \times 4$ tile segment [11].

Each DPU within the TCU executes a scalar dot-product:

$$y_{ij} = \sum_{k=1}^{N} a_{ik} \cdot b_{kj} + c_{ij}, \tag{2.2}$$

implemented through a pipelined MAC datapath whose pipeline stages depend on the numerical format in use [58, 33]. Thread groups — sets of four consecutive threads within a warp — cooperate to load the required matrix fragments into shared register file banks and to schedule the four sets of HMMA instructions needed to produce each $16 \times 16$ output tile [11]. This deterministic scheduling creates a fixed, reproducible mapping from physical fault sites to spatial error patterns in the output matrix, a property extensively exploited in the reliability analysis of Chapter 3 [33, 43].

The reliability of these units depends not only on the adopted arithmetic precision but also on the dataflow strategy (e.g., weight-stationary or output-stationary),

as operand reuse and temporal correlation influence both fault activation probability and error accumulation across HMMA instruction sequences [52, 13].

## 2.2 Numerical Representations and Precision Formats

The numerical representation used in AI accelerators directly impacts their performance, energy efficiency, accuracy, and reliability [13, 59]. Compact, low-precision formats reduce memory bandwidth and arithmetic energy, enabling higher throughput; however, they alter the distribution of representable values and the structural sensitivity of arithmetic hardware to faults [35, 37]. This section introduces the mathematical structure of the number systems used throughout this thesis.

### 2.2.1 IEEE-754 Floating Point

The IEEE Standard for Floating-Point Arithmetic (IEEE 754) [60] is the dominant numeric format in modern processors and accelerators. An IEEE-754 Floating-Point (FP) number of $N$ bits is organized into three fields — sign ($s$, 1 bit), biased exponent ($e$, $w$ bits), and fractional mantissa ($f$, $p$ bits) — and represents the value:

$$x = (-1)^s \times (1.f)_2 \times 2^{(e-B)}, \tag{2.3}$$

where $B = 2^{w-1} - 1$ is the exponent bias ($B = 127$ for FP32, $B = 15$ for FP16) [60, 59]. The standard defines five exchange formats (FP16, BF16, FP32, FP64, FP128) and mandates five rounding modes, five exception flags, and four directed-rounding attributes, ensuring portable reproducibility across platforms [60].

Modern AI accelerators exploit mixed-precision execution: FP32 for parameter storage and gradient accumulation, FP16 or BF16 for TCU computation, and INT8 or FP8 for quantized inference [13, 12]. The precision hierarchy directly shapes fault sensitivity: a single-bit fault in the exponent field $e$ can shift the output value by a factor of $2^{2^{w-1}}$, producing catastrophic large-magnitude deviations. The relative error due to a fault in position $k$ of the mantissa (with $k = 1$ being the MSB) is [59]:

$$\varepsilon_{fp} \approx 2^{-k}, \tag{2.4}$$

showing that low-significance mantissa bits have minimal impact. This asymmetric sensitivity between exponent and mantissa fields is a central property that shapes the vulnerability profile of FP pipeline stages and is analyzed quantitatively in Chapter 3 [35]. A thorough treatment of floating-point arithmetic and its implications for numerical computing is provided in [59], and a detailed account of the hardware arithmetic algorithms is provided in [58].

The standard also defines special values (NaN, $\pm\infty$, subnormals) that provide exception semantics: a corrupted exponent can produce a spurious NaN or $\pm\infty$, which propagates through subsequent computations as a Detected Unrecoverable Error (DUE) [35, 33].

### 2.2.2 Posit Representation

The Posit format, proposed by Gustafson and Yonemoto as a direct replacement for IEEE-754 [61], encodes a real number using four consecutively packed fields within a fixed word of $N$ bits: sign ($s$, 1 bit), regime ($r$, variable-length unary), exponent ($e$, up to $es$ bits), and fraction ($f$, remaining bits). The decoded value is:

$$x = (-1)^s \times \text{useed}^k \times 2^e \times (1 + f), \tag{2.5}$$

where useed $= 2^{2^{es}}$ is the scale base and $k$ is the signed integer value conveyed by the regime's run-length [61]. For Posit16 with $es = 1$, useed $= 4$ and the representable range spans $[\text{minpos}, \text{maxpos}] = [2^{-28}, 2^{28}]$ — substantially wider than the FP16 range [61, 37].

The variable-length regime provides *tapered precision*: values near $\pm 1.0$ receive the maximum number of fraction bits (highest accuracy), while very large or very small values receive fewer fraction bits in exchange for a wider dynamic range [61]. Key differences from IEEE-754 include: (*i*) no NaN or infinity special values (replaced by a single *Not-A-Real* encoding for $\pm\infty$ exceptions); (*ii*) no subnormal gradual underflow; (*iii*) always-on round-to-nearest-even; and (*iv*) an optional 512-bit *quire* fused accumulator enabling exact dot-product summation [61, 37].

From a reliability perspective, bit faults in the regime field can trigger very large-magnitude deviations (analogous to exponent-field faults in FP), whereas errors in the fraction field produce small, often arithmetically masked perturbations [35]. Moreover, the variable-length regime encoding and decoding requires dedicated pipeline stages — *Decoding and Checking* (*D&C*) at the input and *Encoding* (*EN*) at the output — that have no direct equivalent in IEEE-754 hardware and exhibit distinct fault sensitivity profiles [37]. This asymmetric field-level sensitivity and its implications for selective hardening are analyzed quantitatively in Chapter 3.

## 2.3 Fault Models in AI Accelerators

Fault models formally define how physical defects and environmental disturbances alter circuit behavior. The selection of an appropriate model is critical for reliability evaluation, since different models capture different physical failure mechanisms and produce distinct error signatures at the architecture and application levels [45, 62]. This thesis focuses on permanent and transient faults as primary targets, along with their propagation and masking mechanisms.

### 2.3.1 Permanent Faults

Permanent faults are persistent hardware defects that alter circuit behavior indefinitely, arising either from manufacturing imperfections that escape end-of-production testing or from physical degradation during operational life [21, 27]. The dominant degradation mechanisms in advanced Complementary Metal-Oxide-Semiconductor (CMOS) technologies are:

- **Electromigration (EM):** Thermally activated transport of metal ions along conductor lines under high current density, resulting in voids (open circuits) or hillocks (short circuits) [21]. Electromigration is exacerbated in advanced technology nodes by the combination of narrower metal lines and higher current density requirements [22].
- **Bias Temperature Instability (BTI):** Interface-trap generation at the gate-oxide boundary under prolonged voltage stress, causing a progressive threshold-voltage ($V_{th}$) increase and drive-current reduction [14, 17]. Negative BTI (NBTI) dominates in PMOS transistors, while Positive BTI (PBTI) affects NMOS at high-$\kappa$ dielectric nodes. If $V_{th}$ shifts beyond timing margins, setup-time violations manifest as functional or stuck-at faults [17].
- **Time-Dependent Dielectric Breakdown (TDDB):** Trap-assisted tunneling through a stressed gate dielectric eventually creates a conductive path between gate and channel, resulting in a transistor stuck in a leaky or short-circuit state [21, 22].
- **Process variation:** Lithographic and doping imperfections at the nanometer scale cause transistor parameter distributions ($V_{th}$, $I_{on}$) to widen, increasing the probability that individual devices deviate enough to cause functional failures [21].

At the logical level, these physical mechanisms are abstracted as: (*i*) **Stuck-at faults** (SAFs) on logic nodes or flip-flop outputs — the dominant model for manufacturing test and structural reliability evaluation [58, 63]; (*ii*) **permanent bit-flips** in memory cells or register files; and (*iii*) **open or bridging faults** on interconnects [45]. The stuck-at model is adopted throughout this thesis because it accurately represents the dominant effect of both manufacturing defects and aging-induced failures in CMOS logic, and is directly supported by industrial fault simulation tools (e.g., Synopsys Z01X) [64].

Such faults may alter control signals, corrupt arithmetic computations, or freeze data paths, potentially leading to a Silent Data Error (SDE) [29, 30] if silently propagated through the computation, or to a Detected Unrecoverable Error (DUE) if they trigger an exception. In AI accelerators, the absence of exception detection for most numerical corruptions means that permanent faults in arithmetic datapaths

propagate silently through tensor operations, making SDC the most common and most dangerous outcome [35, 33].

### 2.3.2 Transient Faults

Transient faults, or *soft errors*, are short-lived signal perturbations caused by environmental or operational factors including high-energy radiation, electromagnetic interference, and power supply noise [23, 25]. The dominant mechanism in deeply scaled CMOS is the Single Event Upset (SEU): when a high-energy particle (e.g., cosmic-ray neutron, alpha particle, or proton) deposits a charge $Q_{dep}$ at a circuit node exceeding the node's critical charge $Q_c$, the stored bit value is transiently flipped [25, 65]:

$$Q_{dep} > Q_c \quad \Rightarrow \quad \text{bit flip.} \tag{2.6}$$

The critical charge $Q_c$ decreases with technology scaling, as reduced supply voltages and smaller node capacitances lower the energy barrier for charge collection [25, 24]. In GPUs and tensor engines deployed in avionics, automotive, or data-center environments, soft error rates from high-energy neutrons have been directly measured using beam experiments [65, 13], confirming that TCU-intensive workloads exhibit higher per-operation failure rates than conventional FP32 operations due to the amplification of small numerical errors through low-precision arithmetic [13].

The probability that a fault leads to an observable error at the application level depends on a cascade of conditional events:

$$P_{\text{error}} = P_{\text{fault}} \cdot P_{\text{activation}} \cdot P_{\text{propagation}} \cdot P_{\text{manifestation}}, \tag{2.7}$$

where $P_{\text{fault}}$ is the raw particle-strike rate per bit-hour, $P_{\text{activation}}$ is the probability that the affected node holds an architecturally relevant value at the time of the strike, $P_{\text{propagation}}$ is the probability that the corrupted value reaches the architectural output, and $P_{\text{manifestation}}$ is the probability that the resulting output deviation exceeds the system's error tolerance [45, 66]. Reducing any of these four factors constitutes a viable protection strategy.

### 2.3.3 Fault Propagation and Masking Mechanisms

Once activated, a fault's effect must propagate through the circuit to produce an erroneous output. Three classical mechanisms can prevent this propagation [45, 62]:

- **Temporal masking:** The faulty signal value is overwritten or flushed before it is consumed by a downstream stage (e.g., a pipeline register is overwritten in the next cycle, or the result belongs to dead code that is never read) [45].

- **Logical masking:** Boolean conditions in the circuit neutralize the corrupted signal (e.g., the faulty value enters an AND gate whose other input is constantly 0), preventing it from influencing downstream logic [45, 62].
- **Arithmetic masking:** The numerical perturbation induced by the fault is smaller than the precision of the output representation, so the final rounded result is unchanged [59, 35]. This mechanism is most effective for faults in low-significance mantissa or fraction bits of FP and Posit operands.

The combined masking probability significantly determines architectural vulnerability. In practice, the dominant sources of non-masked, large-magnitude errors in AI datapaths are faults in the *Sign & Exponent* (*S&E*) and *Rounding* (*RnD*) stages of FP pipelines, and in the *Decoding & Checking* (*D&C*) and *Encoding* (*EN*) stages of Posit pipelines — precisely because these stages process the magnitude-determining fields of the representation and bypass all three masking mechanisms [35, 37].

### 2.3.4 Fault Injection (FI) Techniques

FI is the primary experimental methodology employed throughout this thesis to characterize fault effects at multiple abstraction levels. The literature distinguishes three main categories [63, 64]:

**Radiation beam experiments** expose a device to high-energy neutron or proton beams and directly measure in-silicon fault rates [65, 13]. They represent the ground truth for soft error rate (SER) characterization and provide device-level FIT figures under realistic operating conditions. However, they require specialized accelerator facilities, are impractical for exhaustive fine-grain structural coverage, and cannot easily isolate the effect of individual fault sites.

**Software-based fault injection (SWIFI)** instruments the target application at the ISA level, flipping bit values in architectural registers or memory locations during execution [36, 38]. Tools such as SASSIFI [36] and NVBitFI [38] instrument GPU kernels using the NVBit binary instrumentation framework [67], enabling large-scale campaigns on real hardware. Although fast and non-intrusive, software injection cannot capture intra-pipeline masking, format-specific exception handling, or the structural error patterns induced by fixed DPU routing within TCUs — a limitation identified in recent cross-layer evaluations [35, 41].

**Structural (simulation-based) fault injection** inserts faults directly into Register-Transfer Level (RTL) or gate-level netlists using dedicated logic simulation engines [35, 37]. This approach provides the highest accuracy for characterizing internal substructures but grows in cost with circuit size and must be complemented by scalable modeling techniques for large workloads.

For each injected fault and each input stimulus, the faulty output is compared against the fault-free (i.e., *golden reference*) and classified as:

- **Masked:** the fault produces no change at the output;
- **Silent Data Corruption (SDC):** the fault corrupts one or more output values silently, without triggering any exception flag;
- **Detected Unrecoverable Error (DUE):** the fault prevents correct execution or triggers an exception (e.g., NaN or $\pm\infty$ in IEEE-754 outputs), making the error detectable but unrecoverable without protection.

## 2.4 Reliability Metrics

The reliability of AI accelerators is characterized through a hierarchy of complementary metrics spanning device-level failure rates, structural vulnerability factors, and application-level impact measures [45, 31, 50]. This section formally defines each metric used throughout the thesis.

### 2.4.1 Silent Data Corruption (SDC) and Silent Data Error (SDE) Rates

The Silent Data Corruption (SDC) and Silent Data Error (SDE) rates quantify the proportion of fault scenarios that lead to undetected numerical corruption or silent application-level deviation, respectively:

$$R_{\text{SDC}} = \frac{N_{\text{SDC}}}{N_{\text{total}}}, \qquad R_{\text{SDE}} = \frac{N_{\text{SDE}}}{N_{\text{total}}}, \tag{2.8}$$

where $N_{\text{SDC}}$ counts fault scenarios producing incorrect but apparently valid outputs, $N_{\text{SDE}}$ counts scenarios where the error manifests only at the application level (e.g., wrong classification), and $N_{\text{total}}$ is the total number of injected faults [29, 30]. In AI workloads, SDCs are the most dangerous failure mode because they propagate silently through multiple computation layers before potentially affecting a safety-critical decision [8, 7]. The distinction between SDC (numerical output corruption) and SDE (application-level consequence) is important: a fault may produce SDC without SDE if downstream inference layers are tolerant enough to absorb the numerical deviation through activation-function clipping or batch normalization [68, 31].

### 2.4.2 Fault Activation and Propagation (FAPR) and Mean Fault Rate per Stimuli (MFRS)

Beyond the standard AVF, this thesis adopts two complementary fine-grain reliability metrics derived from exhaustive structural fault injection campaigns [69, 37].

The Fault Activation and Propagation (FAPR) quantifies the overall structural vulnerability of an arithmetic core as the ratio between the number of fault sites that corrupt at least one output over the complete input stimulus set ($PF$) and the total number of stuck-at faults in the core (SAFs):

$$\text{FAPR} = \frac{PF}{SAFs}. \tag{2.9}$$

FAPR captures what fraction of the physical fault space is *dangerous* — capable of producing at least one output corruption regardless of the input [35, 37].

The Mean Fault Rate per Stimuli (MFRS) measures the average fault sensitivity of the core by computing the mean rate at which fault sites corrupt the output, averaged over all $N$ input stimulus vectors:

$$\text{MFRS} = \frac{\sum_{i=1}^{N} PF_i}{SAFs \times N}, \tag{2.10}$$

where $PF_i$ is the number of fault sites that corrupt the output under stimulus $i$ [69]. A high MFRS implies that, for a propagating fault, a large fraction of the input operand space is susceptible, indicating broad, workload-independent vulnerability [37].

Together, FAPR and MFRS provide complementary views of structural vulnerability: FAPR quantifies *how many* fault sites are potentially dangerous, while MFRS quantifies *how broadly* each dangerous fault corrupts application execution. Both metrics are computed per pipeline stage, enabling targeted identification of the dominant vulnerability sources to guide the selective hardening decisions of Chapter 4.

### 2.4.3 Mean Absolute Error (MAE)

For GEMM-level evaluation, the Mean Absolute Error (MAE) between the fault-free output matrix $\mathbf{D}$ (golden reference) and the faulty matrix $\widetilde{\mathbf{D}}$ provides a scalar measure of numerical corruption magnitude:

$$\text{MAE} = \frac{1}{m \cdot n} \sum_{i=1}^{m} \sum_{j=1}^{n} \left| \widetilde{D}_{ij} - D_{ij} \right|, \tag{2.11}$$

where $m \times n$ is the output tile dimension [43]. The MAE is used in Chapter 3 to validate the accuracy of the analytical error model against full structural fault injection results and to compare per-tile corruption severity across different fault classes and numerical formats.

## 2.5 Summary

This chapter introduced the foundations of reliability analysis for AI-oriented hardware accelerators. It first presented the architectural organization of modern acceleration platforms, including GPUs with TCUs [11, 12], TPU-style systolic-array accelerators [10], edge-oriented NPUs, and custom tensor engines [9]. It further explained how HMMA instruction scheduling and DPU mapping determine reproducible fault-propagation patterns in tiled matrix computations.

The chapter then formalized the two numerical representations studied throughout this thesis: IEEE-754 floating point [60, 59] and Posit [61]. Their key structural vulnerabilities were highlighted, namely the high sensitivity of exponent-field faults in floating-point and the regime/encoding sensitivity introduced by Posit arithmetic.

Next, the chapter classified permanent faults caused by EM, BTI, TDDB, and process variation, as well as transient faults such as SEUs, together with their physical origins and logical-level abstractions [21, 25, 23]. It also described the three canonical masking mechanisms: temporal, logical, and arithmetic masking [45].

Three classes of FI techniques were reviewed: radiation beam experiments [65], software-based fault injection using tools such as SASSIFI [36], NVBitFI [38], and NVBit [67], and structural simulation-based fault injection [35]. Each method offers a different tradeoff between realism, accuracy, scalability, and instrumentation cost.

Finally, the chapter formally defined the reliability metrics adopted throughout this thesis: SDC/SDE rates [29, 30], the structural vulnerability metrics FAPR and MFRS [69, 37], and the MAE metric used to quantify numerical corruption in GEMM-level outputs [43]. The following chapters build upon these principles to develop and validate the proposed cross-layer evaluation and protection techniques.

# Chapter 3

# Reliability Assessment Methodologies

Modern AI accelerators integrate deeply pipelined arithmetic units and massively parallel compute fabrics to sustain the growing computational demands of large-scale ML workloads. As technology nodes continue to shrink and architectural complexity increases, these platforms become progressively more susceptible to hardware faults. Such faults may silently corrupt intermediate computations, propagate through tensor operations, or trigger critical system failures. Understanding how faults originate inside arithmetic datapaths, how they traverse accelerator microarchitectures, and how they affect numerical correctness is therefore essential to design dependable AI systems and to derive realistic, cross-layer reliability models.

This chapter presents the reliability assessment methodology adopted in this thesis. The proposed approach combines (*i*) fine-grain fault characterization at RTL and gate level for arithmetic cores, (*ii*) structural vulnerability analysis of Tensor Core execution pipelines using a cycle-accurate architectural model, and (*iii*) a hardware-aware analytical error-impact model that enables scalable application-level reliability studies while preserving hardware-induced spatial and numerical signatures. The overarching objective is to construct a comprehensive and format-aware evaluation framework applicable to both IEEE 754 floating-point and Posit arithmetic implementations. Figure 3.1 presents the overall cross-layer methodology and the flow linking the three evaluation steps.

The chapter is organized as follows. Section 3.1 motivates the need for reliability assessment and reviews the state of the art, including existing cross-layer frameworks and their limitations. Section 3.2 presents the fine-grain evaluation methodology and results for FP32 and Posit32 arithmetic cores. Section 3.3 extends the analysis to Tensor Core pipelines, characterizing structural fault propagation and deterministic spatial error signatures. Section 3.4 introduces the analytical error-impact model and validates it against full structural fault injection. Finally,

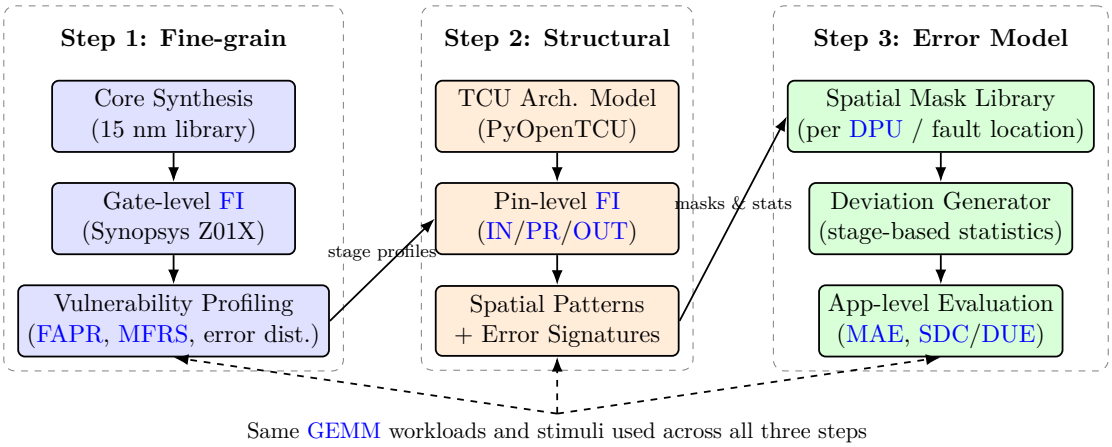



Figure 3.1: Overview of the proposed cross-layer reliability assessment methodology. Step 1 performs exhaustive gate-level fault injection on synthesized FP and Posit arithmetic cores to derive stage-level vulnerability profiles. Step 2 uses an instruction-accurate TCU model to analyze how these faults accumulate and generate spatial error signatures at the GEMM tile level. Step 3 abstracts these behaviors into a fast hardware-aware analytical model for scalable application-level reliability studies. The same GEMM workloads and stimuli are used across all three steps to ensure methodological consistency.

Section 3.5 summarizes the main findings and their positioning with respect to the state of the art.

## 3.1 Motivation and State of the Art

Reliability assessment has become a fundamental component of the design methodology for modern AI-oriented hardware accelerators across several application domains, including autonomous driving, healthcare, and high-performance computing. Multiple technological, architectural, and application-driven trends contribute to this necessity.

**Technology scaling.** Transistor dimensions are now in sub-10 nm territory. At these geometries, reduced supply voltages and noise margins increase susceptibility to manufacturing defects, process variation, electromigration, bias temperature instability, and time-dependent dielectric breakdown [21, 25, 27]. As degradation accumulates, both transient and permanent faults become more likely, while reduced critical charge increases sensitivity to radiation-induced phenomena such as SEUs [23, 24].

**Massively parallel execution.** Accelerators such as GPUs, NPUs, and DLAs execute billions of arithmetic operations per second. Under such conditions, even extremely small per-operation error probabilities can accumulate into significant

deviations, resulting in SDCs or DUEs [29, 30]. Faults affecting convolutional layers, attention blocks, or large-scale matrix multiplications may silently alter numerical outputs, reducing inference accuracy or compromising the stability of training algorithms [66, 7, 8]. Compact numerical formats and mixed-precision execution, while beneficial for throughput and energy efficiency, further increase fault activation probability and error magnitude by reducing the numerical dynamic range [13].

**Safety-critical deployment.** The use of ML components in safety-critical domains imposes strict reliability and functional-safety constraints [5, 6]. Faults that propagate into decision-making or control loops can produce hazardous outcomes, making hardware-level reliability assessment a prerequisite for certification under standards such as ISO 26262.

### 3.1.1 Survey of Reliability Evaluation Approaches

The reliability evaluation of AI accelerators and DNN workloads has been extensively studied over the past decade, with contributions spanning multiple abstraction levels [31, 50]. These surveys confirm a fundamental tension between evaluation accuracy and scalability: techniques closer to the hardware provide higher fidelity but are computationally expensive, while higher-level methods scale well but miss critical hardware-specific effects.

**Application- and software-level injection.** Several frameworks perturb weights, activations, or intermediate feature maps during software execution [8, 40, 68]. TensorFI [8] and PyTorchFI [70] operate at the framework level, injecting single-bit flips into tensors to approximate hardware faults. These tools scale to large networks and require no hardware access, but they cannot capture intra-pipeline structural masking, format-specific exception handling, or the operand-dependent activation conditions that arise inside arithmetic datapaths. Consequently, they may misclassify faults that would be masked at the gate level as propagating, or vice versa.

**Architectural-level injection.** Tools such as SASSIFI [36] and NVBitFI [38] operate at the ISA level, flipping register values at runtime using the NVBit binary instrumentation framework [67]. This approach faithfully models faults that materialize in architectural registers and can scale to realistic GPU workloads. However, arithmetic units are treated as black boxes: the internal pipeline stages responsible for exponent alignment, normalization, regime decoding, and rounding are invisible to these tools. As a result, the structured, format-specific error distributions that arise from gate-level faults in *Sign and Exponent / Decoding and Checking* stages cannot be reproduced, leading to inaccurate vulnerability estimates [41]. GUFI [71] and the SIFI framework [72] extend microarchitecture-level injection to AMD and NVIDIA simulators, providing finer-grain visibility into structural structures such as register files and caches, but still abstaining from arithmetic datapath internals.

**Cross-layer frameworks.** A number of works attempt to bridge abstraction levels. Condia et al. [73] combine architectural simulation and software fault injection to achieve a fast yet accurate CNN reliability evaluation on GPUs; their approach uses RTL fault simulation in a GPU model to derive syndrome tables, which are then applied through software injection on real hardware. Santos et al. [74] revealed GPU vulnerabilities by combining register-transfer and software-level injection, showing that register-file faults produce structural error patterns that software-only injection cannot capture. The gpuFI-4 framework [72] provides a microarchitecture-level cross-layer tool for NVIDIA GPUs, coupling functional simulation at multiple abstraction layers to assess the resilience of GPU applications end-to-end. Bolchini et al. [75] propose a cross-layer methodology that uses NVBitFI to characterize errors at the architectural level and maps them to application-level resilience estimates for different convolution implementations. Guerrero-Balaguera et al. [69] combine gate-level fault injection in GPU parallelism management units with software-level propagation to quantify permanent fault effects in control and scheduling logic. While these works represent important advances, they share a common limitation with respect to the present thesis: they characterize faults at the architectural register or memory level rather than inside the arithmetic cores themselves, leaving the format-specific vulnerability profiles of *Sign and Exponent*, *Decoding and Checking*, *Rounding*, and *Encoding* stages uncharacterized.

**Circuit-level analyses.** Gate-level and RTL fault injection provide the most accurate view of fault propagation inside datapaths [62, 45, 64]. Injection into specific nets, gates, or registers exposes operand-sensitive behavior, structural masking, and normalization effects invisible to higher-level tools. Their main limitation is scalability: the number of fault sites grows linearly with cell count, and exhaustive stimulation requires injecting each fault under many input vectors, leading to runtimes that scale with fault sites $\times$ stimuli [35].

**Error-modeling approaches.** To reduce cost while preserving hardware fidelity, several error-modeling methodologies have been proposed. Bolchini et al. [42] propose a fast simulator that perturbs CNN feature maps with statistically parameterized bit flips, avoiding gate-level simulation entirely. While the approach is fast and effective for transient fault emulation, it is hardware-agnostic: it ignores datapath structure, accumulation reuse, alignment and normalization, scheduling behavior, and numerical format, and therefore cannot reproduce hardware-induced spatial correlations or the magnitude distributions produced by real faults. The industrial EM approach of Saxena and Lotfi [44] proposes shipping encrypted, pre-characterized error macros to system designers. While attractive in closed industrial flows, this approach requires privileged vendor access, is opaque and non-extensible, and cannot be adapted to novel datapaths, academic prototypes, or emerging formats such as Posit.

### 3.1.2 Methodology Positioning and Contributions

A central methodological choice in this thesis is to focus reliability assessment on GEMM operations, rather than directly on full DNN workloads. Architecturally, GEMMs are the dominant compute primitive: convolutions, fully connected layers, and transformer attention reduce to batched matrix multiplications. Methodologically, GEMM-level analysis offers a deterministic execution model where mapping, operand movement, and accumulation reuse can be traced precisely — properties often obscured by full-application optimizations.

The proposed cross-layer methodology shares the general philosophy with some state-of-the-art frameworks [73, 75], thereby connecting multiple abstraction levels to derive application-relevant reliability metrics. However, it differs fundamentally in its *starting point*: rather than beginning at the architectural register or instruction level, this work initiates the characterization inside the *arithmetic core itself* at gate level, exposing format-specific vulnerability profiles — such as exponent-path sensitivity in FP and regime/encoding sensitivity in Posit — that are entirely invisible to tools such as SASSIFI [36] or NVBitFI [38]. This lower-level grounding is the key differentiator: it enables format-aware mitigation decisions that cannot be derived solely from application-level perturbations.

Three specific limitations of the proposed methodology should be acknowledged. First, gate-level FI campaigns are exhaustive but computationally intensive, limiting direct applicability to very large cores or complete accelerator netlists. Second, the evaluation targets stuck-at permanent faults; delay faults and bridging faults are not explicitly covered. Third, the analytical error model is derived from a specific TCU configuration ($16 \times 16$ tiles, $4 \times 4$ DPU array); extensions to different geometries require re-deriving spatial masks, though the methodology is general and applicable to any architecture.

## 3.2 Fine-Grain Reliability Evaluation at Gate Level and RTL

Fine-grain reliability evaluation provides the highest degree of precision when analyzing how hardware faults propagate through arithmetic circuits and affect numerical correctness. Unlike high-level or architectural assessments, RTL and gate-level analyses expose internal signals, control paths, and combinational structures. This visibility is essential for modern AI accelerators, where datapaths implement complex formats such as IEEE 754 FP and Posit, and where small perturbations in exponent alignment, normalization, regime decoding, or rounding can silently produce large deviations.

### 3.2.1 Pipeline Stage Organization of FP and Posit Arithmetic Cores

Each arithmetic unit is decomposed into well-defined structural pipeline stages reflecting its microarchitectural organization [37]. FP and Posit cores share a pipelined approach where the binary fields of the input operands are first separated and then processed by specialized circuits in sequential stages. Figure 3.2 shows the internal organization of the FP and Posit adder pipelines, illustrating the main stages and key substructures.

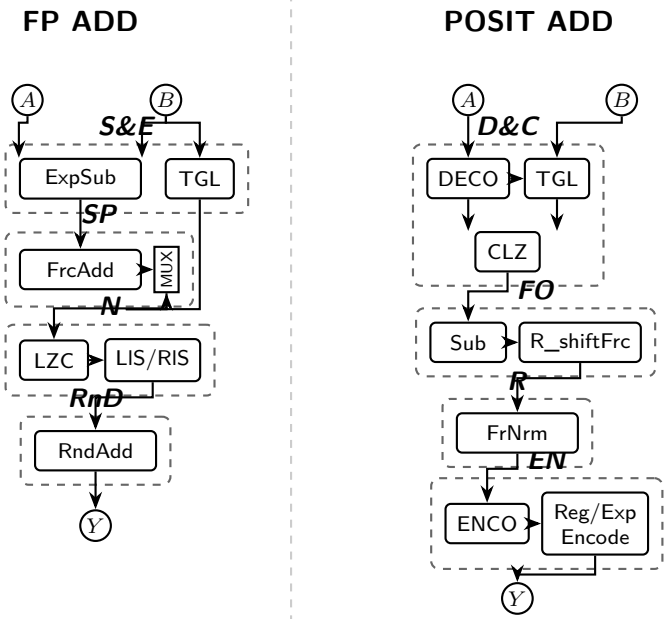



Figure 3.2: Internal pipeline organization of the FP (left) and Posit (right) adder cores used in this evaluation. For FP cores: the *S&E* stage processes sign and exponent fields via *ExpSub* and *TGL*; *SP* handles the fraction alignment and computation via *FrcAdd*; *N* normalizes the result using *LZC* and *LlS/RlS*; and *RnD* performs rounding via *RndAdd*. For Posit cores: *D&C* decodes the variable-length regime and exponent fields via *DECO*, *TGL*, and a *CLZ* counter; *FO* processes the fraction operands via *Sub* and *R_shiftFrc*; *R* normalizes and rounds via *FrNrm*; and *EN* re-encodes the result into Posit format via *ENCO* and *Reg/Exp Encode*. MAC cores add an *AccAdd* accumulation stage (*Accum*); the quire Posit MAC replaces it with a 512-bit *QA* stage. Adapted from [37].

**FP core pipeline.** The execution of FP cores comprises four main processing stages [37, 58]:

- **Sign and Exponent processing (*S&E*):** Handles sign calculation and exponent normalization, alignment, or comparison using XOR-based sign comparators, binary adders/subtractors (*ExpSub* in adders, *ExpAdd* in multipliers), and multiplexers.
- **Significant processing (*SP*):** Aligns input mantissas and computes the output significant through registers, binary shifters, binary adders (*FrcAdd*), and multipliers (*MantMul*).
- **Normalization (*N*):** Normalizes and composes the result using shifting mechanisms (*LlS*, *RlS*), leading-zero counters (*LZC*), and associated control logic.
- **Rounding (*RnD*):** Adjusts the result for precision using comparators and binary adders (*RndAdd*).

**Posit core pipeline.** Posit cores follow a four-stage organization with a distinct structure from FP [37, 61]:

- **Decoding and Checking (*D&C*):** Decodes variable-length regime and exponent fields into extended binary representations using two's-complement decoders, Counter-Leading-Zeros (*CLZ*) units (*Deco*, *TGL*, *TCDec*).
- **Fraction Operation (*FO*):** Processes fraction fields through binary adders (*Sub*, *FrcAdd*), shifting logic (*R_shiftFrc*), and fast binary multipliers (*MulFrac*).
- **Rounding (*R*):** Adjusts, rounds, and normalizes results through normalization logic (*FrNrm*).
- **Encoding (*EN*):** Re-encodes the result into Posit format from the regime, exponent, and fraction fields using encoder structures (*Enco*, *Reg/Exp Encode*).

**MAC extensions.** For MAC cores in both formats, an **Accumulation** stage (*AccAdd*) combines the multiplication product with input accumulator $C$. In the quire Posit MAC (*PQ_MAC*), this stage is replaced by a **Quire Accumulation (*QA*)** step that accumulates intermediate results in a 512-bit extended register without intermediate rounding [37].

The key reliability insight that motivates the entire methodology is the *asymmetric vulnerability* of these stages: faults in the *S&E*/*D&C* stages affect the magnitude-determining fields of the numerical representation (exponent in FP, regime in Posit) and produce large-magnitude catastrophic errors, while faults in *SP*/*FO* stages typically produce small errors that are masked by application-level tolerance in CNN workloads [8, 35].

### 3.2.2 Methodology

The fine-grain evaluation targets permanent SAFs injected exhaustively at the gate level of synthesized arithmetic core netlists. The flow consists of two strictly separated phases.

In the **reference phase**, all arithmetic units are simulated without faults using a large set of real-valued input stimuli derived from $16 \times 16$ matrix-multiplication workloads over two operand ranges ($\pm 1.0$ and $\pm 10.0$). Stimuli are generated once at real-number precision and encoded into FP32 and Posit32 using `numpy` [76] and `SoftPosit` [77], respectively. This ensures that both formats receive mathematically equivalent inputs, isolating format-induced vulnerability differences from input bias. Fault-free outputs are stored as golden references.

In the **fault-injection phase**, every stuck-at fault site is activated in turn, and the complete stimulus set is replayed. For each $fault, stimulus$ pair, the output is compared against the golden reference and classified as masked, SDC, or DUE [35]. When an SDC occurs, the absolute numerical error is recorded together with the fault's structural stage assignment. This produces a fault–error database that links physical fault sites to activation probability, error magnitude, and structural origin, from which FAPR, MFRS, and stage-level error distributions are extracted.

### 3.2.3 Experimental Setup

The experimental campaign targets 13 arithmetic units implementing addition, multiplication, and MAC operations in FP32 and Posit32 formats [35, 37]. All designs are synthesized using the 15 nm FreePDK standard-cell library [78] under identical constraints. Table 3.1 summarizes the main features of each core, including cell count, area, total stuck-at fault count (SAFs), and pipeline stages.

Fault injection is performed using Synopsys Z01X. For each core, every stuck-at fault site is evaluated under 4,096 input vectors, cumulating over 65,000 faults per core. The campaigns required approximately 320 hours on two servers equipped with 12 Intel Xeon CPUs at 2.5 GHz and 256 GB RAM. Fault-free references are generated with ModelSim.

### 3.2.4 Results

The fine-grain evaluation reveals distinct reliability profiles for FP and Posit arithmetic units. Across all designs, adders exhibit lower fault propagation rates than multipliers, consistent with the deeper and more interconnected datapaths of multiplication circuits. The Fault Rate (FR), defined as the percentage of fault sites corrupting at least one operation, ranges from 70%–89% for multipliers and 68%–85% for adders.

Table 3.1: Main features of the evaluated arithmetic units [35, 37]. The suffix _F denotes FloPoCo soft-core implementations [79]; non-_F cores originate from a GPU reference implementation (FP) and a RISC-V processor with Posit support (Posit). FloPoCo cores omit hardware exception handling, which does not affect the validity of the SDC analysis.

| Format | Op. | Core | Cells | Area ($\mu m^2$) | SAFs | Main stages (key substructures) |
|---|---|---|---|---|---|---|
| Posit | ADD | P_ADD | 1,816 | 634.9 | 12,900 | D&C (*Deco, TGL*), FO (*Sub, R_shiftFrc*), R (*FrNrm*), EN (*Enco*) |
| | | P_ADD_F | 1,592 | 581.7 | 11,846 | D&C (*TCDec, TGL*), FO (*Sub*), R (*FrNrm*), EN (*Enco*) |
| | MUL | P_MUL | 3,921 | 1,721.1 | 26,510 | D&C (*Deco*), FO (*MulFrac*), EN (*Enco, Reg/Exp Encode*) |
| | | P_MUL_F | 2,991 | 1,503.3 | 22,038 | FO (*MulFrac*), R (*Norm*), EN (*Enco*) |
| | MAC | P_MAC | 14,024 | 5,040.8 | 94,083 | D&C, FO (*MulFrac*), R (*FrNrm*), EN (*Enco*), Accum (*AccAdd*) |
| | | P_MAC_F | 9,712 | 3,745.2 | 61,830 | FO (*MulFrac*), R (*FrNrm*), EN (*Enco*), Accum (*AccAdd*) |
| | | PQ_MAC | 13,896 | 6,260.8 | 81,796 | D&C, FO (*MulFrac*), R (*FrNrm*), EN (*Enco*), QA |
| FP | ADD | FP_ADD | 1,275 | 345.3 | 9,366 | S&E (*ExpSub, TGL*), SP (*FrcAdd*), N (*LZC, RlS*), RnD (*RndAdd*) |
| | | FP_ADD_F | 1,043 | 387.1 | 6,960 | S&E (*ExpSub*), SP (*Aligner*), N (*LZC*), RnD (*RndAdd*) |
| | MUL | FP_MUL | 1,531 | 611.1 | 11,518 | S&E (*ExpAdd*), SP (*MantMul*), N (*Norm*), RnD (*RndAdd*) |
| | | FP_MUL_F | 2,004 | 984.8 | 13,860 | SP (*MantMul*), N (*Norm*), RnD (*RndAdd*) |
| | MAC | FP_MAC | 2,815 | 956.6 | 20,904 | S&E, SP (*MantMul*), N (*Norm*), RnD (*RndAdd*), Accum (*AccAdd*) |
| | | FP_MAC_F | 3,951 | 1,651.7 | 26,586 | SP (*MantMul*), N (*Norm*), RnD (*RndAdd*), Accum (*AccAdd*) |

The Mean Fault Rate per Operation (MFRO), quantifying the fraction of operations corrupted per propagating fault, shows strong format dependence. Figure 3.3 reports FAPR and MFRS averaged over both operand ranges across all units. FP32 multipliers reach MFRO values from 14%–31%, while Posit32 multipliers reach 46%–57%. This elevated activation in Posit is driven by the structural complexity of the *D&C* and *EN* stages, which involve regime run-length decoding and variable-length re-encoding that has no equivalent in FP hardware.

Error severity shows the *opposite* trend: FP units produce rarer but much larger catastrophic outliers, predominantly linked to *S&E* stage faults (exponent field corruption). Figure 3.4 reports absolute error distributions across the pipeline stages of FP and Posit adders for the ±10 operand range. In the FP adder, the four bars labeled *S&E*, *SP*, *N*, and *RnD* correspond precisely to the four pipeline stages

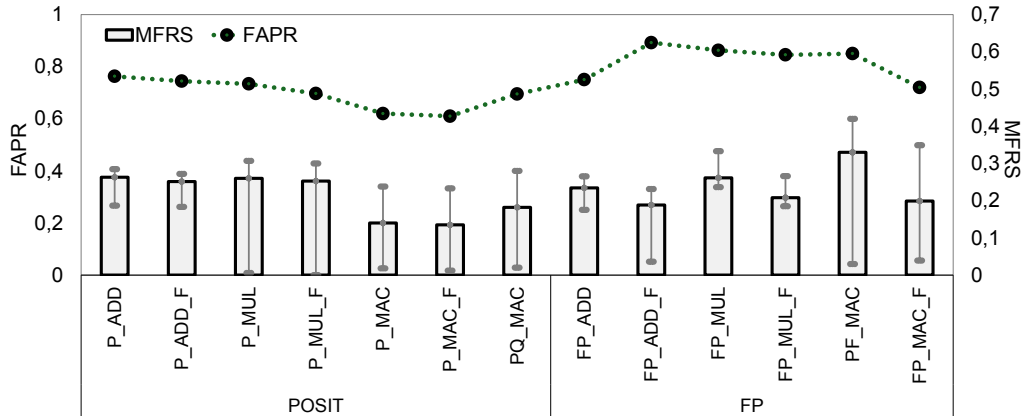
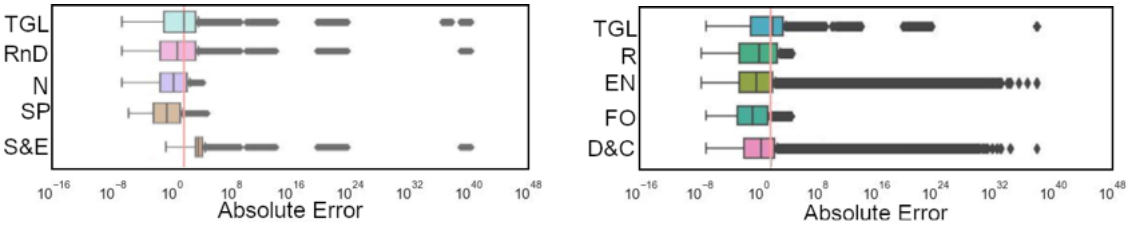



Figure 3.3: Fault Activation and Propagation (FAPR) (FR, left y-axis) and Mean Fault Rate per Stimuli (MFRS) (MFRO, right y-axis) for all FP32 and Posit32 arithmetic units [37]. Higher MFRO in Posit reflects the structural complexity of the *D&C* and *EN* stages. Higher FAPR in multipliers reflects their deeper datapaths.

described above. In the Posit adder, the bars labeled *D&C*, *FO*, *R*, and *EN* correspond to the Posit pipeline stages. In extreme cases, FP adder deviations exceed $3.2 \times 10^{38}$, whereas Posit errors remain bounded near $1.3 \times 10^{36}$, consistent with Posit's bounded dynamic range. The catastrophic FP outliers are exclusively produced by *S&E* and *RnD* stage faults, which corrupt the exponent field or introduce incorrect rounding that exponentially scales the output [37]. An analogous bimodal distribution is observed for multipliers: *S&E* faults (exponent addition) produce deviations above $10^{36}$, while *D&C*/*EN* faults in Posit are bounded but still dominant for large-magnitude outliers in that format.




(a) FP_ADD (b) POSIT_ADD

Figure 3.4: Absolute-error distributions across pipeline stages of FP and Posit adders (±10 input range) [35]. For FP_ADD: *S&E* (sign and exponent), *SP* (significant processing), *N* (normalization), *RnD* (rounding). For POSIT_ADD: *D&C* (decoding and checking), *FO* (fraction operation), *R* (rounding/normalization), *EN* (encoding). Faults in *S&E* (FP) and *D&C*/*EN* (Posit) produce large-magnitude catastrophic errors; faults in *SP*/*FO* produce small benign deviations.

Despite the presence of outliers, most corruptions are small. For multipliers, more than 87.5% of FP errors and 83.9% of Posit errors have magnitude below 1.0; for adders, 58.5% (FP) and 60.4% (Posit) of SDCs fall in this benign region. The concentration of catastrophic behavior in specific stages — *S&E* and *RnD* for FP, *D&C* and *EN* for Posit — is the key insight later exploited for selective hardening in Chapter 4.

Figure 3.5 compares error magnitude cumulative distributions for FP and Posit multipliers, further illustrating the bimodal impact structure [37].

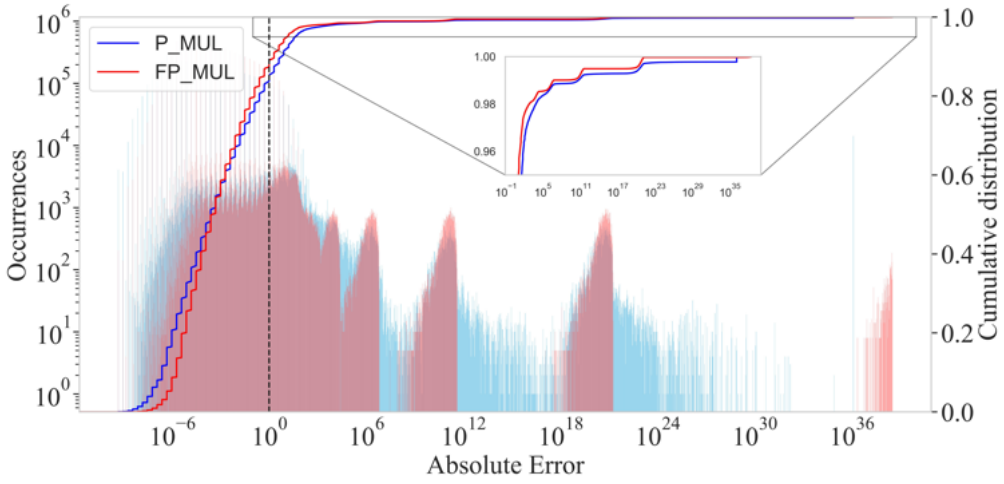



Figure 3.5: Error magnitude and cumulative distribution for FP and Posit multipliers under permanent faults [37]. The bimodal structure — a dominant cluster below 1.0 and a sparse tail of catastrophic outliers — is a universal property of both formats, with FP exhibiting larger outliers due to the wider dynamic range of the exponent field.

### 3.2.5 Summary

Gate-level FI of 13 arithmetic cores reveals that Posit cores exhibit higher fault activation (driven by *D&C*/*EN* structural complexity) and broader distributions of small-to-medium errors, while FP cores produce fewer but much larger catastrophic corruptions concentrated in *S&E* and *RnD* stages. These stage-level vulnerability profiles form the foundation for the structural TCU analysis in the next section.

## 3.3 Structural Vulnerability Analysis of Tensor Core Pipelines

Fine-grain analyses characterize isolated arithmetic cores, but do not capture propagation through complete Tensor Core execution pipelines. In modern GPU architectures, TCUs operate within a microarchitectural context that includes operand buffering, accumulation registers, pipeline timing, and warp scheduling. The interaction among DPUs, operand loaders, and HMMA instruction sequencing shapes how permanent faults accumulate and materialize as spatially structured corruption in GEMM output tiles.

To capture these effects, this thesis evaluates TCU reliability through cycle-accurate architectural modeling using `PyOpenTCU` [33], an open-source Python-based framework developed as part of this doctoral research.[1] `PyOpenTCU` implements the architectural description of NVIDIA-like TCUs, including: a $4 \times 4$ DPU array (16 DPUs per TCU), configurable input/output buffers (BufferA, BufferB, BufferC, BufferD), a controller managing HMMA instruction sequencing, and support for two TCUs per SM, consistent with the Volta/Ampere generation [11, 12]. From an architectural point of view, `PyOpenTCU` follows the same scheme of a tensor-core-style AI accelerator presented at Figure 2.1. The model supports multiple number formats (FP and Posit) through the SoftFloat and SoftPosit libraries [77], enabling direct format comparison.

### 3.3.1 DPU Structure and Fault Classification

Each DPU within a TCU exposes three structurally distinct signal groups, each directly corresponding to pipeline stages characterized in Section 3.2:

- **IN** — **Input faults**: stuck-at faults on the 9 input signals of each DPU: 4 operands from matrix **A**, 4 from matrix **B**, and 1 from the accumulator **C**. These model faults in the data paths loading operands from the buffers into the DPU's multiplier, equivalent to faults at the input boundary of the *S&E*/*D&C* pipeline stages.
- **PR** — **Product Register faults**: stuck-at faults on the 4 internal product registers that store partial dot-product results between HMMA phases. These model faults in the accumulation flip-flops and correspond to the *N*/*RnD* (FP) and *R*/*EN* (Posit) register stages.
- **OUT** — **Output faults**: stuck-at faults on the 1 output adder signal carrying the accumulated sum to the output buffer. It models faults at the final

[1] https://github.com/TheColombianTeam/PyOpenTCU.git

summation stage, equivalent to the *RnD* stage (*RndAdd*) in FP and the *EN* stage (*Enco*) in Posit.

This classification is the methodological bridge between Section 3.2 and Section 3.3: the stage-level vulnerability profiles from gate-level injection determine which DPU fault classes are most dangerous, while the architectural simulation here quantifies how those faults accumulate and manifest spatially across the full $16 \times 16$ tile. Each fault is modeled as a single stuck-at condition affecting one bit for the entire tile execution, enabling accumulation across HMMA phases.

### 3.3.2 Methodology

The structural vulnerability assessment extends fine-grained arithmetic analysis to the architectural level. Permanent faults are injected at the pin-level abstraction of each DPU. This choice is motivated by three considerations: (*i*) pin-level SAFs are a standard architectural-level abstraction capturing gate-level defects that propagate to the DPU interface [45]; (*ii*) pin-level injection is sufficient to capture how stage-level errors from Section 3.2 accumulate spatially across the full TCU execution; and (*iii*) it avoids the exponential growth of fault sites that would result from targeting every internal gate of the DPU. The limitation of this choice — that structural masking effects *within* DPU stages are not captured — is explicitly addressed by the gate-level analysis of Section 3.2.

For each injected fault, a fault-free reference $16 \times 16$ tile is produced and compared element-wise against the faulty output to classify the result as masked, SDC, or DUE. Spatial error maps are extracted to identify which output positions are corrupted, providing the architectural fingerprints reused by the error model in Section 3.4.

### 3.3.3 Experimental Setup

Structural simulations target two TCUs per SM. Each TCU contains a $4 \times 4$ DPU array. Every DPU exposes 9 inputs (INs), 4 product-register signals (PR), and 1 output (OUT), all 16 bits wide, yielding 57,344 distinct stuck-at faults per campaign. A total of 60 campaigns are executed: 30 for FP16 and 30 for Posit16. Each format is evaluated under three tile-distribution patterns (*Random*, *Zero*, *Triangular*) to capture representative operand behaviors. All campaigns evaluate complete $16 \times 16$ executions (all four HMMA phases). The full campaign required approximately 24 days on an Intel i9 workstation with 20 cores and 32 GB RAM. Figure 3.6 illustrates the evaluation workflow for the micro-architectural and functional campaigns.

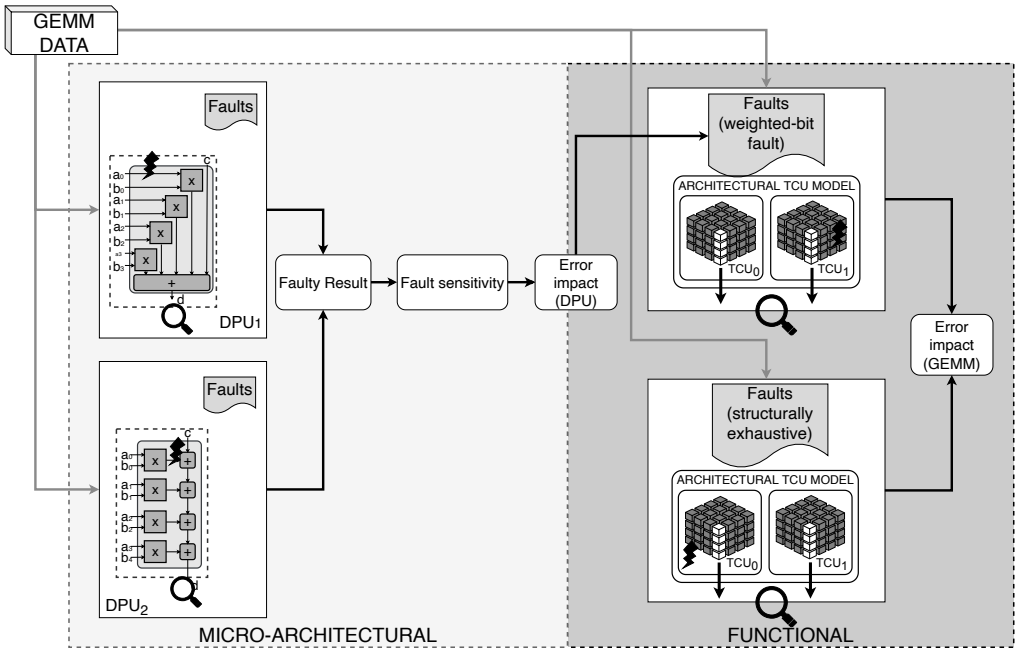



Figure 3.6: Micro-architectural and functional vulnerability analysis workflow: faulty results (from low-level assessment) feeds fault injection at pin level, instruction-accurate TCU execution, and extraction of numerical deviation statistics and spatial error maps [33].

### 3.3.4 Results

**Overall propagation rates.** Figure 3.7 reports the proportions of masked, SDC, and DUE outcomes. Approximately 97% of injected faults produce at least one SDC, about 2% are masked, and roughly 1% result in DUEs. Crucially, DUEs occur exclusively in FP16 execution. The IEEE 754 encoding defines NaN and $\pm\infty$ as explicit exception values; faults affecting the *S&E* stage exponent bits therefore produce detectable invalid states. The Posit number system lacks such special encodings — all bit patterns represent valid real values — so faults produce SDCs across all stages (*D&C*, *FO*, *R*, *EN*) regardless of bit position or fault location.

**Propagation by fault location.** Figure 3.8 reports SDC, DUE, and masked rates per fault location for both formats. The OUT location is the most vulnerable: nearly 99% of faults produce visible SDCs, since the output adder is the last stage before the result is committed to the accumulation buffer, leaving no subsequent masking opportunity. PR and IN faults produce SDCs in approximately 96% of cases, but exhibit a higher DUE rate in FP16 execution: stuck-at faults on bits 13–15 of these signals propagate into the FP exponent field, generating NaN or $\pm\infty$ values. Under Posit16, no such detection mechanism exists, and all fault classes consistently produce SDCs.

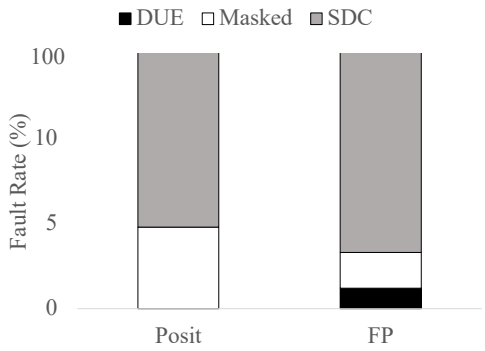



Figure 3.7: Overall proportions of masked, SDC, and DUE outcomes for FP16 and Posit16 TCUs across all fault locations (IN, PR, OUT) [33]. DUEs are exclusive to FP16, where exponent-field corruptions produce IEEE 754 special values (NaN, $\pm\infty$).

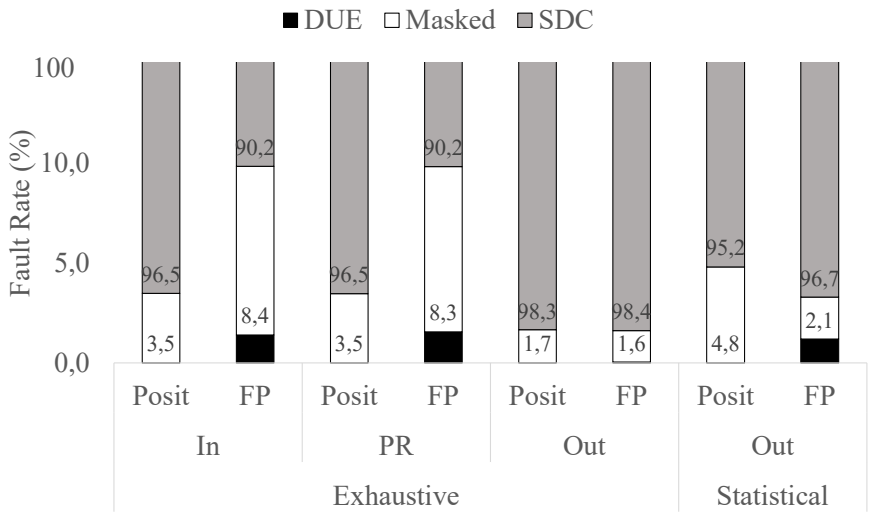
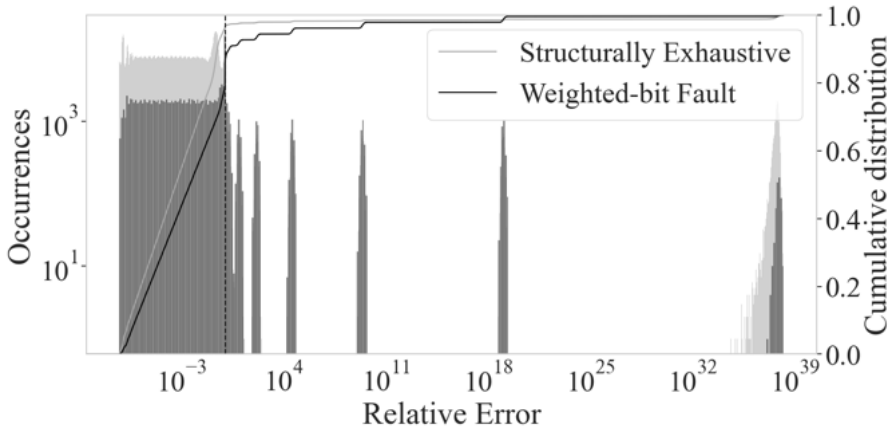


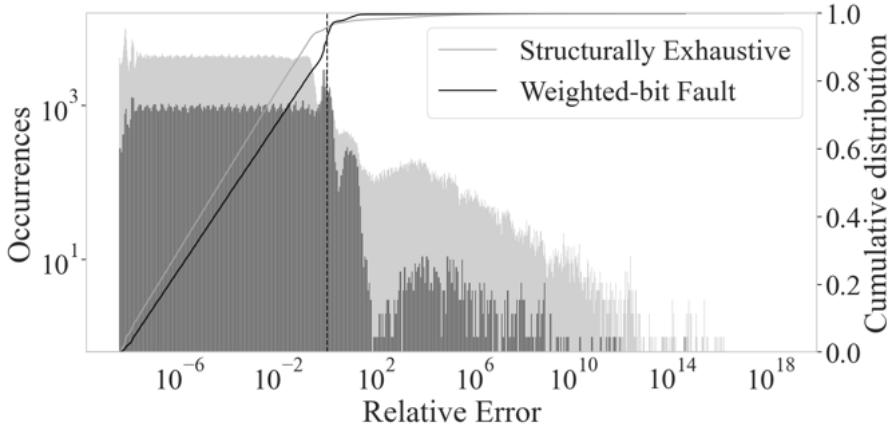

Figure 3.8: Fault classification breakdown per fault location (IN, PR, OUT) for FP16 and Posit16 TCUs [33]. The OUT location has the highest SDC rate (≈99%) since no masking is possible after the final output stage. FP16 PR/IN faults show elevated DUE rates due to exponent-field corruption.

**Error magnitude.** Faults corrupting *S&E* stage bits in FP16 and *D&C/EN* stage bits in Posit16 produce the largest deviations: maxima approach $10^5$ for FP16 and $10^4$ for Posit16, occurring in fewer than 14% of faulty cases. The vast majority of corruptions originate from *SP/FO* stage faults (fraction bits), producing errors below 1.0. Figure 3.9 shows the cumulative distributions.

**Bit-level propagation.** IN and PR faults typically drift into a small neighborhood of output bit positions (within ±3 bits), reflecting carry propagation and




(a) FP16



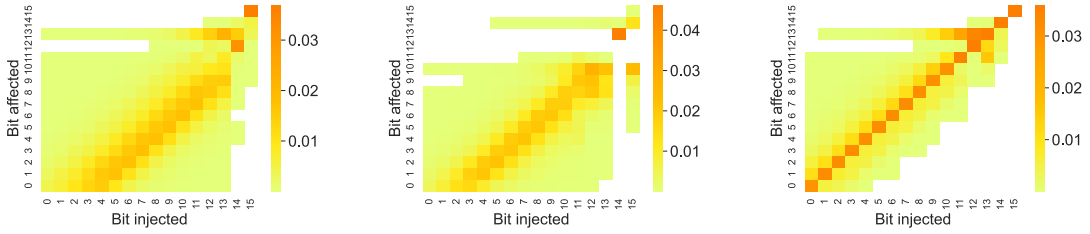

(b) Posit16

Figure 3.9: Cumulative distributions of absolute output errors for FP16 and Posit16 TCUs under permanent faults [33]. Both distributions are bimodal: a dominant cluster below 1.0 (fraction-bit faults) and a sparse tail of large outliers (exponent/regime-bit faults). FP16 exhibits larger extremes due to the wider dynamic range of the exponent field.

normalization effects in the $N/RnD$ and $R/EN$ stages. OUT faults more directly preserve their injection bit position. On average, each permanent fault modifies approximately two output bits.

**Spatial patterns.** Spatial maps reveal deterministic corruption signatures driven by the fixed DPU mapping and HMMA scheduling. A fault in a specific DPU corrupts only the output positions assigned to the warp thread group that uses that DPU fragment. The most common signatures are vertical stripes, diagonal clusters, and multi-cycle stripes that persist across all HMMA phases. Patterns typically affect 2–7 positions per $16 \times 16$ tile (up to 8). Figure 3.10 shows representative

examples.




(a) Mask 1 (b) Mask 2 (c) Mask 3

Figure 3.10: Representative spatial corruption patterns in the $16 \times 16$ output tile produced by permanent DPU faults in FP16 and Posit16 TCUs [33]. Each cell indicates the probability that the corresponding output position $(i, j)$ is corrupted. Patterns are deterministically shaped by warp-to-DPU mapping and HMMA scheduling, producing structured stripes and cluster signatures that are reused by the analytical error model.

### 3.3.5 Summary

Permanent TCU faults propagate with very high probability (≈97% SDC rate) and produce deterministic spatial signatures driven by DPU mapping and HMMA sequencing. FP16 exhibits larger outliers linked to *S&E* stage corruption; Posit16 shows more bounded deviations associated with *D&C/EN* sensitivity; both formats share similar SDC proportions. These architectural fingerprints form the foundation for the analytical model in the next section.

## 3.4 Hardware-Aware Analytical Error Model

The fine-grain gate-level evaluation of Section 3.2 and the structural TCU analysis of Section 3.3 provide detailed, hardware-accurate insight into fault behavior. However, repeating full architectural fault injection for every workload, network layer, or configuration is computationally prohibitive. This section develops an analytical error-impact model that emulates permanent-fault effects on GEMM outputs at a fraction of the cost, while preserving the hardware-induced spatial and numerical signatures that distinguish this approach from hardware-agnostic methods [42, 44].

### 3.4.1 Methodology

The model rests on two empirical properties established by the structural campaigns: (*i*) permanent faults produce repeatable, DPU-index-tied spatial corruption footprints, and (*ii*) numerical deviations follow stage- and bit-dependent statistical distributions that are stable across workloads. Let $D \in \mathbb{R}^{m \times n}$ be the fault-free GEMM output and $\widetilde{D}$ the faulty output. The fault effect is decomposed as:

$$\widetilde{D} = D + E_f, \qquad E_f(i,j) = M_f(i,j) \cdot \Delta_f(i,j), \tag{3.1}$$

where $M_f \in \{0,1\}^{m \times n}$ is a spatial mask and $\Delta_f(i,j)$ is the numerical deviation.

**Spatial mask library.** Masks $M_f$ are extracted from `PyOpenTCU` campaigns for each fault location (IN, PR, OUT) and DPU index, aggregated into a library of template footprints. During emulation, a mask is sampled according to fault location and DPU index and positioned consistently with the tiling and scheduling rules of the target execution.

**Numerical deviation generator.** For each fault location and bit position, the empirical distribution of absolute errors $|\Delta| = |\widetilde{d} - d|$ is stored as a histogram, distinguishing *SP/FO* stage faults (low-magnitude, fraction-field deviations) from *S&E/D&C* stage faults (large-magnitude, exponent/regime-field deviations). When $M_f(i,j) = 1$, a magnitude is sampled from the corresponding distribution, and a sign is assigned from observed statistics.

**Masked/SDC/DUE mixture.** Per-fault probabilities $p_{\text{masked}}$, $p_{\text{sdc}}$, $p_{\text{due}}$ are learned from structural campaigns and applied as a Bernoulli pre-selection before any deviation is injected, preserving the probability of catastrophic DUE outcomes from *S&E/D&C* stage faults.

### 3.4.2 Experimental Setup and Validation

The model is validated against full `PyOpenTCU` fault injection on $16 \times 16$ tiles using operands drawn from the first residual block (RB1) of ResNet-18, corresponding to a GEMM shape of $64 \times 147 \times 12{,}544$ [43]. ResNet-18 RB1 is a representative benchmark for this evaluation because it exhibits the realistic weight and activation value ranges typical of image classification workloads, providing an operand distribution that exercises both the fraction-dominant and exponent/regime-dominant fault pathways. The weight and activation values from this layer populate the input matrices $\mathbf{A}$ and $\mathbf{B}$ of the GEMM execution used for validation. The Mean Absolute Error (MAE) (Equation 2.11 in Chapter 2) is used as the primary scalar comparison metric throughout the validation.

For each format (FP16, Posit16) and each fault location (IN, PR, OUT), two executions are performed: a full `PyOpenTCU` simulation and an analytical emulation. Results are compared on SDC/DUE rates, absolute error distributions, spatial corruption counts, and runtime.

### 3.4.3 Results

Predicted SDC and DUE rates match full fault injection within a few percentage points for both formats. Figure 3.11 compares MAE distributions for a representative ResNet-18 RB1 fault group, showing near overlap between model and simulation. The model captures both the dominant cluster of low-magnitude errors (fraction-bit faults) and the sparse tail of catastrophic outliers (exponent/regime faults). Spatially, the model reproduces the number of corrupted positions per tile and the stripe/cluster structure, because it directly reuses the architectural masks from structural campaigns [43]. The validation achieves up to 93% cross-correlation with full injection campaigns across five different CNN layers.

From a performance standpoint, the analytical model avoids cycle-accurate simulation by applying precomputed masks and sampled deviations directly to fault-free outputs, yielding orders-of-magnitude speedups that enable large-scale reliability studies otherwise infeasible.

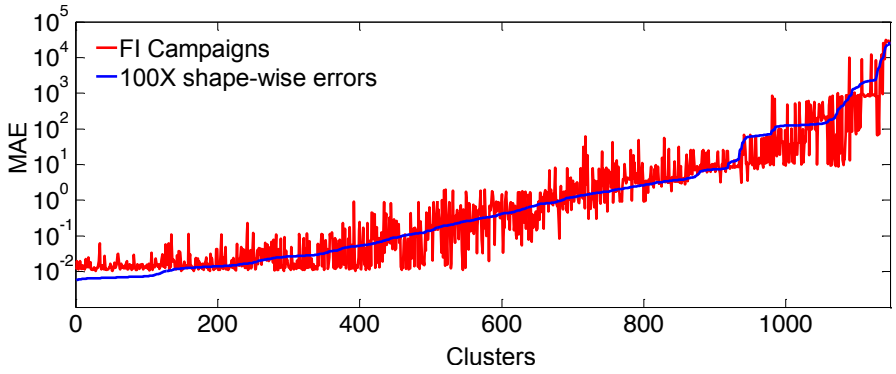



Figure 3.11: Comparison of MAE distributions between full architectural fault injection (`PyOpenTCU`) and the analytical error model for a representative ResNet-18 RB1 layer ($64 \times 147 \times 12{,}544$) [43]. Each x-axis cluster groups fault scenarios with similar spatial and numerical profiles. Near-perfect overlap confirms that the model preserves hardware-accurate error statistics with orders-of-magnitude lower computation cost.

### 3.4.4 Scalability Discussion

The spatial masks are derived for a specific $16 \times 16/4 \times 4$ DPU configuration; different geometries require re-derivation of the masks, though the methodology is general and the derivation is fully automated within `PyOpenTCU`. The numerical deviation distributions are more portable: since they reflect stage-level properties of the arithmetic format (FP16 or Posit16), they transfer to other TCU architectures with only minor recalibration of the *S&E*/*D&C* corruption probabilities.

### 3.4.5 Summary

The analytical error-impact model bridges hardware-level characterization and scalable application evaluation. It accurately reproduces SDC/DUE rates, error magnitudes, and spatial corruption patterns — the three axes of hardware-relevant fault behavior — at a fraction of the cost of full simulation.

## 3.5 Chapter Summary and Discussion

This chapter presented a three-step cross-layer reliability assessment methodology for modern AI accelerators, with emphasis on Tensor Core–style architectures and IEEE 754 FP and Posit arithmetic formats, summarized in the overview of Figure 3.1.

**Step 1 (Section 3.2)** performed exhaustive gate-level FI on 13 synthesized arithmetic cores (ADD, MUL, and MAC for both FP and Posit). The analysis identified the asymmetric vulnerability of pipeline stages: *S&E* and *RnD* in FP cores, and *D&C* and *EN* in Posit cores, are responsible for the large-magnitude catastrophic outliers ($> 10^{30}$ in magnitude) that dominate application-level impact. Fraction-path stages (*SP*/*FO*) produce the majority of SDCs but with magnitudes below 1.0 that are absorbed by network inference resilience.

**Step 2 (Section 3.3)** extended the analysis to complete TCU execution pipelines using instruction-accurate `PyOpenTCU` simulation. Structural campaigns revealed a 97% SDC rate, consistent deterministic spatial error patterns driven by warp-to-DPU mapping and HMMA scheduling, and that FP16 DUEs are exclusively produced by exponent-field corruptions in IN/PR fault locations. The three fault locations (IN, PR, OUT) were defined and mapped to the pipeline stages of Step 1, creating the methodological bridge between the two evaluation levels.

**Step 3 (Section 3.4)** abstracted the structural findings into a hardware-aware analytical model that reproduces SDC/DUE rates, error magnitudes, and spatial signatures with up to 93% correlation against full simulation, while enabling orders-of-magnitude speedups for large-scale reliability studies.

**Positioning.** Compared to hardware-agnostic simulation methods [42], the proposed methodology preserves format-dependent and microarchitectural effects — including deterministic spatial footprints and *S&E*/*D&C* catastrophic-error concentration — that these approaches cannot reproduce. Compared to architectural-injection frameworks such as SASSIFI [36], NVBitFI [38], and gpuFI-4 [72], the proposed approach operates at a finer abstraction level, providing format-aware vulnerability profiles of arithmetic pipeline internals that are invisible at register or instruction level [41]. Compared to existing cross-layer works [73, 74], it uniquely starts from gate-level arithmetic characterization, enabling not only cross-layer propagation analysis but also the format-aware selective hardening strategies

developed in Chapter 4. The industrial EM methodology [44] is opaque and vendor-specific; the proposed framework is fully transparent, reproducible, and adaptable to emerging formats such as Posit, open research designs, and novel TCU geometries. Together, the three steps establish a unified, end-to-end evaluation framework that enables the cost-effective mitigation strategies presented in the next chapters.

# Chapter 4

# Hardware-Based Mitigation Techniques for Arithmetic Units

Modern AI accelerators must combine very high computational throughput with predictable and dependable behavior under increasingly challenging operating conditions. As analyzed in Chapter 3, aggressive technology scaling, massively parallel execution fabrics, and compact numerical formats make arithmetic datapaths progressively more susceptible to permanent hardware faults. Fine-grain reliability evaluation demonstrated that only a limited subset of internal pipeline stages — the *S&E* and *RnD* stages in FP cores, and the *D&C* and *EN* stages in Posit cores — is responsible for the catastrophic numerical outliers and the large majority of dangerous SDCs. Equally important, these critical stages are structurally compact relative to the total core area, making them prime candidates for targeted protection without the prohibitive overhead of blanket redundancy.

This chapter presents the hardware-based mitigation strategies proposed in this thesis, centered on *selective hardening* of FP32 and Posit32 arithmetic units. The methodology builds directly on the vulnerability characterization of Chapter 3: rather than replicating entire cores, it reinforces only the stages and substructures identified as most critical, using three progressive mechanisms — Self-Check and Repair (S-CR), Dual Modular Redundancy (DMR), and Triple Modular Redundancy (TMR) — each offering different tradeoffs between protection strength and implementation cost. The complete evaluation flow, from baseline characterization through hardening decisions to re-evaluation of hardened cores, is summarized in Figure 4.2 at the beginning of Section 4.2.

All mitigation techniques target *permanent hardware faults* modeled as single stuck-at-0/stuck-at-1 faults injected exhaustively at the gate level. This model captures both residual manufacturing defects escaping production testing and wear-out or aging mechanisms (electromigration, BTI, TDDB) that produce persistent logic faults during operational lifetime. The SDC/DUE classification and stimulus sets follow the same criteria established in Chapter 3, maintaining a one-to-one

correspondence between baseline vulnerability profiles and mitigated outcomes that enables fair, direct comparison.

Transient faults are not the primary concern here: while hardened cells may incidentally reduce soft-error sensitivity, systematic runtime protection against transient faults is addressed in Chapter 5. Faults in memories, interconnect, and warp-scheduling logic are also outside the present scope; the focus is strictly on arithmetic-core datapaths, which the fine-grain analysis of Chapter 3 identified as the dominant source of catastrophic numerical corruptions.

The chapter is organized as follows. Section 4.1 reviews the state of the art in hardware mitigation for arithmetic units and motivates the selective hardening approach. Section 4.2 describes the three-phase methodology and the implemented mechanisms. Section 4.3 details the experimental setup. Section 4.4 reports results on SDC reduction, error-magnitude attenuation, and hardware overhead. Section 4.5 summarizes the findings and positions this work with respect to the state of the art.

## 4.1 Motivation and State of the Art

The design of fault-tolerant hardware for arithmetic units has been a long-standing research concern in dependable computing. This section surveys the main families of mitigation techniques, identifies their limitations in the context of AI-oriented arithmetic units, and establishes the rationale for the selective hardening strategy adopted in this thesis.

### 4.1.1 Redundancy-Based Techniques

Triple Modular Redundancy (TMR) offers the strongest protection among classical hardware techniques by triplicating all logic and resolving conflicts through majority voting. Applied at the arithmetic-unit level, it virtually eliminates SDCs caused by single stuck-at faults, including those in the most critical pipeline stages. TMR has been applied to individual GPU compute units [80] and, in combination with software techniques, to scientific workloads running on parallel processors [81]. However, TMR incurs area and power overheads exceeding 200% and complicates timing closure in deeply pipelined accelerators, making it incompatible with the tight resource budgets of AI arithmetic cores.

Dual Modular Redundancy (DMR) reduces this overhead to approximately 100% by duplicating logic and comparing outputs, providing fault detection but not correction. Error-Correcting Codes (ECC) applied to pipeline registers offers single-bit error correction at more modest overhead, but protecting the wide, high-speed datapaths of FP and Posit arithmetic units with full ECC is generally prohibitive in terms of area and critical-path delay.

### 4.1.2 Selective and Precision-Aware Protection

The observation that not all circuit structures contribute equally to application-level faults has motivated several precision-aware and selective protection schemes. Palframan et al. [82] proposed precision-aware soft error protection for GPUs, using the fact that lower-precision datapaths are more tolerant of bit-level errors to apply differential protection. Dos Santos et al. [83] extended this principle with Reduced-Precision DWC for mixed-precision GPU architectures, applying duplication only to high-precision components. Gonçalves et al. [84] applied selective fault tolerance to GPU register files, protecting only the most AVF-critical registers and achieving meaningful reliability improvements at roughly half the overhead of uniform protection. Polian et al. [85] formalized the theoretical foundations of selective hardening, confirming that non-uniform vulnerability distributions in processor logic structures justify concentrating protection resources on the most critical cells.

While these works establish the value of selective protection, they focus on memory structures, register files, or complete processor pipelines, and none provides a format-aware, gate-level methodology for arithmetic core internals of AI accelerators. In particular, none of them distinguishes the *S&E*/*D&C* exponent/regime-path vulnerability from the benign *SP*/*FO* fraction-path behavior that characterizes FP and Posit arithmetic.

### 4.1.3 Algorithm-Based Fault Tolerance

Algorithm-based Fault Tolerance (ABFT) [47] augments matrix kernels with checksum vectors to detect or correct errors at the algorithmic level. Libano et al. [86] demonstrated ABFT for systolic-array GEMM on FPGAs, achieving effective transient fault detection with modest time overhead. Wu et al. [87] extended ABFT to GPU GEMM with online detection and correction, reporting overheads below 9% on average relative to unprotected cuBLAS. Although ABFT is effective for transient faults in dense linear algebra, it does not address the root structural cause of error amplification inside arithmetic cores, requires non-trivial changes to the software or microarchitecture stack, and cannot directly target the exponent/regime pathways responsible for catastrophic outliers.

### 4.1.4 The Need for Selective Structural Hardening

As established in [45], uniform redundancy implicitly assumes all circuit structures are equally critical — an assumption directly contradicted by the fine-grain fault analysis of Chapter 3. Only a small subset of pipeline stages exhibits high fault activation combined with large error magnitude: for FP32 cores, faults concentrated in the *S&E* and *RnD* stages produce errors up to $3.2 \times 10^{38}$; for Posit32 cores, *D&C* and *EN* stage faults produce errors up to $1.3 \times 10^{36}$ [37]. Between 12.5%

and 41.5% of FP faults and 16.1%–39.6% of Posit faults can produce catastrophic effects; conversely, 58.5%–87.5% of all SDCs have magnitude below 1.0 and are absorbed by the intrinsic resilience of CNN workloads [8]. This stark asymmetry motivates *selective hardening*: focusing protection precisely on the pipeline stages responsible for large-magnitude errors, achieving comparable reliability improvement at a fraction of the overhead of full-core protection. Figure 4.1 illustrates the conceptual contrast between full-core TMR and the selective approach.

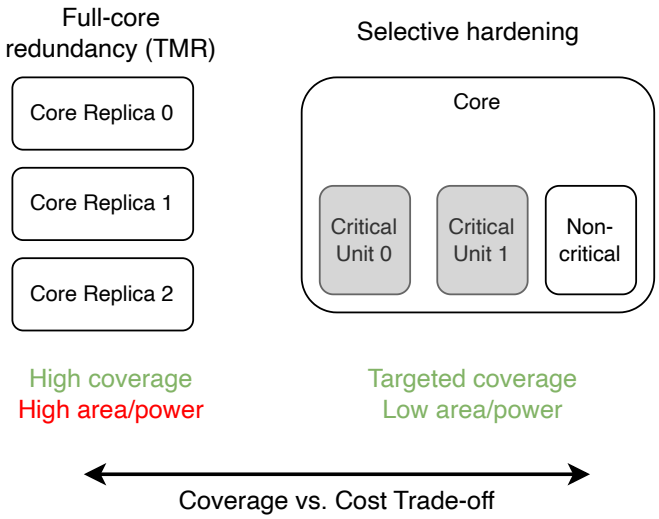



Figure 4.1: Conceptual comparison between full-core TMR and the selective hardening strategy adopted in this thesis. Full TMR (left) replicates all pipeline stages uniformly at >200% overhead, including benign $SP/FO$ stages that produce only low-magnitude errors. Selective hardening (right) concentrates protection exclusively on the high-criticality stages identified by the fine-grain vulnerability analysis — $S\&E/RnD$ for FP and $D\&C/EN$ for Posit — achieving comparable elimination of catastrophic errors at 8–57% overhead depending on core and format.

## 4.2 Methodology for Selective Hardening

The selective hardening methodology is tightly integrated with the cross-layer reliability framework of Chapter 3. It identifies, for each arithmetic core, the minimum set of pipeline stages whose hardening yields the largest reliability improvement — measured by reduction in SDC rate and elimination of large-magnitude errors ($|\Delta| > 1.0$) — within a 15% gate-overhead budget. The methodology proceeds in three phases, summarized by the experimental flow in Figure 4.2.

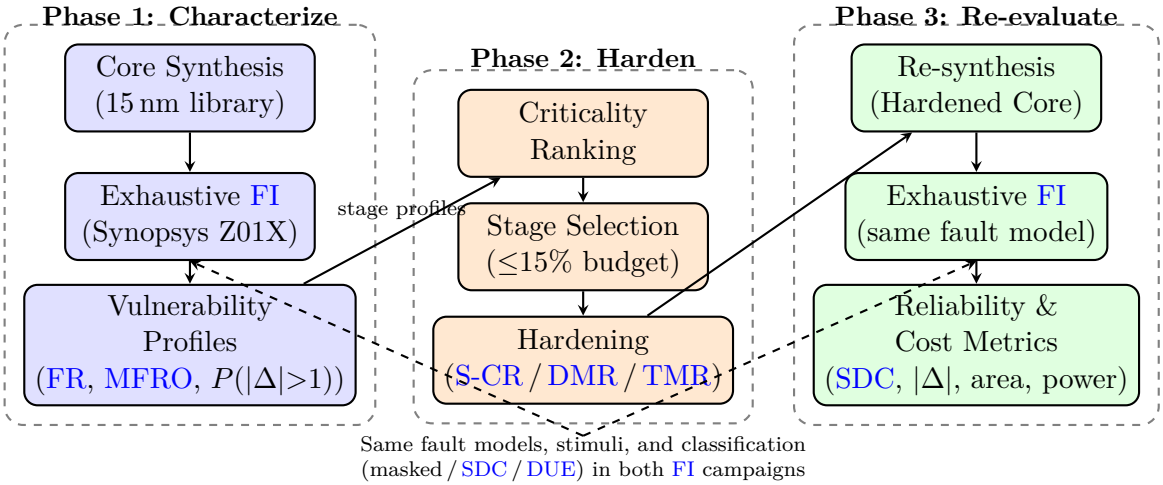



Figure 4.2: Three-phase experimental flow for selective hardening of FP32 and Posit32 arithmetic cores [37]. **Phase 1** derives per-stage vulnerability profiles through exhaustive gate-level fault injection on the baseline cores. **Phase 2** ranks pipeline stages by criticality and implements selective hardening (S-CR, DMR, or TMR) on the top-ranked structures within a ≤15% gate-count budget. **Phase 3** re-synthesizes and re-evaluates the hardened cores under the same fault models, input stimuli, and classification criteria, enabling a direct and fair quantification of reliability gains and hardware cost. The dashed arrows indicate that both fault injection campaigns share identical workloads and fault models.

### 4.2.1 Phase 1: Deriving Structural Vulnerability Profiles

The starting point is the exhaustive fine-grain fault injection of Section 3.2. For each core and each pipeline stage, three metrics are extracted.

The **Fault Rate (FR)** quantifies the fraction of injected permanent faults in that stage that produce at least one corrupted output over the complete stimulus set: it measures how many fault sites are structurally active and dangerous. The **Mean Fault Rate per Operation (MFRO)** [88] measures, for propagating faults, the average fraction of input stimuli for which an SDC occurs: it captures how broadly each dangerous fault affects application execution. The **Tail probability** $P(|\Delta| > 1.0)$ measures the fraction of SDCs in that stage with absolute deviation above 1.0: this is the format-agnostic threshold below which errors are typically absorbed by CNN application resilience [8], so values above this threshold represent genuinely critical corruptions.

These three metrics jointly define a *structural vulnerability profile*. For FP32 multipliers, the *S&E* stage (*ExpAdd*) exhibits high FR and MFRO with $P(|\Delta| > 1.0) \approx 18\%$, since exponent-field corruption shifts output magnitude by powers of the format's base. The *RnD* stage (*RndAdd*) follows with $P(|\Delta| > 1.0) \approx$

12%, because incorrect rounding can overflow into the exponent. The *SP* stage (*MantMul*) has medium FR but only ≈ 2% tail probability, and the *N* stage is negligible (<1%). For Posit32 multipliers, the *D&C* stage (*Deco/Enco*) leads with $P(|\Delta| > 1.0) \approx 14\%$, because regime-bit corruption shifts magnitude by useed$^k$; the *EN* stage follows at 11%, the *R* stage at 4%, and the *FO* stage (*MulFrac*) is negligible (<1%).

### 4.2.2 Phase 2: Criticality Ranking and Hardening Selection

Each stage receives a criticality score combining FR, MFRO, and $P(|\Delta| > 1.0)$ [37]. Stages are ranked from most to least critical, and the top-ranked stages are progressively selected for hardening until the 15% gate-count budget is exhausted. Table 4.1 reports this ranking for the representative FP_MUL and P_MUL cores.

Table 4.1: Criticality ranking and hardening target selection for representative FP32 and Posit32 multiplier cores [37]. The three metrics — FR contribution, MFRO, and tail probability $P(|\Delta| > 1.0)$ — define the criticality score. Stages marked ✓ are selected for hardening within the 15% gate-count budget. Stage nomenclature follows Chapter 3. The *FO/SP* stages are consistently excluded: despite high FR, their tail probabilities are negligible, confirming that fraction-path faults produce only application-benign errors.

| Core | Rank | Stage | Key block(s) | FR | $P(\|\Delta\|>1)$ | Harden? |
|---|---|---|---|---|---|---|
| FP_MUL | 1 | S&E | *ExpAdd* | High | 18% | ✓ |
| | 2 | RnD | *Norm, RndAdd* | High | 12% | ✓ |
| | 3 | SP | *MantMul* | Medium | ≈2% | |
| | 4 | N | Shift logic | Low | <1% | |
| P_MUL | 1 | D&C | *Deco, Enco* | High | 14% | ✓ |
| | 2 | EN | *Reg/Exp Enc., Enco* | High | 11% | ✓ |
| | 3 | R | *FrNrm* | Medium | ≈4% | ✓ |
| | 4 | FO | *MulFrac* | Low | <1% | |

This ranking is stable across the two input ranges (±1.0 and ±10.0), confirming that critical stages are structurally — not operand-— determined. Notably, the selected targets for FP_MUL cover nearly 100% of $|\Delta| > 1.0$ events despite representing only a minority of total fault sites. For P_MUL, the three selected stages together account for over 90% of large-magnitude errors, with the *D&C* stage alone responsible for approximately 60%, due to the global leverage of regime bits over the Posit dynamic range.

### 4.2.3 Phase 3: Hardening Mechanisms and Re-evaluation

Three hardware mechanisms are implemented, offering increasing protection at increasing overhead:

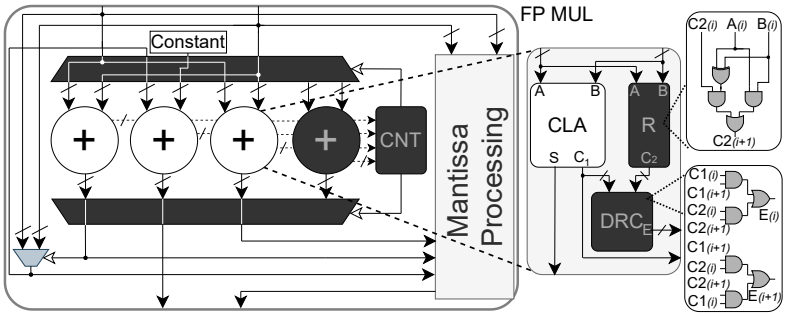
**Self-Check and Repair (S-CR).** The S-CR mechanism combines parity-based detection with cold-spare repair. The critical substructures (e.g., the carry-lookahead adder implementing exponent addition in *S&E*, or the DECO/ENCO logic in *D&C/EN*) are identified as *Basic Blocks*. An independent parity reference circuit (R) computes the expected parity of each block's output; a *Dual Rail Checker* (DRC) detects mismatches; and a controller (CNT) activates a cold-spare unit through input/output multiplexers, replacing the faulty block and restarting the affected pipeline stage. Because spare units are powered and clocked only upon activation, S-CR achieves substantially lower power overhead (0.34×–4× less) than DMR or TMR. The detection-to-correction latency is two additional clock cycles (one for detection and spare activation, one for correction), which is acceptable for pipeline-based arithmetic units.

**Dual Modular Redundancy (DMR).** DMR replicates the critical substructures and uses XOR comparison to detect disagreements. On detection, the redundant output is used for execution. DMR provides detection and limited correction capability at approximately 100% overhead on the duplicated sub-circuit.

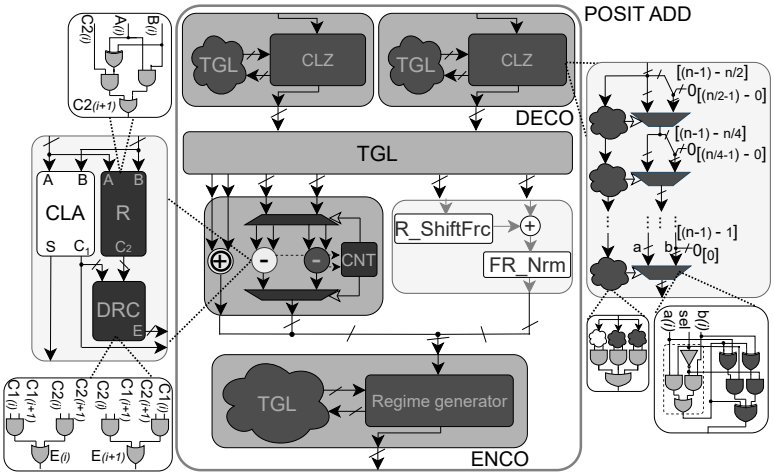
**Triple Modular Redundancy (TMR).** TMR triplicates the critical substructures and resolves conflicts via combinational majority voting. It provides the strongest protection with minimal additional latency (below 15% delay overhead, since voters operate in parallel), but at the highest area and power cost. TMR is applied bit-wise to CLZ counters, multiplexers, and control logic inside the regime-computation path of Posit cores [37].

Figure 4.3 shows the architecture of the hardened FP and Posit cores after applying the S-CR mechanism. Darker-shaded elements are newly added hardening structures; lighter-shaded elements are unchanged baseline logic whose faults produce only low-magnitude errors masked at the application level.

After re-synthesizing the hardened cores under the same 15 nm standard-cell library and synthesis constraints, the exhaustive fault injection campaigns of Chapter 3 are fully repeated on the hardened designs. A hardened structure is considered effective if a fault injected at its inputs or outputs is intercepted and corrected by the S-CR cold-spare (or canceled by the DMR comparator, or resolved by the TMR voter). The primary validation metrics are the reduction in SDC rate and the elimination of $|\Delta| > 1.0$ outliers. The hardened cores preserve the original interface and microarchitecture and are drop-in compatible with any surrounding Tensor Core or accelerator datapath.




(a) FP core: *S&E* and *RnD* stages hardened with S-CR




(b) Posit core: *D&C* and *EN* stages hardened with S-CR + TMR

Figure 4.3: Architecture of hardened arithmetic cores after selective hardening (adapted from [37]). **Darker-shaded elements** are newly added protection structures; **lighter-shaded elements** are unchanged baseline logic. **(a) FP core:** The *S&E* stage is protected by the S-CR mechanism: the carry-lookahead adder (CLA) implementing exponent computation (*ExpAdd*/*ExpSub*) is instrumented with an independent parity reference (R), a Dual Rail Checker (DRC) for mismatch detection, and a controller (CNT) that activates a cold-spare CLA on fault detection. The *SP* stage (*MantMul*) and *N* stage remain unmodified, as their faults produce only low-magnitude errors ($P(|\Delta| > 1) < 2\%$). **(b) Posit core:** The *D&C* and *EN* stages are protected by a combination of the S-CR mechanism on the DECO/ENCO blocks and bit-wise TMR on the CLZ counters and multiplexers in the regime-computation path. The *FO* stage (*MulFrac*) remains unchanged ($P(|\Delta| > 1) < 1\%$).

## 4.3 Experimental Setup

The evaluation closely mirrors the fine-grain campaigns of Section 3.2, extended to include selectively hardened core variants.

**Cores under evaluation.** Eight baseline arithmetic cores implementing FP32 and Posit32 ADD and MUL — FP_ADD, FP_ADD_F, FP_MUL, FP_MUL_F, P_ADD, P_ADD_F, P_MUL, P_MUL_F — and their corresponding hardened variants (suffix "H", e.g., FP_MUL_H, P_ADD_H), synthesized with the 15 nm FreePDK library [78]. The hardening budget is constrained to ≤15% additional gate count per core.

**Fault injection.** Synopsys Z01X, >65,000 stuck-at faults per core, 4,096 stimuli per fault, ModelSim fault-free reference. Baseline and hardened campaigns use identical fault lists, stimuli, and classification criteria (masked / SDC / DUE), enabling direct comparison.

**Workload.** The same 16 × 16 matrix-multiplication tiles, operand ranges (±1.0 and ±10.0), and `numpy/SoftPosit` encoding used in Chapter 3. The ±10.0 range intensifies normalization and regime-decoding activity, stressing the $N/R$ and $D\&C/EN$ stages most relevant to selective hardening.

**Overhead measurement.** Area, power (at 50% switching activity), and critical-path delay are extracted from synthesis reports of baseline and hardened cores under identical constraints. Power analyses used 50% switching activity for consistency.

## 4.4 Results

### 4.4.1 Ranking Validation: Stage-Level Contribution to Large-Magnitude Errors

Before reporting the post-hardening reliability metrics, it is important to confirm that the criticality ranking captures the stages responsible for catastrophic corruptions. Across all evaluated FP32 and Posit32 cores, the ranking is consistent: the two stages selected for FP_MUL (*S&E* and *RnD*) together account for approximately 30% of all baseline SDCs by count, but for essentially all SDCs with $|\Delta| > 1.0$. This confirms the structural concentration of catastrophic errors identified in Chapter 3.

For P_MUL, the three selected stages (*D&C*, *EN*, *R*) cover over 90% of all $|\Delta| > 1.0$ events, with the *D&C* stage alone accounting for approximately 60% due to the global leverage of regime bits over the Posit dynamic range. In stark contrast, the *FO* stage (*MulFrac*) produces a large number of SDCs by count, but with $P(|\Delta| > 1.0) < 1\%$ — confirming that fraction-path faults are application-benign and need not be hardened. This distinction is the core insight that makes selective

hardening possible: the number of SDCs and the severity of SDCs are decoupled across pipeline stages, and only the second metric matters for application-level impact.

### 4.4.2 Impact on SDC Rates and MFRO

Table 4.2 summarizes the baseline and post-hardening SDC rates and large-magnitude error probabilities for all eight cores under the S-CR mechanism.

Table 4.2: Baseline and post-hardening (S-CR mechanism) SDC rates and large-magnitude error probabilities for all evaluated FP32 and Posit32 cores [37]. $\Delta$SDC(%) reports relative SDC reduction; $\Delta P(|\Delta|>1)(\%)$ reports relative reduction in catastrophic outliers. Values are averaged over both operand ranges.

| **Core** | **SDC baseline** | **SDC hardened** | **$\Delta$SDC(%)** | **$P(\|\Delta\|>1)$ base** | **$\Delta P(\|\Delta\|>1)$(%)** |
|---|---|---|---|---|---|
| FP_ADD | 75% | 42% | −44 | 41.5% | −95 |
| FP_ADD_F | 78% | 47% | −40 | 38.2% | −91 |
| FP_MUL | 82% | 38% | −54 | 12.5% | −97 |
| FP_MUL_F | 80% | 40% | −50 | 15.1% | −94 |
| P_ADD | 83% | 24% | −71 | 39.6% | −88 |
| P_ADD_F | 80% | 26% | −68 | 37.4% | −85 |
| P_MUL | 84% | 21% | −75 | 16.1% | −92 |
| P_MUL_F | 78% | 24% | −69 | 18.8% | −90 |

For FP32 adders, hardening the *S&E* and *RnD* stages reduces the SDC rate by 40–44% and the probability of catastrophic outliers by 91–95%. The MFRO of remaining propagating faults also decreases: even when a fault does propagate through the unprotected $SP/N$ stages, it now corrupts a smaller fraction of operations, because the dominant exponent-path activators have been neutralized.

FP32 multipliers show the strongest SDC reduction (50–54%), because the exponent addition unit (*ExpAdd*) has a particularly concentrated criticality profile and dominates large-magnitude outputs. After hardening, extreme deviations above $10^{36}$ effectively disappear ($\Delta P(|\Delta| > 1) = -94\%$ to $-97\%$), and faults with MFRO above 30% become rare.

Posit32 cores benefit most strongly from selective hardening in absolute terms. Reinforcing the *D&C*, *R*, and *EN* stages reduces SDC rates by 68–75% and large-magnitude error probability by 85–92%. The MFRO distributions shift substantially towards lower values, with remaining SDCs concentrated in the *FO* stage and dominated by benign magnitudes ($|\Delta| < 1.0$). The stronger absolute reduction in Posit reflects the initially higher fault activation (driven by the structural complexity of the $D\&C/EN$ stages) combined with effective isolation of those stages through S-CR and TMR.

### 4.4.3 Elimination of Large-Magnitude Errors

The most consequential effect of selective hardening is the dramatic suppression of the error distribution's tail, which is precisely the region that drives classification-accuracy degradation in CNN workloads [8]. In fact, the validation campaigns [37] confirm the following:

In FP32 adders, hardening the *S&E* stage eliminates virtually all outliers above $3.2 \times 10^{38}$ and greatly reduces the mid-range cluster at $\approx 10^{20}$. After hardening, the error distribution becomes single-modal, concentrated below $10^2$, which corresponds primarily to rounding and normalization residuals from the unprotected *SP/N* stages. In FP32 multipliers, errors above $10^{36}$ essentially disappear. In Posit32 cores, regime-bit-corruption deviations — responsible for the distributed high-magnitude cluster from $10^0$ to $10^{30}$ — are eliminated, leaving only the small-magnitude fraction-path errors.

Quantitatively, the fraction of SDCs with $|\Delta| < 1.0$ increases across all cores after hardening: from 58.5% to >95% in FP32 adders, from 87.5% to >97% in FP32 multipliers, from 60.4% to >92% in Posit32 adders, and from 83.9% to >95% in Posit32 multipliers [37]. This confirms that selective hardening surgically removes the critical corruptions while leaving the large population of benign fraction-path deviations unchanged.

### 4.4.4 Hardware Overhead and Cost Analysis

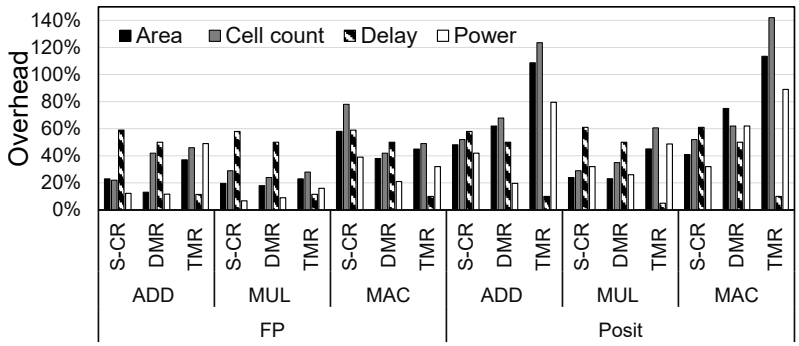



Figure 4.4: Relative overhead costs (area, cell count, power, and critical-path delay) of the S-CR, DMR, and TMR mechanisms for ADD, MUL, and MAC cores in both FP32 and Posit32 formats [37]. Overhead is reported relative to the corresponding baseline core. S-CR achieves the best power tradeoff through cold-spare activation; TMR delivers the lowest delay overhead at the cost of the highest area.

Figure 4.4 reports the relative overhead of all three mechanisms (S-CR, DMR, TMR) across ADD, MUL, and MAC cores in both formats [37]. Several trends emerge from Figure 4.4 and merit discussion.

**Area is higher in Posit than FP across all mechanisms.** This reflects the larger fractional area of the critical stages: the *EN*, *D&C*, and regime-processing structures account for 23.3–54.9% of Posit core area, versus only 8–23% for the *S&E/RnD* stages in FP cores. Consequently, area overhead ranges from 8–80% for FP cores and from 6–142% for Posit cores, depending on operation type and mechanism. For MUL cores — the focus of this evaluation — overhead is 18–25% in FP and 24–45% in Posit, both well within the range acceptable for arithmetic unit integration in a Tensor Core. ADD and MAC cores incur higher costs due to the larger number of structures requiring protection in those pipelines.

**S-CR achieves substantially lower power overhead than DMR or TMR.** The cold-spare units are powered down during fault-free operation; only the parity reference (R) and DRC circuits run continuously. This reduces power overhead by 0.34×–4× compared to TMR, which permanently activates all redundant logic. The S-CR power advantage narrows in MAC cores, where the full-adder-intensive accumulation path requires always-on self-checking circuits [37].

**TMR introduces the lowest timing overhead.** Because the majority of voters are purely combinational and operate in parallel with the replicated logic, the additional delay is below 15% across all evaluated cores and fits within standard timing slack. S-CR and DMR incur approximately 50–60% delay overhead due to the state-machine control needed for spare activation and comparison, but this operates on exception paths rather than the primary data flow, so the impact on sustained throughput is limited.

**Quire Posit MAC requires more elaborate treatment.** The extended 512-bit *QA* accumulator of the *PQ_MAC* core introduces a wide variety of highly sensitive structures that resist efficient adaptation of the three mechanisms under a 15% budget. TMR on the full quire path would require >153% additional area — more than the combined area of the non-quire ADD and MUL cores — indicating that more specialized hardening strategies are needed for this core variant, a direction identified for future work [37].

### 4.4.5 Format-Dependent Observations

The results across all cores confirm the format-specific vulnerability structure identified in Chapter 3, with important implications for hardware design:

FP32 cores are vulnerable to a small set of structurally compact but globally critical exponent-path structures. Two-stage hardening achieves near-complete elimination of catastrophic errors with moderate overhead (18–25% for MUL). This regularity also makes the hardening mechanisms straightforward to implement and validate.

Posit32 cores have a broader set of critical structures across the *D&C* and *EN* stages, driven by the regime/exponent field interaction unique to the Posit

encoding. Hardening three stages achieves stronger absolute SDC reduction (68–75%) but at higher overhead. This suggests that improvements in Posit hardware architecture — particularly in reducing the area fraction and structural complexity of the *D&C*/*EN* stages — would directly reduce the cost of effective mitigation, a direction with implications for future Posit hardware design [37].

These format-specific observations confirm that a one-size-fits-all hardening strategy would be suboptimal. The methodology developed here, grounded in format-specific gate-level characterization, is the necessary foundation for deriving targeted and cost-effective protection for each format.

## 4.5 Chapter Summary and Discussion

This chapter has introduced and evaluated hardware-based selective hardening for FP32 and Posit32 arithmetic units, using vulnerability profiles derived from the fine-grain fault analysis of Chapter 3. The three-phase methodology — characterize, harden, re-evaluate — is summarized by the experimental flow of Figure 4.2, and the resulting architecture of hardened cores is illustrated in Figure 4.3.

The main findings are as follows. Selective hardening of the *S&E* and *RnD* stages in FP32 cores reduces SDC rates by 40–54% and eliminates 91–97% of large-magnitude errors ($|\Delta| > 1.0$), at gate-overhead below 25% for ADD/MUL cores with the S-CR mechanism. Selective hardening of the *D&C*, *R*, and *EN* stages in Posit32 cores achieves even stronger SDC reduction (68–75%) and 85–92% suppression of large-magnitude errors, at somewhat higher overhead (24–57%) reflecting the larger area share of regime/encoding logic. In all cases, the fraction of SDCs with $|\Delta| < 1.0$ increases to above 92–97% after hardening, confirming that the remaining corruptions are application-benign.

From a comparative perspective, the proposed approach achieves a significantly more favorable reliability-per-overhead ratio than full TMR [45]. Compared to ABFT-based techniques [47, 86], it requires no application or microarchitecture changes and directly targets the root cause of catastrophic errors. Compared to precision-aware protection [82, 83], it is grounded in format-specific, gate-level characterization rather than operational precision proxies. The inability to reduce quire Posit MAC overhead below 153% with current mechanisms highlights a genuine design challenge and motivates future work on Posit hardware architecture improvements.

From the cross-layer perspective of this thesis, the vulnerability data produced in Chapter 3 serves triple duty: it drives the gate-level selective hardening of this chapter, informs the analytical error model of Section 3.4, and motivates the structural Tensor Core analysis of Section 3.3. This synergy ensures that mitigation

targets the root causes of numerical unreliability — the exponent and regime pathways responsible for catastrophic outliers — rather than its symptoms. The selectively hardened arithmetic cores developed here provide the protected building blocks for the software-level fault tolerance strategies explored in Chapter 5, where GEMM-level detection and correction are designed with knowledge of the residual error signatures that hardware hardening cannot eliminate.

# Chapter 5

# Software-Level Mitigation Techniques for AI Applications

While hardware-level selective hardening (Chapter 4) directly eliminates the structural root causes of catastrophic numerical outliers inside arithmetic cores, it is fundamentally constrained by design-time choices and silicon cost. Once a device is manufactured, its arithmetic datapaths are fixed: no post-silicon modification can change the vulnerability profiles of the *S&E*, *D&C*, or *EN* pipeline stages identified in Chapter 3. Furthermore, modern GPUs are shared platforms deployed across rapidly evolving workloads and safety requirements; a single device may alternate between best-effort scientific computing, where modest numerical degradation is acceptable, and safety-critical autonomous navigation, where any undetected corruption can be catastrophic. In this context, software-level mitigation provides a complementary and flexible line of defense: it can be deployed in the field without silicon changes, enabled selectively based on the current safety integrity level of the application, and tuned to balance reliability and performance overhead dynamically.

This chapter presents a software-level fault detection and mitigation mechanism for GEMM operations executed on TCUs. The chapter builds directly on insights from Chapter 3: permanent faults in TCUs lead to deterministic and spatially structured output corruptions, driven by the fixed warp-to-DPU mapping and HMMA instruction sequencing. Rather than applying generic kernel duplication that ignores this structure, the proposed mechanism exploits TCU execution semantics — specifically the inherent hardware parallelism from two independent TCUs per SM and the regular HMMA tiling organization — to achieve *hardware-aware* fault detection and mitigation at warp granularity with bounded overhead.

The chapter is organized as follows. Section 5.1 surveys the state of the art in software-level fault mitigation and motivates the proposed approach. Section 5.2 briefly introduces the TCU execution model needed to understand the mechanism. Section 5.3 presents the two-level coarse-grain/fine-grain redundancy mechanism

derived from [46]. Section 5.4 details the experimental platforms and fault injection protocols. Section 5.5 reports static resource overhead, dynamic performance results, and fault coverage. Section 5.6 summarizes key findings and positions the proposed approach within the broader cross-layer reliability framework of this thesis.

## 5.1 Motivation and State of the Art

Software-level fault tolerance has been studied extensively for HPC and safety-critical computing systems. The central appeal is flexibility: software mechanisms can be deployed on unmodified commercial hardware, tuned post-production, and activated selectively. This section surveys the main families of approaches, identifies their limitations for TCU-based GEMM, and establishes the gap addressed by the mechanism proposed here.

### 5.1.1 Software Redundancy Techniques

The simplest class of software fault tolerance is full-kernel redundancy: a GEMM kernel is executed twice on the same or different hardware resources, and the outputs are compared element-wise. Alcaide et al. [89] demonstrated diverse redundancy for GPU kernels by scheduling two kernel replicas on different SMs, achieving high fault detection at the cost of significant performance impact (up to 3× slowdown). Andriotis et al. [90] proposed a software-only approach to enable diverse redundancy on Intel GPUs for safety-related kernels, addressing both permanent and transient faults. While these approaches are effective for detection, they treat the accelerator as a black box: they do not exploit TCU tiling structure, warp-DPU mapping, or HMMA sequencing, so the redundancy cannot be localized below the kernel level. As a result, even a fault affecting only a single DPU and corrupting only a handful of tile positions triggers full-kernel re-execution for mitigation, which is disproportionately expensive.

### 5.1.2 Algorithm-Based Fault Tolerance

Algorithm-based Fault Tolerance (ABFT) [91] augments input matrices with checksum vectors aligned with algebraic properties of matrix operations, allowing output correctness to be verified without re-executing the kernel. ABFT-based methods have been applied to matrix multiplication on GPUs since the early 2010s [92], and more recently optimized for Tensor Core GEMM [87] with online detection and correction. Kosaian and Rashmi [93] demonstrated arithmetic-intensity-guided ABFT for neural network inference on GPUs using the CUTLASS library, reducing overhead to about 17% compared to unprotected operations. Chen

et al. [94] proposed GPU-ABFT for heterogeneous systems, further reducing checksum overhead.

Despite its advantages over full replication, standard ABFT faces two structural limitations in the TCU context. First, ABFT checksum verification operates at the algebraic output level: it detects deviations from the expected linear relationship but cannot identify which DPU within the TCU is faulty or which subset of tile positions is corrupted. This prevents sub-tile localization of faults, requiring full correction whenever any element fails verification. Second, ABFT is sensitive to mixed-precision execution: the FP16 input / FP32 accumulation pipeline introduces rounding differences between the checksum path and the data path, generating false positives unless the threshold $\tau$ is carefully calibrated for each matrix size and input range [46].

### 5.1.3 Testing Library and In-Field Diagnostic Approaches

Hukerikar and Saxena [95] proposed runtime fault diagnostics for GPU Tensor Cores using Universal Test Patterns (UTPs), achieving 92% fault coverage in detecting permanent faults through manufacturing-style test sequences. Ruospo et al. [96] extended this concept to Image Test Libraries (ITLs) for online self-testing of GPU functional units during CNN inference, achieving up to 95% fault coverage for faults in floating-point multipliers. However, these approaches are testing-oriented rather than operational: they interrupt normal execution to apply diagnostic sequences, making them unsuitable for continuous in-field protection during GEMM computation. They also require deep hardware knowledge (e.g., internal schedulers and controllers) for pattern derivation, limiting portability across GPU generations. The Self-Test Library for Tensor Cores developed in [97] follows a similar direction and achieves structured test coverage using matrix multiplication stimuli, but again targets offline or periodic testing rather than transparent in-operation protection.

### 5.1.4 Hardware-Based Detection and Range Checking

Hafezan and Atoofian [98] proposed a hardware-based fault detection strategy for Tensor Cores (FDTC) that exploits data sparsity in ML applications to introduce spatial redundancy on sparse operands, ensuring correctness of non-sparse computations. The approach offers near-zero runtime overhead by leveraging idle hardware cycles. Fang et al. [99] explored MPGemmFI, a fault injection tool for mixed-precision GEMM, and showed that hardware range-check strategies can improve model accuracy by up to 75% in fault-prone scenarios. Both approaches, while effective, require hardware architectural modifications that are infeasible on commercial off-the-shelf GPUs.

### 5.1.5 Algorithm-Based Error Detection at CNN Level

Hari et al. [100] proposed an algorithm-based error detection (ABED) for CNN convolution through output checksum verification. Their approach can detect all transient fault impacts in convolution layers but introduces up to 23% runtime overhead and provides no correction capability; recovery requires re-execution of the affected layer. Bolchini et al. [101] proposed fast error simulation for CNNs combining architectural error models with application-level injection, enabling rapid reliability assessment but not in-operation protection.

### 5.1.6 Positioning the Proposed Approach

The survey reveals a clear gap: existing software-level techniques either incur excessive overhead (full kernel redundancy, $\approx 3\times$ slowdown) or cannot localize faults at sub-tile granularity (ABFT, ABED). Hardware-based approaches achieve low overhead but require silicon modifications. None of the surveyed works exploits the intrinsic hardware parallelism of TCUs — the availability of two independent TCUs per SM and the programmable HMMA tiling structure — to perform detection and localized mitigation entirely in software with bounded overhead. The structural fault signatures identified in Chapter 3 — specifically, that permanent DPU faults produce deterministic, stripe-like or cluster-like corruption patterns over a small, predictable subset of the $16 \times 16$ output tile — make sub-tile localization both feasible and efficient. The proposed mechanism [46] fills this gap by exploiting warp-level temporal and spatial redundancy at two levels of granularity.

## 5.2 TCU Execution Model and Fault Signatures

Before describing the proposed mitigation mechanism, it is necessary to briefly recall the TCU execution model introduced in Chapter 2 and the fault signatures established in Chapter 3, which together motivate the two-level design of the mechanism.

**TCU execution model.** A TCU comprises 16 DPUs arranged in a $4 \times 4$ array. Each DPU computes one or more multiply-accumulate ($A \cdot B + C$) operations per HMMA instruction. A complete $16 \times 16$ output tile is produced through four sequential HMMA phases, each phase assigning specific input slices to specific DPUs. The mapping from warp thread groups to DPUs is fixed for a given GPU architecture and tile configuration. Two TCUs per SM operate independently and can process different data in parallel [11, 12].

**Fault signatures from Chapter 3.** As established in Section 3.3, permanent faults in TCUs produce deterministic spatial corruption patterns. A fault in a specific DPU (IN, PR, or OUT class) corrupts only the output positions assigned to the warp thread group that uses that DPU slice. Typical corrupted footprints

are stripe-like or cluster-like, affecting 2–7 elements per $16 \times 16$ tile (up to 8). Because the DPU mapping is fixed, the same fault produces the same corruption pattern in every execution of the same tile configuration. This *determinism* is the key property exploited by the software mechanism: if the two TCUs per SM receive identical inputs but a fault affects only one, the outputs will differ at exactly those positions corrupted by the faulty DPU, enabling localization.

## 5.3 Methodology: Two-Level Hardware-Aware Software Mitigation

The proposed mechanism [46] operates at the *warp level* inside the GEMM kernel, exploiting both temporal and spatial redundancy within the existing TCU resources. The mechanism uses two levels of granularity: *coarse-grain redundancy* for always-on detection, and *fine-grain redundancy* for conditional localized mitigation. Figure 5.1 shows the execution timeline for both levels.

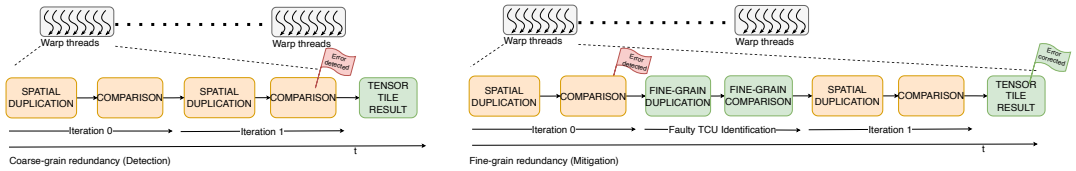



Figure 5.1: Timeline of the proposed two-level fault tolerance mechanism for Tensor Core GEMM. *Left*: Coarse-grain redundancy for fault detection — each warp tile $TM \times M$ is split into two halves ($HM_1$, $HM_2$), each executed twice in parallel on the two TCUs per SM, and the results are compared. *Right*: Fine-grain redundancy for fault mitigation — activated only when a coarse-grain mismatch is detected; each half is further subdivided into quarters ($QM_{x1}$, $QM_{x2}$), which are executed with Quadruple Modular Redundancy to identify and isolate the faulty TCU slice. The correct result is reconstructed using fault-free slice outputs. Adapted from [46].

### 5.3.1 Coarse-Grain Redundancy for Detection

The coarse-grain mechanism splits each warp-level tensor tile $TM \times M$ into two halves, $HM_1$ and $HM_2$, processing them in two consecutive iterations (temporal redundancy). Within each iteration, both halves are spatially duplicated: the same half is executed on two independent execution paths mapped to the two TCUs per SM (spatial redundancy), producing two identical results $HM_{xl}$ and $HM_{xr}$ from $\text{TCU}_0$ and $\text{TCU}_1$, respectively. The deterministic DPU-to-warp mapping ensures that a permanent fault in $\text{TCU}_0$ consistently corrupts the same positions of $HM_{xl}$, while $HM_{xr}$ (from the fault-free $\text{TCU}_1$) remains correct.

After each iteration, the two results are compared element-wise in registers. A detection flag is raised if any element exceeds a threshold $\tau$:

$$\exists(i,j) : \left| HM_{xl}^{(i,j)} - HM_{xr}^{(i,j)} \right| > \tau. \tag{5.1}$$

For permanent faults, deviations are deterministic and typically large-magnitude (consistent with the fault signatures of Chapter 3), so $\tau$ can be set conservatively to suppress rounding-induced false positives in mixed-precision execution. If no mismatch is detected in either iteration, the partial results from the two halves are merged into the final tile $D = HM_1 \oplus HM_2$, and the computation continues without overhead beyond the comparison. Figure 5.2 illustrates this detection flow for a case where $\text{TCU}_0$ is affected by a hardware fault.

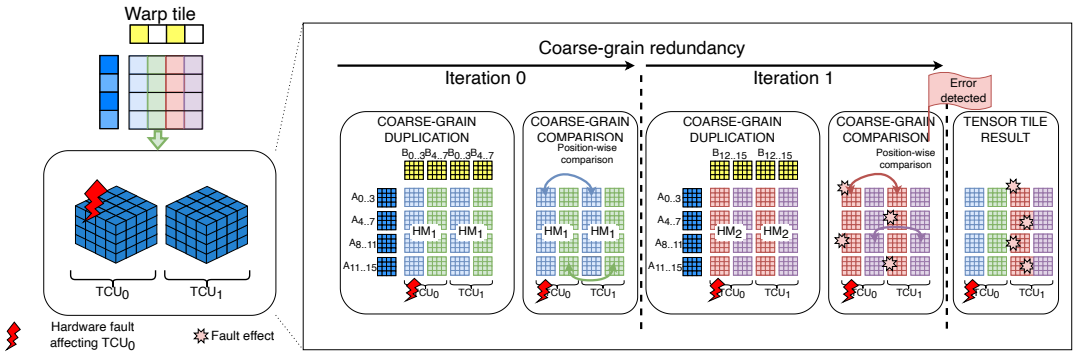



Figure 5.2: Coarse-grain fault detection for a $16 \times 16$ tile with $\text{TCU}_0$ affected by a permanent fault (adapted from [46]). The warp tile is split into halves $HM_1$ (blue/green slices) and $HM_2$ (red/pink slices). Each half is spatially duplicated across $\text{TCU}_0$ and $\text{TCU}_1$. The comparison of $HM_{xl}$ and $HM_{xr}$ reveals the fault **late** (after the second iteration) if the corrupted positions fall in $HM_2$, or **early** (after the first iteration) if they fall in $HM_1$. The detection flag triggers fine-grain mitigation.

### 5.3.2 Fine-Grain Redundancy for Mitigation

When the coarse-grain comparison raises a detection flag, the kernel activates the fine-grain redundancy path. This path identifies *which slice* of the TCU is faulty and reconstructs a correct output using the fault-free slices.

Fine-grain redundancy subdivides each half $HM_x$ into two quarters $QM_{x1}$ and $QM_{x2}$, effectively slicing the $16 \times 16$ TCU configuration into four logical partitions aligned with the DPU mapping. Each quarter is executed four times in Quadruple Modular Redundancy (QMR) mode — four independent execution paths mapped to different TCU resource slices. The comparison stage of QMR identifies which quarter produced an incorrect result: the partition corresponding to the faulty

DPU consistently disagrees with the three fault-free replicas. Once identified, the faulty TCU partition is excluded, and the correct final tile is assembled from the fault-free quarter results:

$$\widehat{D} = \text{Select}(QM_{11}^{(\text{free})}, QM_{12}^{(\text{free})}, QM_{21}^{(\text{free})}, QM_{22}^{(\text{free})}), \tag{5.2}$$

where Select$(\cdot)$ chooses the fault-free result from each quarter, as identified by the QMR comparison stage. Because a single permanent DPU fault typically corrupts only one of the four partitions, three of the four QMR replicas agree, enabling correct reconstruction without a full tile re-execution. Figure 5.3 illustrates the conditional fine-grain flow.

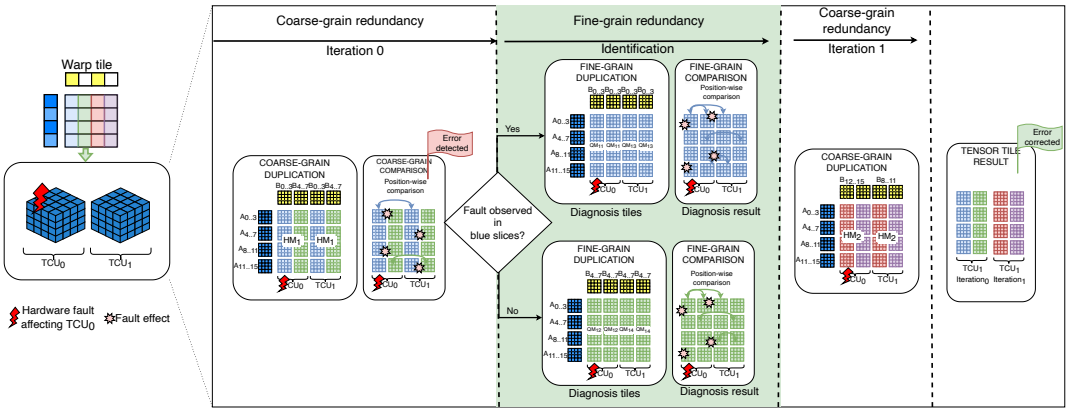



Figure 5.3: Fine-grain fault mitigation for a $16 \times 16$ tile where coarse-grain detection flagged a mismatch (adapted from [46]). Each half $HM_x$ is further subdivided into quarters ($QM_{x1}$, $QM_{x2}$). Quadruple Modular Redundancy (QMR) executes four replicas of each quarter on different TCU slices. The comparison stage identifies the faulty partition (marked in red), and the correct tile is reconstructed from the three fault-free quarter results.

### 5.3.3 Design Properties

Several design properties are worth highlighting. First, **the detection overhead is always-on but bounded**: coarse-grain comparison adds register-level comparisons after each HMMA iteration, with no additional global-memory traffic. Second, **mitigation is conditional**: fine-grain QMR is activated only when a fault is detected, so fault-free execution incurs only the detection overhead. Third, **the mechanism handles both permanent and transient faults**: permanent faults trigger consistent mismatches in coarse-grain detection (because the same DPU fault fires on every HMMA instruction); transient faults, which flip a result in a single dynamic instance, also trigger the comparison if their magnitude exceeds $\tau$. Fourth, **scalability is size-independent**: because the mechanism operates at tile

granularity within the kernel, the overhead does not scale with the GEMM matrix size — both detection and mitigation perform the same fixed amount of additional work per tile, regardless of the overall problem dimensions. This size-independence is a key advantage over application-level ABFT methods, whose checksum overhead depends on matrix dimensions.

## 5.4 Experimental Setup

The software mechanism is evaluated on two NVIDIA platforms: a discrete GPU (GeForce RTX 3060 Ti, Ampere architecture, 4,864 CUDA cores, 8 GB GDDR6) and an embedded platform (Jetson AGX Orin 64 GB, Ampere GPU, 2,048 CUDA cores). Both support FP16 inputs with FP32 accumulation in Tensor Core HMMA operations. The two platforms differ in SM count, warp schedulers, and shared-memory size, enabling assessment of the mechanism's behavior across both high-performance and embedded deployment targets.

**Kernel variants.** Three CUDA kernel variants are evaluated. The **baseline** kernel is the non-protected WMMA-based Tensor Core GEMM without any redundancy. The **detection-oriented version** adds coarse-grain tile-level redundancy (Section 5.3.1) and register-level comparison with threshold $\tau$. The **fault-tolerant version** extends the detection-oriented version with the full conditional fine-grain QMR path (Section 5.3.2). All three variants use the same tiling configuration, memory layout, and WMMA API calls to ensure that measured differences are attributable to detection/mitigation logic rather than to different data movement strategies.

**Workloads.** Square GEMM operations with sizes ranging from $256 \times 256$ to $16{,}384 \times 16{,}384$ in increments of 256, with uniformly distributed random inputs. This range covers both small matrices (where occupancy and overhead effects are prominent) and large matrices (where the mechanism's size-independence becomes apparent).

**Performance profiling.** Static metrics (registers per thread, shared memory per block, theoretical warp occupancy, SASS instruction count) are collected from NVIDIA Nsight. Dynamic metrics (executed instruction count, kernel duration in cycles, SM throughput, memory throughput, and achieved occupancy) are measured using NVIDIA Nsight profiling. Energy consumption is measured using NVML, averaged over 30 trials per configuration with GPU pre-conditioning (a large warm-up GEMM) to minimize thermal transient effects.

**Fault injection.** Fault coverage is evaluated using two complementary tools, consistent with Chapter 3:

**NVBitFI** [38]: a dynamic binary instrumentation tool built on NVBit [67] that injects transient single-bit flips into architectural registers at randomly selected dynamic instances during kernel execution. The injection targets registers holding

Table 5.1: Static resource usage of the three kernel variants on both platforms. Adapted from [46].

| Kernel variant | Platform | Shared mem. (KB) | Regs/thread | Max. warp occ. | SASS instr. |
|---|---|---|---|---|---|
| Baseline | RTX 3060 Ti | 18 | 40 | 40 | 280 |
| | AGX Orin 64 GB | 18 | 40 | 48 | 328 |
| Detection-oriented | RTX 3060 Ti | 18 | 56 | 40 | 456 |
| | AGX Orin 64 GB | 18 | 56 | 48 | 488 |
| Fault-tolerant | RTX 3060 Ti | 19.53 | 64 | 32 | 1,600 |
| | AGX Orin 64 GB | 19.53 | 64 | 32 | 1,632 |

HMMA outputs, emulating faults in TCU arithmetic datapaths (DPU gates or flip-flops) that manifest at the instruction output level. For transient fault evaluation, 1,442 single-bit flips are injected during $256 \times 256$ GEMM execution; this sample size provides a 5% margin of error at 95% confidence following the statistical methodology of [102].

**NVBitPerFI** [69]: extends NVBitFI to inject *permanent* stuck-at faults by forcing a stuck-at condition on a selected register bit that persists across all dynamic instances of the targeted instruction throughout kernel execution. This model's permanent defects in TCU DPU gates or flip-flops — the fault class evaluated throughout this thesis — manifest at HMMA output registers. For permanent fault evaluation, 1,024 single-bit stuck-at faults are injected into the HMMA output registers. Register file and shared memory faults are not the primary focus, as these structures are typically protected by ECC in modern NVIDIA Ampere GPUs [12].

## 5.5 Results

### 5.5.1 Static Resource Overhead

Table 5.1 summarizes the static resource usage of all three kernel variants on both platforms. The baseline kernel uses 40 registers per thread and 18 KB shared memory per block. The detection-oriented version increases register usage to 56 per thread (40% increase) while keeping shared memory unchanged, because coarse-grain redundancy and comparison are performed entirely in registers. The fault-tolerant version increases register usage further to 64 per thread (60% increase) and adds 1.53 KB shared memory per block (total: 19.53 KB, corresponding to 8% increase) to support the fine-grain QMR reconstruction path. The additional shared memory reduces maximum warp occupancy from 40 to 32 warps per SM on the RTX 3060 Ti, and from 48 to 32 on the Jetson AGX Orin. Code size (SASS instruction count) grows from 280/328 (baseline) to 1,600/1,632 (fault-tolerant), reflecting the fine-grain QMR path; the detection-only version grows to 456/488 instructions, a modest 1.5–1.6× expansion.

### 5.5.2 Dynamic Performance Overhead

Figure 5.4 shows the normalized executed instruction count for all kernel variants across GEMM sizes. The detection-oriented version increases instruction count by 4%–25% for small matrices, stabilizing at approximately 9% overhead on RTX 3060 Ti and 4% on Jetson AGX Orin for matrices larger than $\approx 4{,}096 \times 4{,}096$. This convergence confirms the size-independence property of the mechanism: the per-tile redundancy overhead is fixed, so amortization over larger matrices is limited and the overhead plateau represents the steady-state cost. The instruction overhead does not double despite redundant HMMA execution because most memory operations remain unchanged (input tile data is reused, not re-fetched), and the dominant incremental cost comes from register-level comparisons and control-flow decisions.

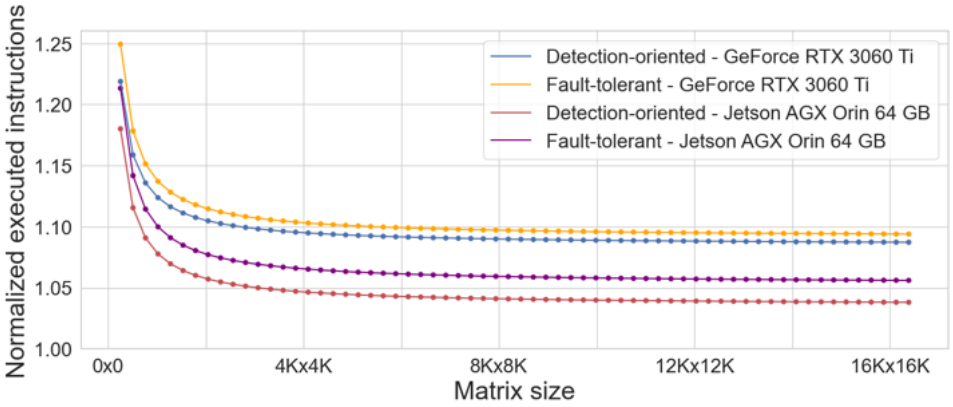



Figure 5.4: Normalized executed instruction count for the detection-oriented and fault-tolerant kernel variants relative to the baseline, across GEMM sizes on RTX 3060 Ti and Jetson AGX Orin 64 GB [46]. Overhead stabilizes at ≈9% (RTX) and ≈4% (Orin) for large matrices.

Memory throughput remains high for all variants (Figure 5.5), with a maximum deviation below 4% relative to baseline. This confirms that the coarse-grain and fine-grain redundancy paths add arithmetic/comparison work in registers but do not increase global-memory pressure.

SM throughput trends (Figure 5.6) show that the baseline GEMM saturates Tensor Core resources while leaving integer and control-flow execution units underutilized. The detection-oriented and fault-tolerant variants introduce additional comparison operations that partially exploit these idle resources, limiting the throughput reduction relative to what would be observed if the SM were fully saturated.

End-to-end kernel execution time overhead is bounded and largely size-independent across both platforms (Figure 5.7), reaching at most 13% on the RTX 3060 Ti and

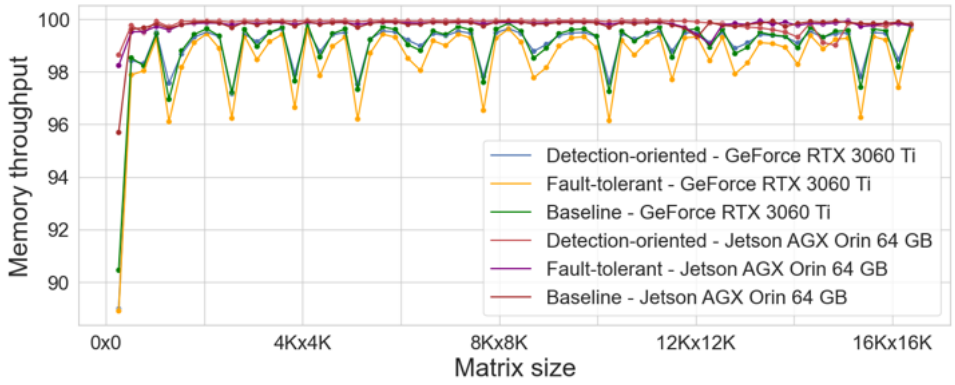



Figure 5.5: Memory throughput of baseline and hardened kernel variants across GEMM sizes. The small deviation (<4%) from baseline indicates that the redundancy paths do not introduce additional global-memory traffic [46].


Figure 5.6: SM throughput of baseline and hardened kernel variants. The comparison and control-flow operations introduced by the mechanism partially exploit integer execution units left idle by the baseline GEMM, limiting SM throughput reduction [46].

11% on the Jetson AGX Orin. This confirms that the mechanism's overhead does not grow with problem size, making it practical for large-scale deployments. Energy measurements follow the same trend, with a maximum increase of approximately 15% on both platforms (Figure 5.8). The consistent behavior across the two platforms — one high-performance discrete GPU and one embedded device — demonstrates that the mechanism is portable and suitable for both safety-critical embedded and HPC domains.

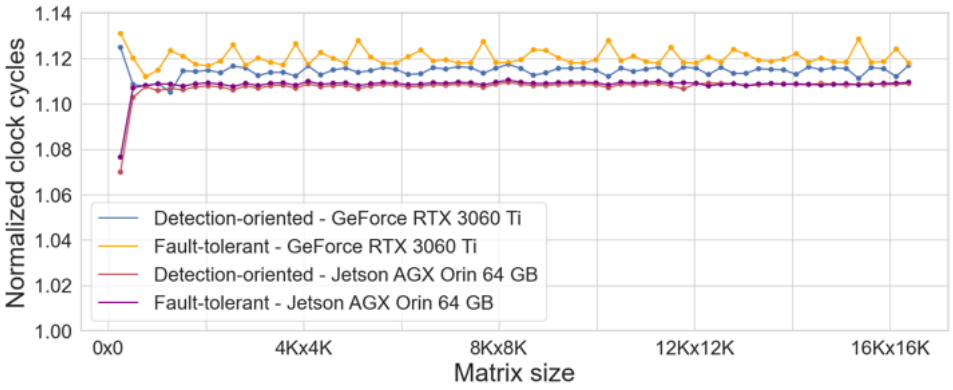



Figure 5.7: Normalized end-to-end kernel execution time (cycles) relative to baseline across GEMM sizes [46]. Overhead is bounded at 13% (RTX 3060 Ti) and 11% (Jetson AGX Orin) and remains largely size-independent.


Figure 5.8: Normalized energy consumption of hardened kernel variants relative to baseline across GEMM sizes [46]. Maximum energy increase is approximately 15% on both platforms.

### 5.5.3 Fault Coverage

Fault injection campaigns validate the ability of the fault-tolerant version to detect and mitigate faults manifesting at HMMA output registers.

**Permanent fault campaign.** All 1,024 injected stuck-at faults into the HMMA output registers are correctly detected and corrected by the fault-tolerant kernel. The coarse-grain comparison reliably flags mismatches on the first (early detection) or second (late detection) iteration depending on which half ($HM_1$ or $HM_2$) contains the corrupted positions. Fine-grain QMR then isolates the faulty TCU partition and reconstructs the correct output from the three fault-free quarters. No false corrections are observed, confirming that the threshold $\tau$ is appropriately calibrated for the test input distribution.

**Transient fault campaign.** All 1,442 single-bit flip injections are also detected and corrected. Transient faults affect only a single HMMA dynamic instance, producing a one-time mismatch in the coarse-grain comparison. Because transient faults are not persistent, fine-grain QMR re-executes the affected quarter and obtains a consistent, fault-free result in all four replicas, confirming the absence of the fault and enabling correct reconstruction. The end-to-end cycle overhead for transient fault scenarios remains within the bounded maximum (13%), since mitigation is activated at most once per tile and the QMR execution is brief.

**Summary.** The 100% detection and correction rate for all injected permanent and transient faults, combined with the bounded overhead of at most 13% in execution time and 15% in energy, validates the fault-tolerant kernel as an effective in-field protection mechanism for Tensor Core GEMM. The results further confirm that conditional mitigation — fine-grain QMR activated only upon detection — does not cause runaway performance degradation even under persistent permanent faults, which would trigger mitigation on every tile containing the affected DPU's output positions.

## 5.6 Chapter Summary and Discussion

This chapter presented a software-level fault detection and mitigation mechanism for Tensor Core GEMM execution, based on the work in [46]. The mechanism exploits the inherent hardware parallelism of TCUs — two independent TCUs per SM and the programmable HMMA tiling structure — to implement hardware-aware coarse-grain/fine-grain redundancy at warp granularity.

The two-level design addresses the gap identified in the state-of-the-art survey (Section 5.1): unlike full-kernel replication [89, 90], it localizes fault effects at sub-tile granularity rather than duplicating the entire computation; unlike algebraic ABFT [91, 93], it does not require checksum verification paths sensitive to mixed-precision rounding, and its overhead is size-independent; unlike testing approaches [95, 96, 97], it provides transparent in-operation protection without interrupting normal execution.

Evaluated on the RTX 3060 Ti and Jetson AGX Orin 64 GB platforms, the fault-tolerant kernel achieves 100% detection and correction for both permanent (1,024 campaigns) and transient (1,442 campaigns) fault injections into HMMA output registers, with bounded overhead: 40% more registers, 8% more shared memory, up to 13% longer execution time, and up to 15% more energy. The overhead is size-independent, converging to approximately 9% (RTX) and 4% (Orin) for large-scale GEMMs.

**Cross-layer positioning.** This chapter completes the cross-layer reliability framework developed throughout this thesis. Chapter 3 established that permanent

TCU faults produce deterministic spatial signatures and characterized their numerical impact across pipeline stages — observations that enable both the hardware hardening of Chapter 4 and the sub-tile fault localization of this chapter. Chapter 4 eliminated the structural root causes of catastrophic outliers inside arithmetic cores through selective hardening of $S\&E/D\&C$ and $RnD/EN$ stages. The software mechanism of this chapter complements hardware hardening in two ways: it provides a post-silicon, in-field layer of protection that can be activated on commercially available devices where hardware modifications are not possible; and it addresses a broader fault spectrum (including transient faults) that hardware-only hardening does not cover. Together, the three layers — hardware-accurate characterization, selective gate-level hardening, and warp-level software redundancy — form a coherent, multi-level strategy for reliable AI accelerator design.

**Residual limitations and future directions.** The mechanism targets faults that manifest at HMMA output registers; faults in shared memory, register-file load/store paths, or warp-scheduling logic are outside its scope. Extending the detection and mitigation to these structures remains an open challenge. Additionally, the threshold $\tau$ in Equation 5.1 currently requires calibration for each platform and input distribution to avoid false positives under mixed-precision rounding; adaptive threshold determination — possibly informed by the error magnitude profiles of Chapter 3 — is a promising direction. Finally, the current implementation uses a fixed partition scheme for fine-grain QMR; leveraging the spatial mask library of Section 3.4 to adapt the partition scheme to the predicted fault footprint of a detected DPU fault could further reduce the QMR overhead in future versions.

# Chapter 6

# Conclusions

This thesis addressed the reliability challenges of modern AI-oriented hardware accelerators, which arise from the convergence of aggressive semiconductor scaling, massive parallelism, compact numerical formats, and the growing deployment of machine-learning workloads in safety-critical and high-availability environments. The central objective was to develop reliability assessment methodologies that are simultaneously accurate, hardware-aware, and scalable across abstraction levels, and to use the resulting structural insights to design lightweight but effective mitigation strategies at the hardware, software, and accelerator microarchitecture levels.

The work was organized around a three-step cross-layer framework linking gate-level arithmetic analysis, accelerator architectural evaluation, and scalable error modeling, complemented by mitigation strategies at two distinct levels.

## Contributions

**Fine-grain reliability characterization of FP and Posit arithmetic units.** The first contribution consists of exhaustive gate-level fault injection campaigns on 13 synthesized arithmetic cores implementing addition, multiplication, and MAC operations in FP32 and Posit32 formats. All cores were synthesized under identical 15 nm conditions, with more than 65,000 stuck-at faults evaluated per core and 4,096 input stimuli per fault. The analysis revealed that fault propagation is strongly dominated by internal circuit structure rather than by operand distribution alone. Posit cores activate a larger fraction of internal faults (70%–89% FR for multipliers) and generate broader SDC distributions due to the structural complexity of their *D&C* and *EN* stages, which implement variable-length regime decoding and re-encoding. In contrast, FP cores produce fewer corruptions overall but generate substantially larger catastrophic outliers — up to $3.2 \times 10^{38}$ — exclusively driven by faults in the *S&E* and *RnD* stages that corrupt exponent-determining logic. Across all formats and operations, 58.5%–87.5% of all SDCs

have magnitude below 1.0 and are application-benign for CNN workloads, while the remaining large-magnitude errors are concentrated in a small set of structurally specific pipeline stages. This quantitative asymmetry constitutes the foundational insight for all subsequent mitigation work.

**Structural vulnerability analysis of Tensor Core execution pipelines.** The second contribution extends the analysis from isolated arithmetic cores to complete Tensor Core (TCU) execution pipelines using `PyOpenTCU`, a cycle-accurate, instruction-aware architectural model developed as part of this research. A total of 60 fault-injection campaigns (30 for FP16, 30 for Posit16) were conducted, evaluating 57,344 permanent stuck-at faults per campaign across the three structural fault classes — IN, PR, and OUT — targeting the input ports, internal product registers, and output node of each DPU in the $4 \times 4$ array. Results show that approximately 97% of injected faults produce at least one SDC, confirming that TCUs are highly sensitive to permanent defects. Critically, the resulting corruptions are spatially deterministic: a given DPU fault consistently corrupts the same subset of output positions (typically 2–7 elements per $16 \times 16$ tile), producing characteristic stripe-like or cluster-like patterns governed by fixed warp-to-DPU mapping and HMMA scheduling. DUEs occur exclusively in FP16 execution, where exponent-field corruptions produce IEEE 754 special values (NaN, $\pm\infty$); Posit16 produces only SDCs across all fault classes, consistent with its absence of exception encodings.

**Hardware-aware analytical error-impact model.** The third contribution is a hardware-aware analytical model that abstracts each permanent fault as a combination of a spatial corruption mask and a statistical error generator, enabling large-scale application-level reliability evaluation without repeating full architectural fault injection. Spatial masks are extracted from `PyOpenTCU` structural campaigns; numerical error distributions are learned per fault class and bit position, distinguishing *S&E*/*D&C* stage faults (large-magnitude, exponent/regime-field deviations) from *SP*/*FO* stage faults (low-magnitude, fraction-field deviations). Validation on ResNet-18 RB1 (GEMM shape $64 \times 147 \times 12{,}544$) demonstrates that the model reproduces SDC/DUE rates, absolute error distributions, and spatial corruption patterns with up to 93% cross-correlation against full architectural fault injection, while achieving orders-of-magnitude speedups that make large-scale CNN-level reliability studies feasible.

**Format-aware selective hardware hardening.** The fourth contribution is a selective hardening methodology for FP32 and Posit32 arithmetic units that leverages the structural vulnerability profiles from the fine-grain analysis to concentrate protection resources on the highest-criticality pipeline stages within a $\leq$15% gate-overhead budget. Three mechanisms are evaluated — Self-Check and Repair (S-CR), DMR, and TMR — each offering different overhead-effectiveness tradeoffs. For FP32 cores, hardening the *S&E* and *RnD* stages reduces SDC rates by 40–54%, increases the fraction of SDCs with $|\Delta| < 1.0$ from 58.5%–87.5% to above 95%–97%, and eliminates 91–97% of catastrophic outliers, at gate-overhead of 18–25%

for ADD/MUL cores with S-CR. For Posit32 cores, reinforcing the *D&C*, *R*, and *EN* stages achieves stronger SDC reduction (68–75%) and 85–92% suppression of large-magnitude errors, at higher overhead (24–57%) reflecting the larger area share of regime/encoding logic. The S-CR mechanism reduces power overhead by 0.34×–4× compared to TMR through cold-spare activation. These results demonstrate that detailed structural knowledge enables significantly better reliability-per-cost trade-offs than traditional full-core TMR, which incurs >200% overhead for comparable coverage.

**Hardware-aware software fault tolerance for Tensor Core GEMM.** The fifth contribution is a two-level software fault-tolerant mechanism for Tensor Core GEMM that exploits the inherent hardware parallelism of TCUs — specifically the two independent TCUs per SM and the programmable HMMA tiling structure — to achieve in-field detection and mitigation at warp granularity with bounded overhead. The mechanism combines always-on coarse-grain redundancy (splitting each warp tile into two halves, executing each half in spatial duplication across the two TCUs and comparing the results) with conditional fine-grain Quadruple Modular Redundancy (activated only upon detection, subdividing each half into quarters to localize and isolate the faulty DPU partition). Both permanent and transient faults are addressed. The mechanism achieves 100% detection and correction for 1,024 injected permanent stuck-at faults and 1,442 injected transient single-bit flips, with bounded overhead: 40% additional registers per thread, 8% additional shared memory, at most 13% longer execution time on RTX 3060 Ti (11% on Jetson AGX Orin 64 GB), and at most 15% higher energy. Overhead is size-independent, converging to approximately 9% (RTX) and 4% (Orin) for large-scale GEMMs, demonstrating suitability for safety-critical embedded and HPC deployment.

# Future Directions

The results and findings of this thesis open several directions for future investigation.

**Reliability-aware design-space exploration.** An important direction is the integration of reliability and resilience as first-class objectives in the *early stages* of AI accelerator design. Current design-space exploration flows primarily optimize performance, throughput, energy efficiency, and area, while reliability is typically assessed after architectural choices are largely fixed. The results of this thesis indicate that architectural parameters — array dimensions, pipeline depth, numerical precision, buffer sizing, and scheduling policies — have a profound and non-trivial impact on fault activation rates, error propagation behavior, and end-to-end robustness. For instance, the larger area fraction of the *D&C*/*EN* stages in Posit cores (23–55% vs. 8–23% for FP *S&E*/*RnD*) directly determines the overhead ceiling for selective hardening. Incorporating hardware-aware reliability models — such as the

spatial error abstractions developed in this thesis — into early design exploration would enable joint optimization of performance, efficiency, and resilience from the outset, rather than treating reliability as a post-design retrofit.

**Broader fault models and technology-aware analysis.** This thesis focused on permanent stuck-at faults as the primary failure model. Extending the characterization to delay faults, bridging faults, and intermittent faults would provide a more complete picture of the reliability challenges in deeply scaled technologies. Similarly, the 15 nm FreePDK library used throughout is a research-grade model; future work should validate the vulnerability profiles and hardening effectiveness on commercial technology nodes (e.g., 7 nm, 5 nm), where electromigration, process variation, and radiation cross-sections differ significantly from 15 nm predictions. Radiation-induced transient faults affecting combinational logic inside arithmetic cores — as opposed to register-level flips captured by NVBitFI — also deserve dedicated characterization.

**Posit hardware architecture improvements.** The analysis consistently identified the *D&C* and *EN* stages as the dominant vulnerability sources in Posit32 cores, primarily due to the structural complexity of variable-length regime decoding and re-encoding. The quire Posit MAC core (*PQ_MAC*) could not be efficiently hardened within practical overhead bounds (>153% TMR overhead), indicating that improvements in Posit hardware architecture — such as fixed-regime approximations, HUB Posit encoding schemes, or redesigned *D&C*/*EN* datapath topologies — would directly reduce the cost of effective mitigation. Such architectural innovations could close the gap between Posit's theoretical numerical advantages and its current hardware reliability overhead.

**Extension to other accelerator architectures and formats.** The methodologies developed here were validated primarily on NVIDIA-like GPU Tensor Core architectures and on two numerical formats (FP and Posit). Extending the framework to other accelerator classes — including systolic arrays (TPU-style), dataflow architectures, and emerging in-memory computing fabrics — would generalize the cross-layer reliability paradigm. Extensions to FP8 and FP4 formats, increasingly adopted in quantized AI inference, would also be valuable, since compact formats have narrower exponent fields that may shift the relative vulnerability of *S&E* vs. *SP* stages and change the effectiveness of selective hardening strategies.

Overall, this thesis demonstrated that achieving effective reliability in AI-oriented accelerators requires a cross-layer perspective that connects arithmetic circuit behavior, accelerator microarchitecture, scalable evaluation models, and multi-level mitigation strategies. By combining detailed structural characterization, fast hardware-aware error modeling, format-specific selective hardening, and warp-level software fault tolerance, the methodologies developed here provide both analytical understanding and practical tools for designing dependable AI accelerators capable of operating safely and efficiently across the demanding environments in which they are increasingly deployed.

# Publications by the Candidate

The following publications were produced during the PhD period. They are organized first by publication type — journal articles, followed by conference and workshop contributions — and within each group, in reverse chronological order.

## Journal Articles

1. Rodriguez Condia, Josie Esteban; Guerrero-Balaguera, Juan-David; **Limas Sierra, Robert**; Sonza Reorda, Matteo. *Investigating and Mitigating Critical Faults in Floating-Point and Posit Arithmetic Hardware. IEEE Transactions on Emerging Topics in Computing*, vol. 13, no. 4, IEEE, 2025, pp. 1605–1617. DOI: 10.1109/TETC.2025.3615827. Available at: https://ieeexplore.ieee.org/document/11194099/.

2. Guerrero-Balaguera, Juan-David; **Limas Sierra, Robert**; Sensoz, Oguz; Rodriguez Condia, Josie E.; Escobar, Maynor Giovanni Ballina; Crespo, Maria Liz; Carrato, Sergio; Sonza Reorda, Matteo. *SHADOWFI: An Open-source Framework for Fault Evaluation of Complex IC Designs Using Hyperscale Computing. IEEE Access*, IEEE, 2025. DOI: 10.1109/ACCESS.2025.3641762. Available at: https://ieeexplore.ieee.org/document/11284814.

3. **Limas Sierra, Robert**; Guerrero-Balaguera, Juan-David; Rodriguez Condia, Josie E.; Sonza Reorda, Matteo. *A Structured Method to Generate Self-Test Libraries for Tensor Cores. Electronics*, vol. 14, no. 11, MDPI, 2025. DOI: 10.3390/electronics14112148. Available at: https://www.mdpi.com/2079-9292/14/11/2148.

4. Rodriguez Condia, Josie E.; Guerrero-Balaguera, Juan-David; Patiño Núñez, Edwar J.; **Limas Sierra, Robert**; Sonza Reorda, Matteo. *Investigating and Reducing the Architectural Impact of Transient Faults in Special Function Units for GPUs. Journal of Electronic Testing*, vol. 40, no. 2, Springer, 2024, pp. 215–228. DOI: 10.1007/s10836-024-06107-9. Available at: https://doi.org/10.1007/s10836-024-06107-9.

5. **Limas Sierra, Robert**; Guerrero-Balaguera, Juan-David; Rodriguez Condia, Josie E.; Sonza Reorda, Matteo. *Exploring Hardware Fault Impacts on Different Real Number Representations of the Structural Resilience of TCUs in GPUs*. *Electronics*, vol. 13, no. 3, MDPI, 2024. DOI: 10.3390/electronics13030578. Available at: https://www.mdpi.com/2079-9292/13/3/578.

## Book Chapters

1. **Limas Sierra, Robert**; Guerrero-Balaguera, Juan-David; Rodriguez Condia, Josie E.; Sonza Reorda, Matteo. *Analyzing the Reliability of TCUs Through Micro-architecture and Structural Evaluations for Two Real Number Formats*. In: *VLSI-SoC 2023: Silicon Innovations for Trustworthy Artificial Intelligence*, Springer, vol. 680, 2024, pp. 149–176. DOI: 10.1007/978-3-031-70947-0_8. Available at: https://link.springer.com/chapter/10.1007/978-3-031-70947-0_8.

## Conference and Workshop Papers

1. Guerrero-Balaguera, Juan-David; Rodriguez Condia, Josie Esteban; **Limas Sierra, Robert**; Sonza Reorda, Matteo. *About the Functional In-Field Self-Testing of AI Accelerators*. In: *2025 IEEE 26th Latin American Test Symposium (LATS)*, IEEE, 2025, pp. 1–6. DOI: 10.1109/LATS65346.2025.10963960. Available at: https://ieeexplore.ieee.org/document/10963960/.

2. **Limas Sierra, Robert**; Esposito, Giuseppe; Guerrero-Balaguera, Juan-David; Rodriguez Condia, Josie E.; Sonza Reorda, Matteo. *Improving CNN Runtime Robustness Against Soft Errors by Dropout Layer Optimization*. In: *2025 IEEE 26th Latin American Test Symposium (LATS)*, IEEE, 2025, pp. 1–6. DOI: 10.1109/LATS65346.2025.10963963. Available at: https://ieeexplore.ieee.org/document/10963963.

3. Caro-Anzola, Edward-W.; Esposito, Giuseppe; Guerrero-Balaguera, Juan-David; Jiménez-López, Andrés-F.; Jiménez-López, Fabián-R.; **Limas Sierra, Robert**; Ramirez-Espinosa, Gustavo; Giusto, Edoardo; Montrucchio, Bartolomeo; Vera-Cely, Oscar-F.; Rodriguez Condia, Josie Esteban. *IoT and Edge Computing: Applications and Reliability Implications*. In: *2025 IEEE 26th Latin American Test Symposium (LATS)*, IEEE, 2025, pp. 1–10. DOI: 10.1109/LATS65346.2025.10963944. Available at: https://ieeexplore.ieee.org/document/10963944/.

4. Pessia, Francesco; Guerrero-Balaguera, Juan-David; **Limas Sierra, Robert**;

Rodriguez Condia, Josie E.; Levorato, Marco; Sonza Reorda, Matteo. *Effective Application-level Error Modeling of Permanent Faults on AI Accelerators*. In: *2024 IEEE 30th International Symposium on On-Line Testing and Robust System Design (IOLTS)*, IEEE, 2024, pp. 1–7. DOI: 10.1109/IOLTS60994.2024.10616087. Available at: https://ieeexplore.ieee.org/document/10616087.

5. Chen, Junchao; Esposito, Giuseppe; Fernandes dos Santos, Fernando; Guerrero-Balaguera, Juan-David; Kritikakou, Angeliki; Krstic, Milos; **Limas Sierra, Robert**; Rodriguez-Condia, Josie-Esteban; Sonza Reorda, Matteo; Traiola, Marcello; Veronesi, Alessandro. *Reliability Assessment of Large DNN Models: Trading Off Performance and Accuracy*. In: *2024 IFIP/IEEE 32nd International Conference on Very Large Scale Integration (VLSI-SoC)*, IEEE, 2024, pp. 1–10. DOI: 10.1109/VLSI-SoC62099.2024.10767814. Available at: https://ieeexplore.ieee.org/abstract/document/10767814.

6. Rodriguez Condia, Josie E.; Guerrero-Balaguera, Juan-David; **Limas Sierra, Robert**; Sonza Reorda, Matteo. *Analyzing the Structural and Operational Impact of Faults in Floating-Point and Posit Arithmetic Cores for CNN Operations*. In: *2024 IEEE 29th European Test Symposium (ETS)*, IEEE, 2024, pp. 1–4. DOI: 10.1109/ETS61313.2024.10567884. Available at: https://ieeexplore.ieee.org/document/10567884.

7. **Limas Sierra, Robert**; Guerrero-Balaguera, Juan-David; Pessia, Francesco; Rodriguez Condia, Josie E.; Sonza Reorda, Matteo. *Analyzing the Impact of Scheduling Policies on the Reliability of GPUs Running CNN Operations*. In: *2024 IEEE 42nd VLSI Test Symposium (VTS)*, IEEE, 2024. DOI: 10.1109/VTS60656.2024.10538940. Available at: https://ieeexplore.ieee.org/document/10538940.

8. Ahmadilivani, Mohammad Hasan; Bosio, Alberto; Deveautour, Bastien; Fernandes Dos Santos, Fernando; Guerrero-Balaguera, Juan-David; Jenihhin, Maksim; Kritikakou, Angeliki; **Limas Sierra, Robert**; Pappalardo, Salvatore; Raik, Jaan; Rodriguez Condia, Josie E.; Sonza Reorda, Matteo; Taheri, Mahdi; Traiola, Marcello. *Special Session: Reliability Assessment Recipes for DNN Accelerators*. In: *2024 IEEE 42nd VLSI Test Symposium (VTS)*, IEEE, 2024. DOI: 10.1109/VTS60656.2024.10538707. Available at: https://ieeexplore.ieee.org/document/10538707.

9. **Limas Sierra, Robert**; Guerrero-Balaguera, Juan-David; Rodriguez Condia, Josie Esteban; Sonza Reorda, Matteo. *A Reliability-aware Environment for Design Exploration for GPU Devices*. In: *2023 26th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS)*,

IEEE, 2023, pp. 169–174. DOI: 10.1109/DDECS57882.2023.10139643. Available at: https://ieeexplore.ieee.org/document/10139643.

10. Rodriguez Condia, Josie E.; Guerrero-Balaguera, Juan-David; Patiño Núñez, Edwar J.; **Limas Sierra, Robert**; Sonza Reorda, Matteo. *Evaluating the Prevalence of SFUs in the Reliability of GPUs*. In: *2023 IEEE 28th European Test Symposium (ETS)*, IEEE, 2023, pp. 1–6. DOI: 10.1109/ETS56758.2023.10174110. Available at: https://ieeexplore.ieee.org/document/10174110.

11. **Limas Sierra, Robert**; Guerrero-Balaguera, Juan-David; Rodriguez Condia, Josie E.; Sonza Reorda, Matteo. *Analyzing the Impact of Different Real Number Formats on the Structural Reliability of TCUs in GPUs*. In: *2023 IFIP/IEEE 31st International Conference on Very Large Scale Integration (VLSI-SoC)*, IEEE, 2023, pp. 1–6. DOI: 10.1109/VLSI-SoC57769.2023.10321881. Available at: https://ieeexplore.ieee.org/document/10321881.

12. Rodriguez Condia, Josie E.; Guerrero-Balaguera, Juan-David; Patiño Núñez, Edwar J.; **Limas Sierra, Robert**; Sonza Reorda, Matteo. *Analyzing the Architectural Impact of Transient Fault Effects in SFUs of GPUs*. In: *2023 IEEE 24th Latin American Test Symposium (LATS)*, IEEE, 2023, pp. 1–6. DOI: 10.1109/LATS58125.2023.10154504. Available at: https://ieeexplore.ieee.org/document/10154504.

13. Guerrero-Balaguera, Juan-David; Galasso, Luigi; **Limas Sierra, Robert**; Sanchez, Ernesto; Sonza Reorda, Matteo. *Evaluating the Impact of Permanent Faults in a GPU Running a Deep Neural Network*. In: *2022 IEEE International Test Conference in Asia (ITC-Asia)*, IEEE, 2022, pp. 96–101. DOI: 10.1109/ITCASIA55616.2022.00027. Available at: https://ieeexplore.ieee.org/document/9946344.

14. Guerrero-Balaguera, Juan-David; **Limas Sierra, Robert**; Sonza Reorda, Matteo. *Effective Fault Simulation of GPU's Permanent Faults for Reliability Estimation of CNNs*. In: *2022 IEEE 28th International Symposium on On-Line Testing and Robust System Design (IOLTS)*, IEEE, 2022, pp. 1–6. DOI: 10.1109/IOLTS56730.2022.9897823. Available at: https://ieeexplore.ieee.org/document/9897823.

# Bibliography


[1] B. Fleming. "Microcontroller Units in Automobiles [Automotive Electronics]". In: *IEEE Vehicular Technology Magazine* 6.3 (2011), pp. 4–8. DOI: 10.1109/MVT.2011.941888.

[2] J. P. Trovao. "Trends in Automotive Electronics [Automotive Electronics]". In: *IEEE Vehicular Technology Magazine* 14.4 (2019), pp. 100–109. DOI: 10.1109/MVT.2019.2939757.

[3] J. Janai et al. "Computer Vision for Autonomous Vehicles: Problems, Datasets and State of the Art". In: *Foundations and Trends® in Computer Graphics and Vision* 12.1–3 (2020), pp. 1–308. ISSN: 1572-2740. DOI: 10.1561/0600000079.

[4] J. C. Knight. "Safety critical systems: challenges and directions". In: *Proceedings of the 24th International Conference on Software Engineering. ICSE 2002*. 2002, pp. 547–550. ISBN: 1-58113-472-X.

[5] *ISO 26262: Road Vehicles – Functional Safety*. 2nd edition. International Organization for Standardization, 2018.

[6] Colin Paterson et al. "Safety assurance of Machine Learning for autonomous systems". In: *Reliability Engineering  System Safety* 264 (2025), p. 111311. ISSN: 0951-8320. DOI: https://doi.org/10.1016/j.ress.2025.111311. URL: https://www.sciencedirect.com/science/article/pii/S0951832025005125.

[7] Le-Ha Hoang, Muhammad Abdullah Hanif, and Muhammad Shafique. "FT-ClipAct: Resilience Analysis of Deep Neural Networks and Improving their Fault Tolerance using Clipped Activation". In: *2020 Design, Automation Test in Europe Conference  Exhibition (DATE)*. 2020, pp. 1241–1246. DOI: 10.23919/DATE48585.2020.9116571.

[8] Guanpeng Li et al. "Understanding Error Propagation in Deep Learning Neural Network (DNN) Accelerators and Applications". In: *SC17: International Conference for High Performance Computing, Networking, Storage and Analysis*. 2017, pp. 1–12.

[9] Biagio Peccerillo et al. “A survey on hardware accelerators: Taxonomy, trends, challenges, and perspectives”. In: *Journal of Systems Architecture* 129 (2022), p. 102561.

[10] Norman P. Jouppi et al. “In-Datacenter Performance Analysis of a Tensor Processing Unit”. In: *Proceedings of the 44th Annual International Symposium on Computer Architecture (ISCA)*. ACM, 2017, pp. 1–12. DOI: 10.1145/3079856.3080246.

[11] Md Aamir Raihan, Negar Goli, and Tor Aamodt. “Modeling Deep Learning Accelerator Enabled GPUs”. In: *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 2019, pp. 79–92.

[12] NVIDIA Corporation. “NVIDIA VOLTA V100 ARCHITECTURE”. In: *White paper* (2018).

[13] P. M. Basso et al. “Impact of Tensor Cores and Mixed Precision on the Reliability of Matrix Multiplication in GPUs”. In: *IEEE Transactions on Nuclear Science* 67.7 (2020), pp. 1560–1565. DOI: 10.1109/TNS.2020.2977583.

[14] S. Khan et al. “BTI impact on logical gates in nano-scale CMOS technology”. In: *2012 IEEE 15th International Symposium on Design and Diagnostics of Electronic Circuits Systems (DDECS)*. 2012, pp. 348–353. DOI: 10.1109/DDECS.2012.6219086.

[15] S. Hamdioui et al. “Reliability challenges of real-time systems in forthcoming technology nodes”. In: *2013 Design, Automation Test in Europe Conference Exhibition (DATE)*. 2013, pp. 129–134. DOI: 10.7873/DATE.2013.040.

[16] I. Agbo et al. “Read path degradation analysis in SRAM”. In: *2016 21th IEEE European Test Symposium (ETS)*. 2016, pp. 1–2. DOI: 10.1109/ETS.2016.7519325.

[17] S. Pae et al. “Effect of BTI Degradation on Transistor Variability in Advanced Semiconductor Technologies”. In: *IEEE Transactions on Device and Materials Reliability* 8.3 (2008), pp. 519–525. DOI: 10.1109/TDMR.2008.2002351.

[18] K. Iniewski. *Radiation effects in semiconductors*. CRC press, 2018, pp. 1–431. ISBN: 978-1-4398-2694-2. DOI: 10.1201/9781315217864.

[19] D. Gil-Tomás et al. “Studying the effects of intermittent faults on a microcontroller”. In: *Microelectronics Reliability* 52.11 (2012), pp. 2837–2846. DOI: 10.1016/j.microrel.2012.06.004.

[20] M. Radetzki et al. “Methods for fault tolerance in networks-on-chip”. In: *ACM Computing Surveys (CSUR)* 46.1 (2013), pp. 1–38. DOI: 10.1145/2522968.2522976.

[21] S. Borkar et al. “Microprocessors in the Era of Terascale Integration”. In: *2007 Design, Automation Test in Europe Conference Exhibition*. 2007, pp. 1–6. DOI: 10.1109/DATE.2007.364597.

[22] H. G. Kerkhoff et al. “Intermittent Resistive Faults in Digital CMOS Circuits”. In: *2015 IEEE 18th International Symposium on Design and Diagnostics of Electronic Circuits Systems*. 2015, pp. 211–216. DOI: 10.1109/DDECS.2015.12.

[23] Fernanda Lima Kastensmidt et al. “Designing and testing fault-tolerant techniques for SRAM-based FPGAs”. In: *Proceedings of the 1st Conference on Computing Frontiers*. CF ’04. Ischia, Italy: Association for Computing Machinery, 2004, pp. 419–432. ISBN: 1581137419. DOI: 10.1145/977091.977150. URL: https://doi.org/10.1145/977091.977150.

[24] Sparsh Mittal. “A Survey of Soft-Error Mitigation Techniques for Non-Volatile Memories”. In: *Computers* 6.1 (2017). ISSN: 2073-431X. DOI: 10.3390/computers6010008. URL: https://www.mdpi.com/2073-431X/6/1/8.

[25] M. Badawi and K. Hamouda, eds. *Reliability of Nanoscale Circuits and Systems*. Springer, 2006. DOI: 10.5555/1942206. URL: https://dl.acm.org/doi/abs/10.5555/1942206.

[26] Mark White et al. *Scaled CMOS technology reliability users guide*. Tech. rep. Pasadena, CA: Jet Propulsion Laboratory, National Aeronautics and Space, 2010.

[27] Ahmed Ben and Slim Ben. “A Survey of Network-On-Chip Tools”. In: *International Journal of Advanced Computer Science and Applications* 4.9 (2013). ISSN: 2156-5570. DOI: 10.14569/ijacsa.2013.040910. URL: http://dx.doi.org/10.14569/IJACSA.2013.040910.

[28] M. Psarakis et al. “Microprocessor Software-Based Self-Testing”. In: *IEEE Design Test of Computers* 27.3 (2010), pp. 4–19. DOI: 10.1109/MDT.2010.5.

[29] Harish Dixit et al. “Silent Data Corruptions at Scale”. In: *arXiv preprint arXiv:2102.11245* (2021).

[30] Peter H. Hochschild et al. “Cores that Don’t Count”. In: *Proceedings of the Workshop on Hot Topics in Operating Systems (HotOS)*. 2021, pp. 9–16.

[31] Cristiana Bolchini, Luca Cassano, and Antonio Miele. “Resilience of deep learning applications: A systematic literature review of analysis and hardening techniques”. In: *Computer Science Review* 54 (2024), p. 100682.

[32] H. Ardebili et al. “10 - Trends and challenges”. In: *Encapsulation Technologies for Electronic Applications (Second Edition)*. Second Edition. Materials and Processes for Electronic Applications. William Andrew Publishing, 2019, pp. 431–479. ISBN: 978-0-12-811978-5. DOI: 10.1016/B978-0-12-811978-5.00010-9.

[33] Robert Limas Sierra et al. “Analyzing the Impact of Different Real Number Formats on the Structural Reliability of TCUs in GPUs”. In: *2023 IFIP/IEEE 31st International Conference on Very Large Scale Integration (VLSI-SoC)*. 2023, pp. 1–6. DOI: 10.1109/VLSI-SoC57769.2023.10321881.

[34] Sotiris Tselonis, Vasileios Dimitsas, and Dimitris Gizopoulos. “The Functional and Performance Tolerance of GPUs to Permanent Faults in Registers”. In: *IEEE International On-Line Testing Symposium (IOLTS)*. 2013, pp. 1–6.

[35] Josie E. Rodriguez Condia et al. “Analyzing the Structural and Operational Impact of Faults in Floating-Point and Posit Arithmetic Cores for CNN Operations”. In: *2024 IEEE European Test Symposium (ETS)*. 2024, pp. 1–4. DOI: 10.1109/ETS61313.2024.10567884.

[36] S. K. S. Hari et al. “SASSIFI: An architecture-level fault injection tool for GPU application resilience evaluation”. In: *2017 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 2017, pp. 249–258. DOI: 10.1109/ISPASS.2017.7975296.

[37] Josie E. Rodriguez Condia et al. “Investigating and Mitigating Critical Faults in Floating-Point and Posit Arithmetic Hardware”. In: *IEEE Transactions on Emerging Topics in Computing* 13.4 (2025), pp. 1605–1617. DOI: 10.1109/TETC.2025.3615827.

[38] Timothy Tsai et al. “NVBitFI: Dynamic Fault Injection for GPUs”. In: *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. 2021, pp. 284–291. DOI: 10.1109/DSN48987.2021.00041.

[39] A. Mahmoud et al. “Optimizing Software-Directed Instruction Replication for GPU Error Detection”. In: *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*. 2018, pp. 842–854. DOI: 10.1109/SC.2018.00070.

[40] Cristian Constantinescu, Mike Butler, and Chris Weller. “Error injection-based study of soft error propagation in AMD Bulldozer microprocessor module”. In: *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2012)*. 2012, pp. 1–6. DOI: 10.1109/DSN.2012.6263922.

[41] Dimitris Sartzetakis et al. “Probing Weaknesses in GPU Reliability Assessment: A Cross-Layer Approach”. In: *IEEE European Test Symposium (ETS)*. 2024, pp. 1–6. DOI: 10.1109/ETS61313.2024.10590041.

[42] Cristiana Bolchini et al. “Fast and Accurate Error Simulation for CNNs Against Soft Errors”. In: *IEEE Trans. Comput.* (2023).

[43] Francesco Pessia et al. “Effective Application-level Error Modeling of Permanent Faults on AI Accelerators”. In: *2024 IEEE 30th International Symposium on On-Line Testing and Robust System Design (IOLTS)*. 2024, pp. 1–7. DOI: 10.1109/IOLTS60994.2024.10616087.

[44] Nirmal Saxena and Atieh Lotfi. “Error Model (EM)—A New Way of Doing Fault Simulation”. In: *2022 IEEE International Test Conference (ITC)*. 2022, pp. 324–333. DOI: 10.1109/ITC50671.2022.00040.

[45] Shubu Mukherjee. *Architecture Design for Soft Errors*. Morgan Kaufmann, 2008. ISBN: 978-0-12-369529-1. DOI: 10.1016/B978-0-12-369529-1.X5001-0. URL: https://www.sciencedirect.com/book/9780123695291/architecture-design-for-soft-errors.

[46] R. Limas Sierra et al. “A Software Fault-Tolerant Mechanism for Matrix Multiplication on Tensor Cores”. In: *IEEE Transactions on Computers* (2026). Submitted for publication.

[47] V.S.S. Nair, J.A. Abraham, and P. Banerjee. “Efficient techniques for the analysis of algorithm-based fault tolerance (ABFT) schemes”. In: *IEEE Transactions on Computers* 45.4 (1996), pp. 499–503. DOI: 10.1109/12.494110.

[48] Robert Limas Sierra et al. “Analyzing the Reliability of TCUs Through Micro-architecture and Structural Evaluations for Two Real Number Formats”. In: *VLSI-SoC 2023: Innovations for Trustworthy Artificial Intelligence*. Ed. by Ibrahim (Abe) M. Elfadel and Lutfi Albasha. Cham: Springer Nature Switzerland, 2024, pp. 149–176. ISBN: 978-3-031-70947-0.

[49] Robert Limas Sierra et al. “Optimizing the Analysis and Evaluation of Logic Simulation Workloads in HPC Systems”. In: *2023 IEEE 17th International Conference on Application of Information and Communication Technologies (AICT)*. 2023, pp. 1–6. DOI: 10.1109/AICT59525.2023.10313156.

[50] Annachiara Ruospo et al. “A Survey on Deep Learning Resilience Assessment Methodologies”. In: *Computer* 56.2 (2023), pp. 57–66.

[51] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. *Deep Learning*. Nature, vol. 521, pp. 436–444. 2015. DOI: 10.1038/nature14539.

[52] Vivienne Sze et al. “Efficient Processing of Deep Neural Networks: A Tutorial and Survey”. In: *Proceedings of the IEEE* 105.11 (2017), pp. 2295–2329. DOI: 10.1109/JPROC.2017.2761740.

[53] C. M. Wittenbrink, E. Kilgariff, and A. Prabhu. "Fermi GF100 GPU Architecture". In: *IEEE Micro* 31.2 (2011), pp. 50–59. DOI: 10.1109/MM.2011.24.

[54] J. Nickolls et al. "The GPU Computing Era". In: vol. 30. 2. 2010, pp. 56–69. DOI: 10.1109/MM.2010.41.

[55] A. Bakhoda et al. "Analyzing CUDA workloads using a detailed GPU simulator". In: *2009 IEEE International Symposium on Performance Analysis of Systems and Software*. 2009, pp. 163–174. DOI: 10.1109/ISPASS.2009.4919648.

[56] S. Markidis et al. "NVIDIA Tensor Core Programmability, Performance Precision". In: *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. 2018, pp. 522–531. DOI: 10.1109/IPDPSW.2018.00091.

[57] Brent Ralph Boswell et al. *Generalized acceleration of matrix multiply accumulate operations*. US Patent and Trademark Office, US Patent 10,338,919. July 2019.

[58] Israel Koren. *Computer Arithmetic Algorithms*. 2nd. A.K. Peters/CRC Press, 2002. ISBN: 978-1-5688-1160-4.

[59] David Goldberg. "What every computer scientist should know about floating-point arithmetic". In: *ACM Computing Surveys* 23.1 (1991), pp. 5–48.

[60] *IEEE Standard for Floating-Point Arithmetic*. IEEE, 2019. DOI: 10.1109/IEEESTD.2019.8766229.

[61] John L. Gustafson and Isaac T. Yonemoto. "Beating Floating Point at its Own Game: Posit Arithmetic". In: *Supercomputing Frontiers and Innovations* 4.2 (2017), pp. 71–86.

[62] Jonas Gava et al. "SOFIA: An automated framework for early soft error assessment, identification, and mitigation". In: *Journal of Systems Architecture* 131 (2022), p. 102710. ISSN: 1383-7621. DOI: https://doi.org/10.1016/j.sysarc.2022.102710. URL: https://www.sciencedirect.com/science/article/pii/S1383762122002028.

[63] H. Ziade et al. "A survey on fault injection techniques". In: *International Arab Journal of Information Technology* Vol. 1, No. 2, July (2004), pp. 171–186.

[64] M. Eslami et al. "A survey on fault injection methods of digital integrated circuits". In: *Integration* 71 (2020), pp. 154–163. ISSN: 0167-9260. DOI: https://doi.org/10.1016/j.vlsi.2019.11.006.

[65] P. Rech et al. "Neutron radiation test of graphic processing units". In: *2012 IEEE 18th International On-Line Testing Symposium (IOLTS)*. 2012, pp. 55–60. DOI: 10.1109/IOLTS.2012.6313841.

[66] Adrian Sampson et al. “EnerJ: approximate data types for safe and general low-power computation”. In: *SIGPLAN Not.* 46.6 (June 2011), pp. 164–174. ISSN: 0362-1340. DOI: 10.1145/1993316.1993518. URL: https://doi.org/10.1145/1993316.1993518.

[67] Oreste Villa et al. “NVBit: A Dynamic Binary Instrumentation Framework for NVIDIA GPUs”. In: *52nd IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 2019, pp. 372–383.

[68] Robert Limas Sierra et al. “Improving CNN Runtime Robustness Against Soft Errors by Dropout Layer Optimization”. In: *2025 IEEE 26th Latin American Test Symposium (LATS)*. 2025, pp. 1–6. DOI: 10.1109/LATS65346.2025.10963963.

[69] Juan David Guerrero Balaguera et al. “Understanding the Effects of Permanent Faults in GPU’s Parallelism Management and Control Units”. In: *Int. Conf. High Perform. Comput. Netw. Storage Anal. (SC’23)*. 2023, pp. 1–10. ISBN: 9798400701092. DOI: 10.1145/3581784.3607086.

[70] Abdulrahman Mahmoud et al. “PyTorchFI: A Runtime Perturbation Tool for DNNs”. In: *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*. 2020, pp. 25–31. DOI: 10.1109/DSN-W50199.2020.00014.

[71] S. Tselonis et al. “GUFI: A framework for GPUs reliability assessment”. In: *2016 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 2016, pp. 90–100. DOI: 10.1109/ISPASS.2016.7482077.

[72] A. Vallero et al. “Multi-faceted microarchitecture level reliability characterization for NVIDIA and AMD GPUs”. In: *2018 IEEE 36th VLSI Test Symposium (VTS)*. 2018, pp. 1–6. DOI: 10.1109/VTS.2018.8368665.

[73] J. E. R. Condia et al. “Combining Architectural Simulation and Software Fault Injection for a Fast andAccurate CNNs Reliability Evaluation on GPUs”. In: *2021 IEEE 39th VLSI Test Symposium (VTS), Virtual Event*. 2021, "in press", pp. 1–6.

[74] F. F. d. Santos et al. “Analyzing and Increasing the Reliability of Convolutional Neural Networks on GPUs”. In: *IEEE Transactions on Reliability* 68.2 (2019), pp. 663–677. DOI: 10.1109/TR.2018.2878387.

[75] Cristiana Bolchini et al. “Analyzing the Reliability of Alternative Convolution Implementations for Deep Learning Applications”. In: *IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*. 2023, pp. 1–6.

[76] NumPy Developers. *NumPy: The fundamental package for scientific computing with Python*. https://numpy.org/. Accessed: 2026-01-06. 2024.

[77] Cerlane Leong. *SoftPosit: A C Library Emulating the Posit Standard*. 2020. DOI: 10.5281/zenodo.3709035. URL: https://gitlab.com/cerlane/SoftPosit.

[78] M. Martins et al. "Open Cell Library in 15nm FreePDK Technology". In: *Proceedings of the 2015 Symposium on International Symposium on Physical Design*. ISPD '15. 2015, pp. 171–178. ISBN: 9781450333993. DOI: 10.1145/2717764.2717783.

[79] Florent de Dinechin and Bogdan Pasca. *FloPoCo: Floating-Point to Custom Operators for FPGA*. https://flopoco.org. 2011.

[80] H. Wunderlich et al. "Efficacy and efficiency of algorithm-based fault-tolerance on GPUs". In: *2013 IEEE 19th International On-Line Testing Symposium (IOLTS)*. 2013, pp. 240–243. DOI: 10.1109/IOLTS.2013.6604090.

[81] L. L. Pilla et al. "Software-Based Hardening Strategies for Neutron Sensitive FFT Algorithms on GPUs". In: *IEEE Transactions on Nuclear Science* 61.4 (2014), pp. 1874–1880. DOI: 10.1109/TNS.2014.2301768.

[82] D. J. Palframan et al. "Precision-aware soft error protection for GPUs". In: *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*. 2014, pp. 49–59. DOI: 10.1109/HPCA.2014.6835966.

[83] F. F. Dos Santos et al. "Reduced Precision DWC: an Efficient Hardening Strategy for Mixed-Precision Architectures". In: *IEEE Transactions on Computers* (2021), pp. 1–1. DOI: 10.1109/TC.2021.3058872.

[84] M. Gonçalves et al. "Selective Fault Tolerance for Register Files of Graphics Processing Units". In: *IEEE Transactions on Nuclear Science* 66.7 (2019), pp. 1449–1456. DOI: 10.1109/TNS.2019.2903027.

[85] I. Polian et al. "Selective Hardening: Toward Cost-Effective Error Tolerance". In: *IEEE Design and Test of Computers* 28.3 (2011), pp. 54–63. DOI: 10.1109/MDT.2010.120.

[86] Fabiano Libano, Paolo Rech, and John Brunhaver. "Efficient Error Detection for Matrix Multiplication With Systolic Arrays on FPGAs". In: *IEEE Transactions on Computers* 72.8 (2023), pp. 2390–2403. DOI: 10.1109/TC.2023.3248282.

[87] Shixun Wu et al. "Anatomy of High-Performance GEMM with Online Fault Tolerance on GPUs". In: *Proceedings of the 37th ACM International Conference on Supercomputing*. ICS '23. Orlando, FL, USA: Association for Computing Machinery, 2023, pp. 360–372. ISBN: 9798400700569. DOI: 10.1145/3577193.3593715. URL: https://doi.org/10.1145/3577193.3593715.

[88] Jorge Tonfat et al. “Soft error susceptibility analysis methodology of HLS designs in SRAM-based FPGAs”. In: *Microprocessors and Microsystems* 51 (2017), pp. 209–219. ISSN: 0141-9331. DOI: https://doi.org/10.1016/j.micpro.2017.04.016. URL: https://www.sciencedirect.com/science/article/pii/S014193311730217X.

[89] Sergi Alcaide et al. “Achieving Diverse Redundancy for GPU Kernels”. In: *IEEE Transactions on Emerging Topics in Computing* 10.2 (2022), pp. 618–634.

[90] Nikolaos Andriotis et al. “A Software-Only Approach to Enable Diverse Redundancy on Intel GPUs for Safety-Related Kernels”. In: *38th ACM/SIGAPP Symposium on Applied Computing (SAC’23)*. 2023, pp. 451–460.

[91] Kuang-Hua Huang and Jacob A. Abraham. “Algorithm-Based Fault Tolerance for Matrix Operations”. In: *IEEE Transactions on Computers* C-33.6 (1984), pp. 518–528.

[92] P. Rech et al. “An Efficient and Experimentally Tuned Software-Based Hardening Strategy for Matrix Multiplication on GPUs”. In: *IEEE Transactions on Nuclear Science* 60.4 (2013), pp. 2797–2804. DOI: 10.1109/TNS.2013.2252625.

[93] Jack Kosaian and K. V. Rashmi. “Arithmetic-intensity-guided fault tolerance for neural network inference on GPUs”. In: *International Conference for High Performance Computing, Networking, Storage and Analysis (SC’21)*. 2021.

[94] J. Chen, S. Li, and Z. Chen. “GPU-ABFT: Optimizing Algorithm-Based Fault Tolerance for Heterogeneous Systems with GPUs”. In: *2016 IEEE International Conference on Networking, Architecture and Storage (NAS)*. 2016, pp. 1–2. DOI: 10.1109/NAS.2016.7549404.

[95] Saurabh Hukerikar and Nirmal Saxena. “Runtime Fault Diagnostics for GPU Tensor Cores”. In: *IEEE International Test Conference (ITC)*. 2022, pp. 524–528. DOI: 10.1109/ITC50671.2022.00065.

[96] Annachiara Ruospo et al. “Image Test Libraries for the On-Line Self-Test of Functional Units in GPUs Running CNNs”. In: *IEEE European Test Symposium (ETS)*. 2023, pp. 1–6. DOI: 10.1109/ETS56758.2023.10174176.

[97] R. Limas Sierra et al. “A Structured Method to Generate Self-Test Libraries for Tensor Cores”. In: *Electronics* 14.11 (2025), p. 2148. DOI: 10.3390/electronics14112148.

[98] Mohammad Hassan Hafezan and Ehsan Atoofian. “Transient Fault Detection in Tensor Cores for Modern GPUs”. In: *ACM Transactions on Embedded Computing Systems* 23.5 (2024).

[99] Bo Fang et al. *MPGemmFI: A Fault Injection Technique for Mixed Precision GEMM in ML Applications*. 2023. arXiv: 2311.05782 [cs.DC]. URL: https://arxiv.org/abs/2311.05782.

[100] Siva Kumar Sastry Hari et al. "Making Convolutions Resilient Via Algorithm-Based Error Detection Techniques". In: *IEEE Transactions on Dependable and Secure Computing* 19.4 (2022), pp. 2546–2558. DOI: 10.1109/TDSC.2021.3063083.

[101] Cristiana Bolchini et al. "Fast and Accurate Error Simulation for CNNs Against Soft Errors". In: *IEEE Transactions on Computers* 72.4 (2023), pp. 984–997.

[102] R. Leveugle et al. "Statistical fault injection: Quantified error and confidence". In: *2009 Design, Automation Test in Europe Conference Exhibition*. 2009, pp. 502–506. DOI: 10.1109/DATE.2009.5090716.

This Ph.D. thesis has been typeset by means of the TEX-system facilities. The typesetting engine was pdfLATEX. The document class was `toptesi`, by Claudio Beccari, with option `tipotesi=scudo`. This class is available in every up-to-date and complete TEX-system installation.