POLITECNICO DI TORINO
Repository ISTITUZIONALE

Fast SEU Detection and Recovery in FPGA-Based AI Accelerators

Original
Fast SEU Detection and Recovery in FPGA-Based AI Accelerators / Cora, G., Vacca, E., De Sio, C., Azimi, S., Sterpone, L.. - In: ACM TRANSACTIONS ON RECONFIGURABLE TECHNOLOGY AND SYSTEMS. - ISSN 1936-7406. - 19:3(2026), pp. 1-34. [10.1145/3806052]

Availability:
This version is available at: 11583/3009709 since: 2026-04-09T09:58:34Z

Publisher:
ACM

Published
DOI:10.1145/3806052

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright

(Article begins on next page)

30 October 2026

Fast SEU Detection and Recovery in FPGA-Based AI Accelerators

GIORGIO CORA, ELEONORA VACCA, CORRADO DE SIO, SARAH AZIMI, and LUCA STERPONE, Department of Control and Computer Engineering, Politecnico di Torino, Turin, Italy

Heterogeneous FPGA platforms combining RISC-V processors and deep-learning accelerators are increasingly adopted in avionics and space, where performance must be paired with fault tolerance. We present RePAIR, a reconfigurable platform integrating an open source RISC-V core with a TPU-like systolic-array accelerator for runtime fault detection, correction, and recovery. RePAIR extends the accelerator ISA with runtime self-test, enabling detection of structural faults in the array during inference. The platform supports dual inference modes: a plain mode with no overhead and a testing mode that performs checksum validation at a fixed cost of three extra cycles per matrix multiplication, with limited accelerator area overhead. Upon fault detection, the accelerator notifies the RISC-V processor, which triggers dynamic partial reconfiguration of the faulty region while preserving execution state, allowing inference to resume from the last correct step. Compared with full-device reconfiguration, recovery time is reduced by up to 900× on AMD KCU105 and 1,400× on AMD ZCU102, while inference overhead remains ≤30% in the worst case. The methodology is hardware-agnostic and portable across FPGA devices, as shown by multi-platform implementations. Fault-injection campaigns combined with space-environment modeling estimate mean time to failure under mission conditions, demonstrating scalable and reliable FPGA-based AI acceleration for safety-critical applications.

CCS Concepts: • **Hardware → Error detection and error correction**;

Additional Key Words and Phrases: RISC-V, Systolic Arrays, Fault Detection, Partial Reconfiguration, AI

ACM Reference format:
Giorgio Cora, Eleonora Vacca, Corrado De Sio, Sarah Azimi, and Luca Sterpone. 2026. Fast SEU Detection and Recovery in FPGA-Based AI Accelerators. *ACM Trans. Reconfig. Technol. Syst.* 19, 3, Article 42 (September 2026), 34 pages.
https://doi.org/10.1145/3806052

1 Introduction

The widespread adoption of **Deep Learning (DL)** techniques, combined with the increase in model complexity, has driven the development of high-performance platforms capable of delivering the computational power required for efficient and rapid inference. Due to the open source nature of the RISC-V **Instruction Set Architecture (ISA)**, researchers are increasingly focusing on enhancing

Authors' Contact Information: Giorgio Cora (corresponding author), Department of Control and Computer Engineering, Politecnico di Torino, Turin, Italy; e-mail: giorgio.cora@polito.it; Eleonora Vacca, Department of Control and Computer Engineering, Politecnico di Torino, Turin, Italy; e-mail: eleonora.vacca@polito.it; Corrado De Sio, Department of Control and Computer Engineering, Politecnico di Torino, Turin, Italy; e-mail: corrado.desio@polito.it; Sarah Azimi, Department of Control and Computer Engineering, Politecnico di Torino, Turin, Italy; e-mail: sarah.azimi@polito.it; Luca Sterpone, Department of Control and Computer Engineering, Politecnico di Torino, Turin, Italy; e-mail: luca.sterpone@polito.it.



This work is licensed under Creative Commons Attribution International 4.0.

© 2026 Copyright held by the owner/author(s).
ACM 1936-7414/2026/9-ART42
https://doi.org/10.1145/3806052

RISC-V-based solutions to meet these computational demands. The predominant strategies involve either extending the ISA [1] or coupling RISC-V with application-specific accelerators [2]. Since most DL workloads involve large-scale matrix multiplications, many proposed solutions integrate methods for executing neural networks, such as **General Matrix Multiplications (GEMM)**, within the RISC-V environment [3, 4], which has renewed interest in **Systolic Array (SA)** architectures. Solutions based on SA employ a grid-like structure of **Processing Elements (PEs)** for efficient parallel computation, minimizing memory access, maximizing data reuse, and performing on-chip computation. These aspects are appealing for **Deep Neural Networks (DNNs)**-based applications, where minimizing data movement becomes a priority to achieve high computation efficiency. Due to the complexity of DNN, modern SA-based accelerators, with Google's **Tensor Processing Unit (TPU)** being a pioneering example [5], are equipped with their own ISA supporting single-instruction multiple data execution to process large vectors of data in one clock cycle.

However, when DNN models are employed in safety-critical applications, focusing only on computational efficiency is insufficient. Indeed, ensuring the correctness of the inference execution is as important as the performance. For this reason, several works proposed methodologies to address hardware fault detection in the SA accelerator [6] or fault masking acting on the DNN model [7, 8]. Still, such solutions that target reconfigurable platforms, such as **Field-Programmable Gate Arrays (FPGAs)**, have several limitations, including non-negligible resource overhead, limited runtime applicability, and lack of support for error correction capabilities, which are all essential requirement when running safety-critical applications.

Despite their flexibility and performance, one of the key limitations of SRAM-based FPGAs in radiation environments is the susceptibility of their Configuration Memory (CRAM) to radiation-induced faults. In space, electronic systems are exposed to a broad spectrum of high-energy particles, including protons, heavy ions, and electrons, originating from sources such as the Sun and **Galactic Cosmic Rays (GCRs)**. When these particles interact with the FPGA, they may trigger various types of radiation effects, depending on both the particle's energy and the specific region of the device that is impacted [9].

Among these, the most frequent phenomenon is the **Single Event Upset (SEU)**, a transient error in which a charged particle causes a corruption of the logic value of a memory element. The CRAM, which stores configuration data that defines the implemented logic, is particularly sensitive to SEUs. Although SEUs do not cause permanent physical damage, unlike destructive effects such as Single Event Burnout or Single Event Latch-Up, they can significantly disrupt the nominal behavior of the FPGA application, leading to malfunctioning logic, corrupted control paths, or faulty data processing. Even a single bit flip may compromise the success of an entire mission. A straightforward approach to addressing hardware faults in the implemented circuit due to configuration memory corruption is to reload the configuration data (CDATA) to restore the correct circuit functionalities using the uncorrupted configuration bitstream. While a reconfiguration latency in the order of seconds can be acceptable for offline or delay-tolerant workloads, it might be unacceptable in safety-critical systems and applications requiring continuous operation [10]. For instance, in on-board processing systems supporting time-critical functions (e.g., guidance, navigation and control or autonomous monitoring/FDIR), a loss of acceleration or processing availability for several seconds can lead to missed deadlines, delayed anomaly detection, temporary loss of situational awareness, or accumulation of unprocessed sensor data, ultimately degrading the system's safety. Moreover, runtime fault detection in FPGA devices is often invasive to applications and computational systems. Standard fault detection methodologies may require additional modules allocated along with the implemented circuit, like SEM-IP [11], or to perform periodic memory readback to compare with a golden configuration, eventually triggering complete reconfiguration upon error detection. The former approach also diminishes the resources dedicated to computational

units, and additionally, correcting the error does not consider errors already propagated in the user logic between the error and the correction. The latter fails to meet the real-time constraints of DNN, as it can take several seconds to detect and correct an error, potentially leading to catastrophic failures or service disruption in safety-critical systems.

Additionally, SEU sensitivity is strongly dependent on the underlying semiconductor technology. Not all FPGA technologies behave the same way in space; factors such as feature size, fabrication process, and architectural redundancy play critical roles in determining a device's vulnerability. Devices built using newer, more compact process nodes often exhibit increased sensitivity to SEUs due to lower critical charge and smaller node capacitances.

Moreover, the frequency and severity of radiation effects during a space mission are not constant, but highly dependent on the space environment. The radiation environment is shaped by a combination of orbital parameters (e.g., **Low Earth Orbit (LEO)** vs. **Geostationary Orbit (GEO)** vs. deep space) and temporal factors such as solar activity. For example, regions like the South Atlantic Anomaly or polar orbits expose spacecraft to elevated fluxes of trapped particles, while solar flares can temporarily increase particle intensity by orders of magnitude. As a result, reliable deployment of SRAM-based FPGAs in space demands careful consideration of both device-level sensitivity, mission-specific environmental conditions, and circuit characteristics.

1.1 Main Contributions

This work is based on and extend the RePAIR platform [12], which is a heterogeneous computing platform integrating and extending an open source RISC-V processor [13] and an open source TPU-like accelerator [14]. RePAIR targets reconfigurable devices and offers fault detection, fault correction, and recovery capability, preserving high performance, and minimum overhead.

To the best of our knowledge, this is the first platform to combine performance optimization and fault tolerance in this manner. The main innovations and contributions are as follows:

- —Design and implementation of the RePAIR platform, integrating a RISC-V processor with a SA-based accelerator.
- —Extension of TinyTPU ISA [14] for Runtime Self-Test: The ISA of the AI accelerator is extended to enable runtime self-test capabilities. This feature allows the detection of structural faults in the accelerator datapath with minimal impact on the inference process.
- —Dual Inference Modes: the computational platform can execute either in a plain mode for standard inference operation without additional overhead or in testing mode, which introduces a checksum test for the current inference workload at the cost of three additional clock cycles per matrix multiplication. This mechanism enables fault detection during inference without disrupting the normal operational flow with an accelerator area overhead of only 0.31%.
- —Fault Detection, Correction, and Recovery: When a fault is detected in the TPU datapath, the accelerator notifies the RISC-V processor. The processor triggers a **Dynamic Partial Reconfiguration (DPR)** to reload the bitstream section associated with the faulty accelerator while preserving its ongoing correct workload. By leveraging partial reconfiguration, the RePAIR platform was first implemented on the AMD KCU105 [12] and subsequently ported to a second FPGA platform, the AMD ZCU102, thereby confirming that the proposed platform is not hardware-dependent and can be implemented across different reconfigurable devices. Experimental results show that RePAIR reduces system downtime by up to 900× on the KCU105 and 1,400× on the ZCU102 compared to conventional approaches. This ensures rapid fault recovery and enhances system availability. The recovery mechanism resumes the workload from the last correctly executed instructions. This approach limits the inference

execution overhead to 30% in the worst-case scenario, compared to full device reconfiguration, which can result in up to 96% overhead.

— Accumulated fault injection campaigns to assess the reliability of the proposed approach and the effectiveness of the employed mitigation techniques. Environmental analysis was conducted to characterize realistic operating conditions in the space environment. This characterization has been combined with the results obtained from the fault injection experiments to estimate the **Mean Time to Failure (MTTF)** of the system under real-case mission scenarios. This evaluation underlines the effectiveness, portability, and scalability of the proposed techniques, highlighting their suitability for deployment in space-oriented FPGA-based accelerators.

By combining a device-agnostic error-detection technique that leverages already available computational resources, a ready-to-use and lightweight RISC-V core, implemented with limited hardware resources even when **Triple Modular Redundancy (TMR)**-mitigated, and the reconfigurability of FPGAs enabled by DPR, we realize a platform for efficient and reliable FPGA-oriented AI inference. Although DPR is inherently FPGA- and design-dependent, since it requires device-specific integration (e.g., definition of reconfigurable regions, floorplanning constraints, and toolchain support), the technological independence of our approach lies in the proposed fault-detection and recovery strategy, which operates at the accelerator datapath level and can be applied across different reconfigurable platforms. In this context, DPR acts as an enabling mechanism for localized repair, while the overall runtime detection–recovery–resumption flow remains portable and describable at the HDL/C level. To further validate the effectiveness of the proposed approach, we implemented and evaluated the platform across two different FPGA families, namely the UltraScale+ ZCU102 and the UltraScale KCU105. The fault-injection results underline the efficiency of the proposed solution in terms of timing, reliability, and area overhead, without relying on technology-specific assumptions, as the proposed methodology operates purely at the HDL level. Moreover, deploying RePAIR on multiple boards and validating the complete platform through fault-injection campaigns and MTTF estimation enables an assessment under realistic mission-oriented assumptions. Overall, this extended evaluation extend what was presented in [12], further supporting the effectiveness of the proposed methodology and the robustness of the RePAIR platform against radiation induced faults.

The content of the article is structured as follows. Section 2 introduces the related works. Section 3 provides an overview of the radiation effects in the space environment. Section 4 details the SA architecture and how it is used in the TPU accelerator. Section 5 proposes the fault detection mechanism, while Section 6 outlines the integration of the RISC-V processor with the accelerator. Section 7 presents the environmental analysis, Section 8 introduces the experimental results, and Section 9 provides the conclusion.

2 State of the Art

Over the years, several heterogeneous processor–coprocessor architectures have been proposed to accelerate compute-intensive workloads by coupling a general-purpose processor with dedicated hardware engines, with particular focus on FPGA-based SoCs. In the context of AI inference, commercial SoC platforms (e.g., ARM-based MPSoCs) have been widely adopted to host CNN accelerators, demonstrating the effectiveness of offloading convolutional and matrix-intensive kernels to specialized datapaths, while maintaining software programmability and system-level control on the host processor. This design paradigm has enabled high throughput and energy-efficiency improvements over CPU-only execution, making it a common choice for embedded and edge AI deployments [15, 16].

Such architectures became even more relevant with the introduction of the RISC-V ISA, leading to the development of various computing platforms [17] to address the needs of many domains of targeting DNN applications [18–20], with remarkable results. For instance, authors in [3] propose a custom, high-performance, and multithread library for convolutional operations targeting RISC-V processors. Results show extremely good performance in terms of Floating-Point Operations Per Second, even when compared to other processors, such as ARM ones, proving their suitability for carrying out DNN operations. One of the main characteristics of RISC-V processors is the open source nature of the ISA, allowing for easier pipeline modification to support custom instruction execution and efficient coupling with external modules. Following this trend, authors in [21] propose a custom architecture that couples a 64-bit RISC-V architecture, Ariane, with a co-processor for DNN inference applications. The possibility of extending the ISA of the RISC grants to achieve faster and more efficient DDR access, thereby improving overall execution time. Authors in [4] developed an architecture that embeds custom modules and co-processors in the RI5CY processor pipeline to support GEMM operations. The acceleration of GEMM operation is achieved through the ISA extension, which provides three custom **Single Instruction Multiple Data (SIMD)** instructions to be executed depending on the selected parallelism. Similarly, authors in [22] propose and compare two similar designs, evaluating the differences in terms of execution speedup when co-processors are instantiated inside or outside the RISC-V pipeline. In the first approach, the co-processor is coupled with the 64-bit Rocket architecture through the TileLink Bus, while in the latter solution, the matrix multiply unit is directly instantiated inside the soft-processor pipeline. Results show a speedup of 1.3× for the latter method with respect to the external coupling, while both grant improved performances against the plain version of Rocket architecture.

However, these processor–coprocessor platforms primarily focus on performance, leaving a gap in the adoption of DNN reliability for critical domains. When operating in radiation-prone environments such as space, reliability becomes a key requirement, especially when platforms are implemented on SRAM-based FPGAs, where **Single Event Effects (SEE)** and multiple bit upsets may corrupt configuration memory and compromise system availability. In this context, mitigation techniques such as TMR or **Duplication With Comparison (DWC)** are often combined with error-correction and recovery mechanisms—such as configuration-memory scrubbing or DPR—to limit the consequences of configuration upsets. Authors in [23] propose a custom ARM-based processor–coprocessor architecture combining mitigation techniques (TMR/DWC) with DPR to enable efficient correction of faulty modules; however, such approaches can incur significant hardware overhead, which is often a limiting factor on FPGA-based platforms. A similar approach is adopted in [24], where a hypervisor-based architecture monitors triplicated coprocessors and restores faulty instances through DPR. On the other hand, authors in [25] propose a processor–coprocessor architecture where DPR is used both for error correction and for runtime functional adaptation; however, fault detection mainly relies on SEM-IP-based scrubbing, which can introduce non-negligible detection latency. Overall, while these approaches enable runtime fault management, they may entail either considerable hardware overhead or high detection latency, which can reduce their suitability for safety-critical applications.

On the other hand, authors in [26] proposed an online error-detection mechanism for FPGA-based LLM accelerators, with DPR used to recover faulty modules. However, their solution requires training a separate classifier tailored to each model and relies on software-emulated faults rather than configuration-memory (CRAM) corruption, making it less representative of radiation-induced configuration upsets. Other self-test routines have been proposed over the years. However, they often require additional dedicated hardware resources to execute runtime tests, or they cannot be performed transparently at runtime, forcing the accelerator to pause its workload to execute self-test sequences, as in [27, 28].

Through this article, we fill the gaps detailed above by proposing a custom platform combining a lightweight RISC-V processor and a TPU accelerator oriented to enhance the system's dependability by combining three features: an error detection technique based on checksum, a fault correction mechanism based on FPGA reconfiguration capability, and an execution recovery mechanism for resuming execution from the last correct computation. In our proposed approach, we ensure runtime error detection capabilities in the DNN accelerator by extending the ISA of a TPU, coupled with a processor system, with custom instructions that detect and notify faults. We not only notify the RISC-V processor system that a fault is affecting the datapath, but we also trigger a DPR of the accelerator to correct the soft errors and recover the DNN inference from the last correct operation. As a result, we proposed the RePAIR platform, which targets performance and reliability with minimal performance penalties and resource overhead.

3 Background on Radiation Effects on SRAM-Based FPGA

FPGAs have emerged as a compelling technology for safety-critical applications, particularly in space systems, due to their flexibility, in-flight reconfigurability, and ability to implement complex, mission-specific digital architectures. These devices are extensively adopted as spacecraft onboard computing elements, serving roles that span from specialized accelerators for algorithmic processing to central control units. Notable examples include NASA's Double Asteroid Redirection Test [29] and ESA's HERA mission [30], both of which incorporated FPGA-based platforms in their flight hardware.

Despite these advantages, the use of **Commercial Off-the-Shelf (COTS)** FPGAs in radiation-prone environments introduces substantial reliability challenges, primarily due to their vulnerability to SEEs [31]. Among these, SEUs are particularly concerning. SEUs occur when ionizing particles, such as protons, neutrons, or heavy ions, interact with the silicon substrate and deposit sufficient charge to flip the state of a memory element. In SRAM-based FPGAs, the most critical of these elements is the CRAM, which encodes the device's routing and logic configuration. A single upset in CRAM can result in an unintentional modification of the implemented circuit, potentially affecting system functionality [32].

While SEUs are often associated with space missions and deep-space radiation, they are not exclusive to extraterrestrial environments. At high altitudes, secondary neutrons generated by cosmic ray interactions with the atmosphere can induce SEUs in avionics systems, posing significant risks to flight safety. Furthermore, SEUs also affect ground-based applications exposed to artificial or localized sources of radiation. For instance, high-energy physics experiments, such as those at CERN's Large Hadron Collider, place FPGAs near intense particle collisions, subjecting them to considerable radiation levels. Similarly, medical environments, particularly in proton therapy and heavy-ion cancer treatments, generate particle fluxes that may upset nearby FPGA-based instrumentation. These use cases clearly demonstrate that SEU-induced reliability concerns extend well beyond the aerospace domain, affecting a diverse range of mission-critical applications.

At the circuit level, an SEU in CRAM can compromise system functionality in various ways. For example, an upset in a configuration bit controlling a Programmable Interconnection Point can unintentionally enable or disable routing paths, leading to short circuits, open nets, or misrouted signals. Likewise, an SEU in a Look-Up Table configuration can modify logic functions, altering the application's intended behavior. On the other hand, if the upset affects a configuration bit associated with an unused resource, the fault may remain latent and go undetected during regular operation. These scenarios underscore the importance of understanding not only the overall sensitivity of the device but also the location and criticality of each configuration bit in the context of the implemented design.

Crucially, the sensitivity of SRAM-based FPGAs to SEUs is not uniform across all technologies [33]. It depends on various physical and technological factors, including the process node, layout geometry, doping concentrations, and the sensitive volume of the memory cell.

Radiation testing under controlled particle beams is the most direct method to quantify the sensitivity of an FPGA to SEUs. During these experiments, the device under test is exposed to a known flux of ionizing particles while its configuration and behavior are monitored for bit flips or functional failures. These tests are typically conducted at dedicated facilities around the world. The output of this characterization is the SEU cross-section, defined as the probability that an incident particle will induce an upset. The cross-section is typically expressed in units of cm^2/bit or cm^2/device, and is a fundamental metric derived from radiation testing that quantifies the likelihood of configuration corruption under a given particle fluence. The SEU cross-section, a key metric extracted from such campaigns, is defined as:

$$\sigma_{SEU} = \frac{N_{SEU}}{\Phi},$$

where N_{SEU} is the number of observed bit upsets and Φ is the particle fluence (particles per cm^2), representing the areal density of particles incident on the device during irradiation testing. It provides a probabilistic measure of how likely a configuration bit is to be flipped by an incoming particle at a given energy. A higher cross-section indicates greater vulnerability of the configuration memory to SEUs, and thus a higher risk of functional failure in a radiation environment.

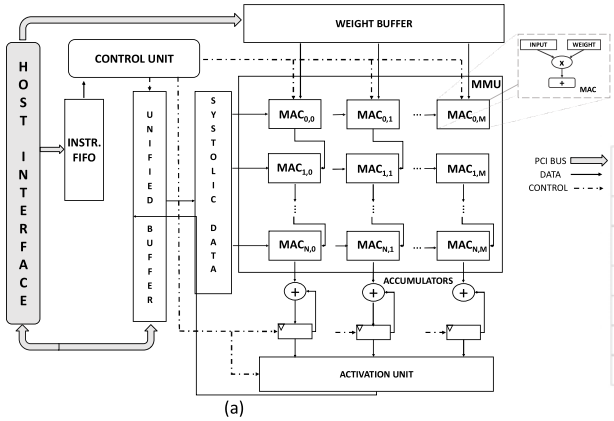
4 Background on SAs and TPU-Like Architecture

SAs are a class of parallel architectures originally proposed in the early 1980s, with the core idea of organizing computation as a regular grid of PEs that rhythmically exchange data with their nearest neighbors [34]. Early research explored their application to linear algebra [35] and signal processing [36] but practical adoption was limited by the semiconductor technology of the time. In the last decade, SAs have experienced a significant resurgence, driven primarily by the rapid growth of machine learning workloads [37]. A major milestone was the introduction of Google's TPU in 2018 [5], which demonstrated that large, highly regular arrays of simple PEs could be efficiently integrated on a single chip using advanced technology nodes. Modern process scaling enables hundreds or thousands of PEs to be placed on-die, operating at high throughput and energy efficiency, while being supported by sufficient on-chip memory bandwidth.

TPU-like accelerators are specifically designed to accelerate neural network inference (and, in later generations, training). These workloads are primarily dominated by **Multiply-and-Accumulate (MAC)** operations, which naturally arise from the mathematical formulation of neural networks. Fully connected layers, convolutional layers, and even modern attention mechanisms can all be expressed primarily as matrix–matrix or matrix–vector multiplications. Consequently, optimizing matrix multiplication directly translates into accelerating the core computation of neural networks.

Matrix multiplication is particularly well-suited to SA architectures. In a typical SA implementation, the interconnection layout of PEs is carefully crafted to enable direct links between PEs, allowing for a high degree of spatial and temporal reuse of data, and thereby avoiding the need for memory accesses to store and load intermediate results. Specifically, each PE performs a simple MAC operation and passes partial results and operands to neighboring PEs in a highly regular, pipelined fashion. Since data movement is significantly more energy-expensive than computation, this localized communication model results in substantial gains in energy efficiency.

Furthermore, the regular structure of SA simplifies control logic and enables aggressive optimization for area and power. By specializing the PEs for MAC operations and tailoring the dataflow



PCI BUS
DATA
CONTROL

Instr. opcode 8-bit	Buffer Addr. 24-bit	Accumulator Addr. 16-bit	Calculation Length 32-bit
matmul	data source	result destination	# vectors
load weights	data source		# vectors
activation	result destination	data source	# vectors
synchronize	not used	not used	not used
halt	not used	not used	not used
nop	not used	not used	not used

(b)

Fig. 1. (a) Overview on the TPU-like architecture and (b) The TinyTPU ISA.

to matrix multiplication, TPU-like architectures sacrifice generality in favor of efficiency. This specialization is well-aligned with neural network inference, where computational patterns are predictable and highly structured. As a consequence, TPU-like accelerators were originally conceived as relatively simple yet highly efficient co-processors, tightly coupled to a host CPU and equipped with a small, domain-specific ISA. Figure 1(a) provides an overview on the TPU-like architecture.

4.1 TinyTPU

In this work, we adopt TinyTPU [14], an open source accelerator architecture that closely reflects the design principles of Google's first-generation TPU. TinyTPU exposes a Complex Instruction Set Computing architecture. Its instructions are wide and composed of multiple fields, each controlling different aspects of data movement and computation. This design choice enables the execution of complex operations, such as performing MAC operations over several input vectors using a single instruction. Figure 1(b) illustrates the TinyTPU ISA. Each instruction contains several fields, among which the *Calculation Length* field plays a key role. This field specifies the number of weight or input vectors to be fetched from memory and processed, starting from a given base address. In this way, a single instruction can orchestrate the execution of a sequence of matrix operations within the SA.

Similar to the commercial TPU implementation, TinyTPU features three distinct memory structures: a *Weights Buffer*, dedicated to storing neural network parameters; a *Unified Buffer*, used for both input activations and output results; and an *instruction FIFO*, which decouples instruction issue from execution. Additionally, the architecture features a bank of accumulator registers, which are utilized to store and accumulate intermediate and final results generated by the SA.

The semantics of the buffer address fields depend on the type of instruction being executed. For the *load weights* instruction, the buffer address field specifies the starting memory location from which a given number of weight vectors are fetched and loaded into the two-dimensional grid of MAC units composing the SA. Conversely, the *matmul* instruction uses the unified buffer address field to identify the memory region containing the input activation vectors. The accumulator address field specifies the destination in the accumulator register bank where the output vectors generated by the matrix–matrix multiplication are stored. This instruction encapsulates three distinct operations: reading input data from memory, performing computation within the SA, and writing the resulting partial or final outputs into the accumulator registers. Finally, the *activation* instruction operates on data stored in the accumulator register bank. The accumulator address

identifies the source registers from which a specified number of vectors are read, while the unified buffer address indicates the destination memory region where the activated output vectors are written back. This instruction completes the inference pipeline by applying the activation function (ReLU or Sigmoid) and storing the final results in memory.

4.1.1 Host Interface and System Integration. TinyTPU is conceived as an FPGA-implementable accelerator, enabling flexible integration within heterogeneous systems-on-chip. The architecture can be coupled either with a soft-core processor instantiated on the FPGA fabric or with a hardwired processor, such as the ARM cores available in most FPGA-based SoCs. In our experimental platform, TinyTPU is integrated with a RISC-V soft-core processor, which acts as the host CPU and is responsible for overall system control and orchestration. The communication between the host processor and TinyTPU is implemented through an AXI interface, allowing the accelerator to operate as a memory-mapped peripheral. From the host CPU's perspective, TinyTPU is accessed via standard memory read and write transactions, without requiring a dedicated communication protocol.

The interaction model is intentionally simple and exposes only a limited set of access points to the host. Specifically, the host CPU can:

(1) Write neural network weights into the *Weights Buffer*.
(2) Provide input data for inference by writing to the *Unified Buffer*.
(3) Issue low-level assembly instructions by pushing them into the *instruction FIFO*.

Once the instruction stream is issued, TinyTPU executes the inference autonomously, without further host intervention. Upon completion of the inference process, the accelerator raises a synchronization signal, which is mapped to an interrupt on the host CPU. This interrupt notifies the host that execution has completed, and the output data is ready for consumption. Finally, the host CPU retrieves the inference results by reading the corresponding locations in the *Unified Buffer.*

4.1.2 Execution Model and Programming Support. TinyTPU is a pure hardware inference accelerator whose execution model is entirely centered on GEMM executed on a SA following a weight-stationary dataflow. Neural network inference can be performed only if the computation is statically decomposed into a sequence of GEMM operations with explicit accumulation and activation stages, fully matching the spatial and temporal dataflow of the array. All scheduling, data placement, and control decisions are resolved at compile time and encoded in the instruction stream. A minimal Python framework is provided to automatically map fully connected layers onto GEMM kernels, generating low-level TinyTPU assembly instructions. Supporting convolutional neural networks required a substantial additional effort. In this work, we developed a custom compiler from scratch that translates TensorFlow quantized models into sequences of GEMM-based kernels. Convolutional layers are transformed into equivalent matrix multiplications, while explicitly managing operand streaming, partial-sum accumulation, and output write-back in accordance with the weight-stationary systolic dataflow, preserving numerical correctness and quantization semantics. Consequently, both the fault detection mechanism presented in the following section and the experimental validation are built around the GEMM execution flow on the core.

5 The Proposed Error Detection Mechanism

5.1 Fault Model

The proposed methodology introduces a novel approach for detecting computational errors caused by faults in the datapath of SAs, targeting implementation in reconfigurable computing platforms.

As mentioned in the previous section, these faults typically arise from the corruption of CRAM, a common issue in harsh environments like space applications.

We focus on three representative fault cases: (1) corruption in the arithmetic logic of a single MAC unit, which directly impacts the correctness of the output at that position; (2) corruption of a weight value stored locally, mimicking a SEU during operand fetch or storage, which affects all subsequent computations involving that specific weight; and (3) disruption of interconnect paths. For the latter, the spatial propagation of the fault depends on which signal path is affected: if the fault impacts the path delivering the input activation to a MAC, the error propagates horizontally across the row; conversely, if the path delivering the partial product is disrupted, the fault propagates vertically along the column. This fault model allows us to capture the structured and cascading effects that characterize failures in systolic dataflows and enables a more precise analysis of vulnerability and fault containment strategies within these architectures.

5.2 Limitation of Traditional Reliability Approaches

Traditional reliability enhancement techniques adopted in reconfigurable platforms are based on hardware and/or software redundancy, imposing significant overheads on computation, memory, and energy, making them impractical for DNN applications. Additionally, such techniques just mask the SEU-induced error, while not solving the fault. As a consequence, if no actions are taken, SEUs in CRAM will accumulate over time, eventually causing system failure. Given that, detecting and solving SEUs is as important as guaranteeing a correct output. For instance, while AMD's **Soft Error Mitigation (SEM)** IP can detect and restore CRAM faults in runtime, the detection latency, ranging from 25 ms for Kintex-7 to 13 ms for KU040 [11] devices, is incompatible with the real-time detection demands of DNN inference. Furthermore, SEM IP is resource-intensive, consuming up to 835 LUTs, 506 FFs, and multiple BRAM (Block RAM) blocks, reducing resources available for application-specific designs. Another aspect to consider is that using SEM-IP provokes operational frequency design constraints. Indeed, the vendor datasheet imposes a max frequency of 100 MHz, which is incompatible with high-performance computing platforms [11]. Finally, correcting configuration memory will not correct previously erroneous generated output or faulty state, creating transient errors that could manifest unexpectedly in the future. Another potential scenario is full device reconfiguration, which does not require additional hardware resources but has two significant drawbacks. First, since the FPGA reboots during this process, the program restarts from the beginning, resulting in the loss of all previously completed computations, therefore increasing the inference execution overhead. Second, full device reconfiguration is time-intensive, taking several seconds to complete—a duration that grows with the device size—substantially prolonging system downtime. Considering KCU105 device, the full board reconfiguration takes 13 seconds while moving to a smaller device as the Ultrascale+ PYNQ-ZU, it requires 6.9 seconds.

5.3 Proposed Approach

As discussed in detail in Section 4, since SAs are specifically designed to accelerate matrix multiplication, this work focuses on detecting faults that occur during GEMM execution. Mapping either convolutional or fully connected layers onto a SA requires not only translating the original model into GEMM form, but also applying operation tiling, whereby large matrix multiplications are decomposed into a sequence of smaller matrix multiplication operations whose partial results are accumulated to produce the final output. Consequently, faults arising during any intermediate matrix multiplication may propagate through subsequent accumulations and corrupt the final inference result.

The goal of this work is therefore to develop a lightweight fault detection method capable of identifying faults at the level of individual matmul executions, rather than only at the final output. Early detection at this granularity allows fault propagation to be prevented and enables timely recovery actions, as the one proposed in this work through partial reconfiguration. While the ideal solution would be to monitor and verify the computation of each PE independently, such a fine-grained approach would introduce substantial hardware overhead in terms of additional logic and parallel checking units. This overhead would negatively impact both performance and reliability, especially in FPGA-based implementations, where increased area usage directly correlates with a higher probability of CRAM upsets affecting active resources. For this reason, and in line with prior work [6, 38], we do not target PE-level fault detection. Instead, we adopt a column-wise checking strategy, exploiting the intrinsic dataflow of the SA. For an $N \times N$ SA, this approach enables N independent checks per matmul, one per column, providing fault localization at the column level. This represents a significant improvement over approaches that only detect the presence or absence of a fault in the overall computation, while still avoiding the cost of fine-grained monitoring. A key advantage of the proposed method is that it introduces no additional computing resources for checksum generation. Instead, fault detection is achieved by reusing the existing computational paths of the SA, making the approach particularly suitable for resource-constrained FPGA platforms. As a result, the method aligns well with the requirements of high-performance, safety-critical systems, where both reliability and area efficiency are paramount.

The proposed fault detection methodology combines key elements of **Algorithm-Based Fault Tolerance (ABFT)** for SAs and scan chain techniques. Similar to SA-oriented ABFT [6, 38], fault detection is based on checksum computation over the data processed by the accelerator. However, inspired by scan-chain techniques [28, 39], these checksums are generated by propagating carefully selected test patterns through the functional datapath of the SA itself, rather than through dedicated hardware.

During normal operation, after the neural network weights are loaded into the PE grid, each $PE_{i,j}$ performs a MAC operation by multiplying the incoming input value by its locally stored weight $w_{i,j}$ and accumulating the partial sum received from the $PE_{i-1,j}$. In the next clock cycle, new input data feeds the $PE_{i,j}$, while the previous input data is propagated to the $PE_{i,j+1}$. Input data propagate horizontally from left to right, while partial sums propagate vertically from top to bottom within each column, following a weight-stationary dataflow. The proposed fault detection mechanism exploits the input functional paths and the column-wise partial product functional path to propagate test patterns that generate both a checksum and its complement for the weights loaded in each column.

The availability of base and complemented checksums in consecutive clock cycles allows the system to distinguish between transient faults (soft errors) and structural faults caused by CRAM corruption. This distinction is crucial: transient faults can be mitigated by reloading the weights and re-executing the computation at the software level, whereas structural faults require CRAM refresh using correct configuration data.

Going into detail on the checksum generation shown in Figure 2, two test patterns are applied as input vectors to the SA in two consecutive clock cycles. These test vectors have n elements when considering a PE grid of size $n \times m$. These test vectors are built to produce checksum values C_{SAj}, corresponding to the sum of the weights loaded in the processing grid at column j, and its complementary value $\overline{C_{SAJ}}$ as reported in Equations (1) and (2). To compute C_{SAj} it is sufficient to have all the elements of the first vector equal to 1. In contrast, to produce $\overline{C_{SAJ}}$ all elements of the second vector are equal to -1, and the second operand of the adders of the SA first MACs row is

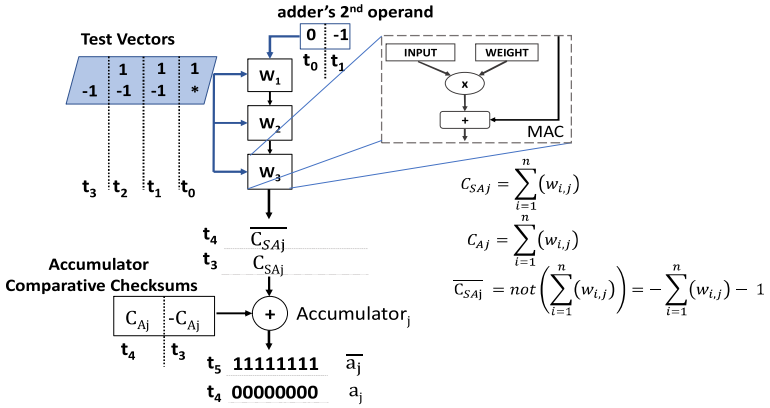


Fig. 2. Dataflow of the proposed fault detection mechanism considering the j column of MACs and the j Accumulator [12].

fed with −1:

$$C_{SAj} = \sum_{i=1}^{n} \left(w_{i,j}\right) \qquad \text{with } j = 0, 1, \ldots, m, \tag{1}$$

$$\overline{C_{SAj}} = not\left(\sum_{i=1}^{n} \left(w_{i,j}\right)\right) = -\sum_{i=1}^{n} \left(w_{i,j}\right) - 1 \quad \text{with } j = 0, 1, \ldots, m. \tag{2}$$

Since the two test vectors used for checksum computation have their LSB set to 1, an additional vector of 0 s must be propagated to evaluate the flipping of this bit. This ensures that no stuck-at-1 fault affects it. The SA will produce all 0 s in case no fault affects this bit. For the ABFT approach to be effective, we must compare the generated checksums to those produced by another unit to detect any mismatch signaling a possible error. Commonly, TPU architecture is equipped with an external bank of accumulators adjacent to the MAC grid to support tiling when the matrix multiplication does not fit in with the available resources. Our proposed approach exploits these external accumulators to compute the comparative checksum value. To do so, when the weights are loaded in the SA one vector at a time, they will also flow through the accumulators, which sum the weights data, generating C_{Aj}, as in Equation (3). The comparison between the checksum values produced by the SA and those produced by the accumulators will happen by exploiting the accumulators themselves:

$$C_{Aj} = \sum_{i=1}^{n} \left(w_{i,j}\right). \tag{3}$$

As soon as the SA checksums are produced, they are accumulated with those produced by the accumulators, following the traditional datapath of the accelerator. Hence, what changes with respect to the canonical use of the accumulation process is only the meaning of the operands. Indeed, these will not be the results of consecutive matrix multiplications to be merged but checksums values accumulated. Specifically, consider C_{SAj} and $\overline{C_{SAJ}}$ as checksum values direct and complemented produced by the column j of the SA, and C_{Aj} as the one produced by the accumulator j. Each accumulator A_j will perform the following operation:

$$a_j = C_{SAj} - C_{Aj}, \tag{4}$$

$$a_j^* = \overline{C_{SAj}} + C_{Aj}. \tag{5}$$

In the absence of faults, regardless of the weights' values, each a_j will assume value 0, and each $\overline{a_J}$ the value -1, i.e., all 1 s in binary). By examining the values assumed by each pair of $(a_j, \overline{a_J})$, for each column of PEs, we can identify and distinguish either the presence of a bitflip in a weight register when the pair a_j, a_J^* differs from the fault-free values, and values are complementary or a structural fault. Specifically, if $a_j, \overline{a_J}$ differs from the fault-free values and are not complementary, then we have a structural fault in datapath. To identify the fault location, we need to look at the checksums produced by the SA. Specifically, if the pair $a_j, \overline{a_J}$ is not complementary while C_{SAj} and $\overline{C_{SAJ}}$ are complementary, then the structural fault is located in the j accumulator. Otherwise, the fault is located in the j column of the SA.

With respect to previous works that required N adders [38], and 2N + 1 adders + 1 MAC [6] to compute checksum values in an SA of size N × N, our proposed approach does not require any additional resources while inducing a penalty of just three clock cycles to process the test vectors.

6 The Repair Platform

The RePAIR platform is based on a pair of open source computing cores: the NEORV32 RISC-V processor [13] and the TinyTPU accelerator [14]. In the context of DNN applications, the platform implements a traditional processor–coprocessor paradigm, where the NEORV32 orchestrates the execution by supplying the TinyTPU accelerator with a microcode transmitted via the accelerator's instruction FIFO to perform computation layer-by-layer. Upon completing the computation of a layer, NEORV32 retrieves the results and reforms the data to suit the requirements of the subsequent layer. Each layer execution can be conducted in a testing mode, employing the fault detection methodology described in Section 5.3. If a fault is detected in the TinyTPU datapath, the NEORV32 triggers and manages the DPR of the accelerator to correct the faulty datapath. Once the accelerator is operational again, NEORV32 resumes computation from the last successful execution. This mechanism ensures continuity of application execution while minimizing system downtime.

6.1 Proposed Platform

A schematic view of the implemented architecture and its internal connections, including the UART interface that can be used to communicate with an external host PC is shown in Figure 3. As can be seen, data exchange between the processor and the accelerator is handled through AXI interfaces and shared memory. One of the key advantages of the platform is the self-detection capability embedded in the TPU. Through a dedicated ISA extension, whose functionality is detailed in Section 5.3, the TPU can operate in testing mode, in which it autonomously detects errors affecting the datapath and signals such events via interrupts. This methodology introduces a minimal hardware overhead (0.31% with respect to the baseline implementation) and a constant latency overhead of only three clock cycles for each matrix multiplication executed in testing mode. All error events are captured by the NEORV32 processor through dedicated external GPIO interrupts.

Another key feature concerns the actions triggered upon fault detection, namely error correction and execution resumption. When an error is detected, the NEORV32 initiates the correction procedure via DPR and prepares the system to resume execution from the last correctly executed instruction once the fault has been mitigated. DPR [40] is handled by controlling the **Dynamic Function eXchange (DFX)** modules. In particular, the DFX Controller [41] oversees the reconfiguration process by fetching the configuration data (CDATA) from DDR and writing it to the configuration memory. A DFX decoupler placed between the TPU and the rest of the system ensures that interface signals are held at stable low values while partial reconfiguration is in progress.

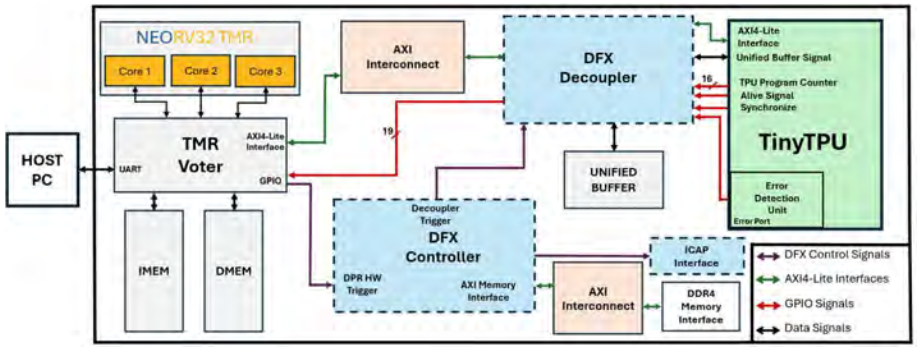


Fig. 3. The RePAIR platform [12].

If partial reconfiguration fails to correct the datapath error, a full FPGA reconfiguration is triggered as a last resort to restore functionality. While the TPU is being reconfigured, the NEORV32 prepares the instruction sequence to be sent to the accelerator as soon as it becomes operational again. Overall, the integration of the proposed fault detection techniques with the correction and execution-resumption mechanisms managed by NEORV32 and the DFX infrastructure enables a platform that supports efficient fault detection and recovery while significantly reducing system downtime in the presence of errors.

6.2 Neorv32

The NEORV32 [13] is an open source RISC-V processor based on a 4-stage pipe-lined multi-cycle architecture with a maximum operating frequency of 250 MHz supporting the base RV32I ISA. It is highly customizable with various additional modules and extensions. To include NEORV32 in the RePAIR platform, the AXI-Lite communication to interface the system with the TPU has been enabled. Moreover, the UART module has been enabled to support debugging and communication with a host PC. No ALU extensions have been enabled for our proposed platform since most of the computational work in DNN inference is carried out by the TPU accelerator, which performs matrix multiplications and applies activation functions. Since the platform's main goal is to provide performance and highly reliable execution, the NEORV processor has been hardened by design, applying TMR.

6.3 Recovery Routine

A new version of the *load_weights* and *matmul* instructions, *t_load_weights*, and *t_matmul*, respectively, have been added to the TPU ISA to support the fault detection methodology. The *t_load_weights* instruction performs two tasks. First, it loads weight values from memory into the SA. Second, it propagates these weights through the accumulators to compute the weight C_{Aj} checksums. At the moment of weight flowing, the accumulators could be busy accumulating the previous matrix multiplication results if needed by the program code, like when tiling is performed. To avoid delaying the pipeline execution or allocating additional hardware resources, we exploit the mapping of the accumulators on the on-chip DSP to operate the accumulators in SIMD mode. Indeed, modern FPGA devices support DSP operands on more than 48 bits (AMD Ultrascale 48 bits, AMD Versal 58 bits, Intel Agilex 54 bits). By doing so, while matrix multiplication results on 32 bits are accumulated (baseline behavior), the weights, represented on 8 bits, are accumulated to produce the checksums (testing behavior). Once all the weights for the instruction are read, the computed checksum values are stored in a dedicated location of the registers file, referred to as R0 and

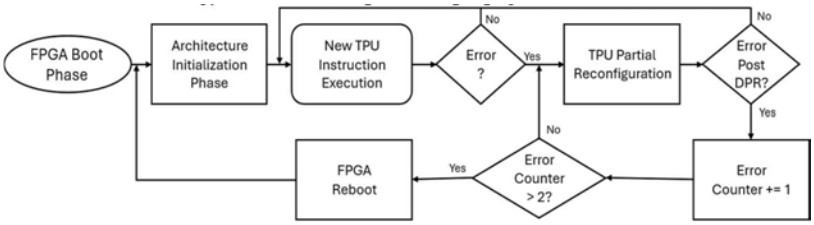


Fig. 4. Error detection and correction flow [12].

R1. The matrix multiply instruction always follows the read weight instruction, and similarly, *t_matmul* always follows the *t_load_weights*. In the testing mode, additional test vectors, required to compute the SA checksum C_{SAj} and $\overline{C_{SAJ}}$, detailed in the previous section, are appended to the input vectors, generating the checksums. Once the SA checksums are computed, they flow through the accumulators, just like partial products from the SA. However, they are always summed up with R0 and R1 content. This sum follows the operations described by Equations (4) and (5) in Section 5. Without faults, the results must be all 0 s and all 1 s, which are written back to R0 and R1. A XOR-based detection unit compares the contents of R0, R1, C_{SAj} and $\overline{C_{SAJ}}$ to detect faults and to perform diagnosis as explained in Section 5.3. The diagnosis unit notifies any detected error to NEORV using interruption, as well as the detailed information in the status. When the accelerator triggers the interruption, it needs to be reconfigured to fix errors in the datapath.

Considering the DNN workload, a single network layer is typically decomposed in hundreds of matrix multiplications. It is a programmer's choice to execute the full layer in testing mode, hence executing each matrix multiplication with the additional checksum computation or just some operations during the overall layer executions. When required by NEORV, the TPU starts fetching instructions from its instruction FIFO, prepared by the NEORV processor. In the testing mode, while a new instruction is fetched and executed, a stage of the TPU pipeline will evaluate the testing checksums produced by the previous instruction. If no errors are detected, the pipeline operation will continue normally. Differently, when an error is detected, the TPU triggers an interruption to the RISCV that starts the recovery procedure. In the recovery procedure, whose flow is highlighted in Figure 4, the TPU pipeline is flushed first, as no further instructions should be executed while the datapath is compromised. Simultaneously, the NEORV32 acquires detailed information about the specific instruction execution that triggered the error. This fault information is crucial for recovery: after reconfiguration, the program resumes from a correctly executed instruction. This approach ensures that the execution continues from the remaining instructions of the current DNN layer, avoiding a full restart.

If computations within a layer are fully executed in testing mode, the fault detection latency is confined to a single matrix multiplication operation, allowing all preceding executed instructions to be considered correct. However, if the layer is not processed entirely in testing mode—except for key operations like the first and last matrix multiplications (ensuring correct datapath behavior at entry and exit points of the layer)—the program must restart from the beginning of the faulty layer. This conservative strategy minimizes the potential propagation of undetected faults across layers.

The RISCV processor requests the DPR of the TPU through the DFX controller. Once the partial reconfiguration process is triggered, the NEORV32 waits for its completion. In the proposed design, this is achieved by including in the TPU architecture an alive signal that activates as soon as the accelerator becomes operational. The soft processor continuously polls this signal, and as soon as it transitions to 1, indicating that the TPU is functional again, the interrupt subroutine concludes, and normal execution flow resumes. However, if the partial reconfiguration fails, meaning the

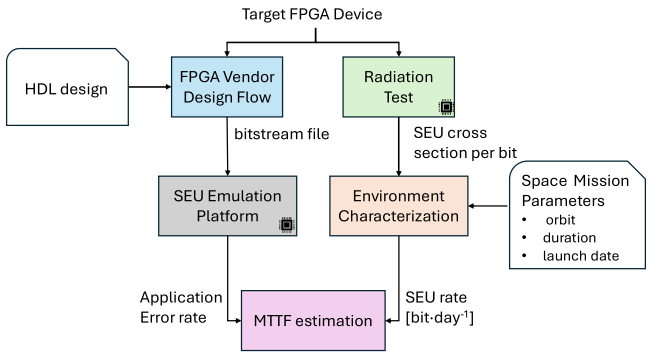


Fig. 5. Adopted MTTF flow combining technology sensitivity with space environment and application error rate.

soft processor detects consecutive TPU errors during two reconfiguration attempts, the system initiates a full-board reprogramming process to restore the correct behavior. Because the device undergoes complete reconfiguration, the memory content is not retained. This means that the platform performs a cold start.

7 Mission-Tailored Reliability Evaluation

A comprehensive estimation of system reliability and the effectiveness of proposed mitigation strategies must be grounded in realistic operating conditions. While radiation testing offers the advantage of directly mimicking the space environment and its failure sources, it is most commonly used to characterize the sensitivity of the underlying technology, rather than the implemented application. This is especially true for FPGA-based designs, where radiation test efforts are largely focused on understanding the vulnerability of the CRAM, particularly as technology nodes scale down [42]. As a result, radiation tests often quantify the error rate (e.g., SEUs per bit) under specific particle types and energies, but these data alone are insufficient to assess the reliability of a particular design operating in a real mission environment, focusing mainly on the device characterization rather than on the effect on the specific application. Moreover, due to the cost and complexity of radiation testing, such campaigns are typically deferred until the final validation stages.

To evaluate application-level resilience earlier in the design cycle, fault injections are commonly used to emulate the effects of radiation-induced errors. However, fault injection alone cannot predict how often these faults would occur in flight. This is where space environment modeling becomes essential: by combining radiation test data on device sensitivity with particle flux estimations tailored to specific mission orbits and correlating this with application-specific fault injection results, designers can derive a much more accurate and mission-relevant reliability assessment. The synergy among these approaches is thus key to bridging the gap between low-level sensitivity data and high-level application robustness.

To accurately estimate the MTTF of an FPGA-based design operating in space, our methodology, shown in Figure 5, combines two parallel flows: the design-level fault emulation path and the mission-specific space environment modeling path. On one side, we begin with the standard HDL design flow, targeting an FPGA implementation of a complex system, such as the RePAIR platform in our case study. After generating the bitstream, we employ SEU emulation platforms to simulate radiation-induced upsets in the CRAM. This can be done at runtime using tools like AMD SEM-IP, as we did in our design, or statically through bitstream corruption using platforms such as PyXEL [43].

Table 1. RePAIR Resource Utilization on KCU105 [12]

RePAIR Modules	LUTs	FFs	BRAMs	DSP
TinyTPU	4,294	7,564	111	210
TMR NEORV32	3,219	3,180	24	0
DPR Logic	1,185	989	0	0
Glue Logic Resources	13,874	17,670	89.5	3
Total (%)	9.31%	5.99%	37.41%	11.09%

To emulate realistic radiation effects and estimate the system's robustness, fault injection with accumulation is typically performed: SEUs are progressively injected into the bitstream until the system fails. The outcome of this process is the average number of CRAM bitflips the design can tolerate before failure. In parallel, space environment modeling tools like OMERE [44] by TRAD or SPENVIS [45] by ESA are used to simulate the radiation conditions for a specific mission orbit, be it LEO, GEO, or interplanetary. These tools are functionally equivalent and publicly available, and both provide a graphical user interface. The user manually configures the mission parameters such as orbit type, altitude, inclination, and shielding assumptions and supplies technology-specific radiation sensitivity data, typically in the form of SEU cross-section versus particle energy curves. Based on this information, the tools return an estimated SEU rate per bit per unit time (e.g., $\text{bit}^{-1}\cdot\text{day}^{-1}$), accounting for the dominant radiation sources of the selected environment (e.g., trapped protons in LEO or GCRs in GEO). The interaction between fault injection and space environment modeling is not automated, but rather performed at the methodological level. On one side, the user obtains from fault injection experiments a statistical measure of the design's tolerance to CRAM upsets (i.e., number of bitflips leading to failure). On the other side, SPENVIS or OMERE provides the expected bit-level upset rate for the target mission. These two independent results are then combined through simple analytical calculations (multiplication and division) to correlate the device-level SEU rate with the design-level failure statistics, ultimately enabling the estimation of the MTTF under realistic mission conditions [46]. For example, if a design fails after 60 SEUs on average and the predicted environment causes 1 SEU per bit per day, the expected MTTF is 60 days. Importantly, this value is highly sensitive to both the target technology and the orbital profile, meaning that even with the same design, the MTTF can vary dramatically across different mission scenarios, as will be proved in the following section.

8 Experimental Analysis

8.1 RePAIR Implementation Details

The RePAIR platform has been validated on two different boards: the AMD KCU105 Evaluation Board, based on CMOS technology, and the AMD ZCU102 Development Board, which uses FinFET technology. The TPU accelerator, which implements a 14×14 SA, uses the on-chip DSP blocks to implement all MACs and accumulator units. The weight buffers and unified buffers are mapped to BRAM resources. Minimal LUT usage is required for implementing control logic and glue logic. Tables 1 and 2 report the implementation details for the two processing cores.

The area overhead introduced by interfacing the DPR and the two cores, listed as glue logic in the table, remain within acceptable limits. Compared to a baseline version of the design that integrates the plain NEORV32 and TPU without error-detection features, the proposed RePAIR platform introduces only marginal resource overhead. Tables 3 and 4 report the resource utilization of the baseline design. When compared against RePAIR, DSP, and BRAM usage remains unchanged, as

Table 2. RePAIR Resource Utilization on ZCU102

RePAIR Modules	LUTs	FFs	BRAMs	DSP
TinyTPU	4,564	7,578	111	210
TMR NEORV32	3,261	3,117	24	0
DPR Logic	1,155	981	0	0
Glue Logic Resources	8,173	12,275	64	0
Total (%)	6.26%	4.37%	21.82%	8.33%

Table 3. Baseline Design Resource Utilization on KCU105

RePAIR Modules	LUTs	FFs	BRAMs	DSP
TinyTPU	3,912	6,872	111	210
NEORV32	1,088	1,042	24	0
Glue Logic Resources	12,987	16,322	89.5	3
Total (%)	7.43%	4.99%	37.41%	11.09%

Table 4. Baseline Design Resource Utilization on ZCU102

RePAIR Modules	LUTs	FFs	BRAMs	DSP
TinyTPU	4,173	6,906	111	210
NEORV32	1,088	1,042	24	0
Glue Logic Resources	7,384	11,372	64	0
Total (%)	4.61%	3.52%	21.82%	8.33%

expected, since these resources are not TMR-mitigated and are only used by the TPU datapath and the RISC-V subsystem. Differences are limited to the logic fabric: LUT and FF counts change due to the additional modules introduced by RePAIR (e.g., DPR management logic, the TPU error-detection circuitry, and the TMR-mitigated NEORV32). Overall, the overhead is extremely limited, with a maximum LUT increase of 1.88% on the KCU105, while all other variations remain below 1.5% with respect to the baseline.

Another useful metric to quantify the impact of the additional logic is the fraction of essential bits in the implemented design. Essential bits are defined by AMD as configuration bits whose corruption is more likely to disrupt the circuit behavior and compromise correct system operation. In our case, on both the ZCU102 and KCU105 boards, the essential-bit fraction increases by approximately 1% when moving from the baseline design to RePAIR. Specifically, it rises from 6.61% to 7.65% on the ZCU102 and from 13.63% to 14.9% on the KCU105. Overall, despite integrating runtime error detection, DPR-based correction/recovery, and TMR protection for the processor, the additional resource usage remains very limited, resulting in a negligible increase in the essential-bits count while preserving low resource utilization.

The primary difference in resource utilization between the two board implementations lies in the mechanism with which the systems manage communication between PL and DDR memory, which stores the partial bitstream data required for DPR. Although the TPU is capable of operating at higher frequencies when coupled with a hardwired ARM core in an FPGA-based SoC, reaching 177 MHz

in its original implementation delivered for a 28 nm Artix-7 FPGA [14], its tight coupling with the NEORV32 and the associated processor-side/interconnect constraints limit the achievable system frequency. Using a higher-performance RISC-V core and/or redesigning the processor–coprocessor interface could improve overall performance. An alternative approach is a multi-clock design, allowing the TPU to operate at the maximum achievable clock speed. To further increase the reliability of the design, we enabled the ECC in the BRAMs to prevent data corruption due to radiation-induced SEU.

8.2 Experimental Setup and Adopted Benchmarks

The experimental analysis focuses on two primary aspects of the platform: fault detection capability and dynamic reconfiguration properties. These have been evaluated with experiments using the actual hardware platform. SEU-induced structural faults in the TPU Datapath have been emulated by bitstream corruption to reproduce faults. To test fault detection capability, a fault injection campaign targeting the resources of the SA was conducted. The campaign injected 20,000 distinct faults in the configuration memory. Each fault has been evaluated individually. Bitflips in the CRAM were used to emulate various fault types, including stuck-at, bridge, conflict, and open faults. As a general-purpose benchmark, we selected two classification tasks, MNIST digit recognition and CIFAR-10. In addition to synthetic workloads, we implemented two space-oriented benchmark applications to emulate more realistic operational scenarios. The first leverages Earth observation imagery from the Sentinel-2 satellite [47], part of the Copernicus program, using land use and land cover datasets to develop a CNN benchmark representative of an actual on-board processing task. Combined with the known orbital parameters of Sentinel-2, this benchmark is particularly suitable for deriving realistic failure rate estimates by coupling space environment modeling with technology characterization and fault injection, as detailed in the experimental section.

To further demonstrate the versatility of the RePAIR platform, we implemented a second benchmark application that focuses on early satellite anomaly detection using neural networks. For this purpose, we used the ESA Anomalies Dataset [48], the first and largest open source telemetry anomaly database released by the European Space Agency, covering two missions (M1 and M2). The dataset spans 168 months of operation for M1 and approximately 43 months for M2, each containing over 700 million telemetry data points organized into channels corresponding to specific satellite subsystems (e.g., power, thermal, attitude control). Each record includes annotations for anomalies, rare nominal events, and communication gaps, along with the affected channels. Traditionally, anomaly detection in satellites has relied on rule-based or threshold-based systems, where parameter limits are defined using engineering models and expert knowledge. While interpretable and simple to implement, such approaches require continuous manual adjustment and are often unable to identify subtle or previously unseen faults. Given that, in previous work [49], we evaluated multiple models on this dataset, including Deep and Shallow **Fully Connected Neural Networks (FC-NN)**, Multi-Head Transformers, Support Vector Machines, Random Forest, Logistic Regression, and Gaussian Naïve Bayes, using the F0.5 score as the primary metric, in line with ESA's recommendation to penalize false positives more heavily due to their operational cost. Among these, the Deep FC-NN achieved the highest F0.5 score, precision, and recall, confirming its ability to capture complex nonlinear patterns and generalize effectively across missions and subsystems. This model was therefore ported to the RePAIR platform to emulate on-board anomaly detection, enabling the system to perform two complementary functions: (i) continuous self-test and fault detection in its own datapath, and (ii) execution of a machine learning model to detect faults in other satellite subsystems.

Given that both the SA and the integrated fault detection mechanism are specifically optimized for matrix–matrix multiplication, the selected benchmarks were deliberately composed exclusively

of convolutional and fully connected layers. This design choice ensures that the workloads fully exploit the parallelism of the PEs grid while enabling controlled observation of error propagation across layers. By doing so, we maximize computational stress on the accelerator while assessing fault effects under workloads with varying architectural complexity and dataset characteristics. The use of relatively simple datasets and models is primarily motivated by practical constraints of the hardware-based fault injection methodology, rather than by limitations of the proposed approach. In particular, SEU-induced structural faults are emulated through CRAM corruption of the FPGA bitstream, a realistic and widely adopted technique for reproducing radiation effects at the hardware level. However, this process is inherently time-consuming, as each injected fault requires reconfiguration, execution of the workload, and result collection. Executing significantly larger or more complex models for every injected fault would therefore render the fault injection campaign impractical, potentially extending the experimental duration to several months.

Importantly, the proposed fault detection and recovery mechanism is architecture- and datapath-dependent, rather than model-dependent. Regardless of the neural network complexity, any workload executed on TinyTPU must ultimately be decomposed into a sequence of GEMM operations to leverage both SA acceleration and the integrated fault detection strategy. Increasing model complexity thus translates into longer sequences of matrix multiplications and accumulations, affecting inference latency but not the detection granularity, coverage, or recovery capability of the proposed method.

Consequently, the approach naturally scales to larger datasets, deeper networks, and larger SAs without introducing additional hardware overhead or modifying the fault detection logic. The effectiveness of the method is tied to the regularity of GEMM execution, not to the semantic structure of the neural network. Overall, applications dominated by GEMM-based workloads represent the typical and most effective use case for SA architectures; therefore, the focus of this work is fully aligned with state-of-the-art design principles adopted by modern TPU-like accelerators, where performance, efficiency, and correctness are fundamentally driven by optimized matrix multiplication execution. Nevertheless, while simple models such as those used for MNIST were included to enable controlled and repeatable fault injection experiments, we also evaluated a fundamentally different workload based on time-series satellite anomaly detection, whose model size is 36.2× larger than the MNIST model. This demonstrates that the effectiveness of the proposed approach depends on the core GEMM execution on the SA, rather than on the semantic meaning or modality of the data being processed.

Since hardware deployment, the benchmark models were implemented using the quantized Qkeras TensorFlow library and trained from scratch to align with the hardware characteristics of the TinyTPU accelerators, especially regarding the quantized activation functions integrated within the core. The details of the models' implementation are provided in Table 5. For each injected fault, inference was performed on 10 randomly selected samples from the test dataset to analyze possible data-induced fault-masking effects during processing. The inferences have been executed in testing mode for all the computations operations.

8.3 Fault Detection Rate and System Recovery Time

The experimental results for fault detection are shown in Figure 6(a). A major concern when evaluating this kind of application is that faults may affect the confidence level of the prediction without changing the results of classification. In this work, we classify the observed events as either **Silent Data Corruption (SDC)** or *Misclassification*. In prior work on AI accelerators, such as [50, 51], SDCs are defined at the application/semantic level: a fault is considered an SDC only if it produces a user-meaningful deviation from a golden run (e.g., misclassification for image classification, changes in detected objects for object detection), while masked deviations are not

Table 5. Benchmark Characteristics

Dataset	Convolutional Layers (#)	Fully Connected Layers (#)	Parameters (#)	Accuracy (%)	F0.5 Score
MNIST	3	1	40,874	97	/
CIFAR10	6	1	91,648	83.4	/
Sentinel-2	3	1	39,472	98.3	/
Satellite-Anomalies	/	15	1,479,881	99.3	0.7642

counted as SDCs. In our work, we adopt a different notion, classifying an event as SDC when inference completes normally, the predicted class remains correct, and the confidence score (or output activation/logit) differs from the corresponding golden execution for the same input and model. In other words, corruption remains silent at the application level because it does not alter the final decision; yet, it still represents a deviation from fault-free computation, which may reduce the decision margin and increase vulnerability to future perturbations. This definition complements semantic SDC metrics by capturing "near-miss" corruptions that do not immediately cause misclassification but can contribute to accuracy degradation and fault accumulation over time.

We want to emphasize that the faults affecting designs implemented within FPGAs due to CRAM corruption are multiple and challenging to detect, largely due to the lack of documentation on CRAM and CDATA, which are not provided by vendors. Still, the proposed approach demonstrated robust fault resilience, achieving an average detection rate of approximately 94%, as shown in Figure 6(a). It is worth noting that the observed differences among benchmarks are not related to inference time, but rather to model robustness, decision margin, and task complexity. In TinyTPU, all neural networks are decomposed into sequences of matrix multiplication operations executed on the SA, and inference time scales with the number of such operations. However, all benchmarks are executed using the same execution policy, with the testing mode enabled for every matrix multiplication, ensuring a uniform fault detection granularity across all workloads. This explains why the detection rate remains consistent across benchmarks, as the proposed mechanism operates as a functional, datapath-level test that is agnostic to the semantic meaning of the processed data. In contrast, differences in misclassification rates are model-dependent. The higher misclassification rate observed for CIFAR-10 compared to MNIST, Sentinel-2, and satellite anomaly detection can be attributed to a combination of model robustness, decision margin, and task complexity. The CIFAR-10 model exhibits the lowest baseline accuracy (83.4%, as shown in Table 5), indicating less distinct decision boundaries and higher sensitivity to perturbations compared to the other models, whose accuracies are all equal or higher than 97%. As a result, even small fault-induced numerical deviations are more likely to cause the inference output to shift across a decision boundary, leading to misclassification. Moreover, CIFAR-10 is a multi-class image classification task characterized by higher intra-class variability and inter-class similarity than MNIST digits or the more structured patterns found in Sentinel-2 imagery and satellite telemetry time series. This increased complexity places greater reliance on fine-grained feature representations, making the model more susceptible to fault-induced perturbations in convolutional and fully connected layers.

For SDCs, although the corruption is silent from the application's perspective since no crash or obvious malfunction occurs, the proposed fault detection mechanism is still able to detect the underlying fault during computation, as it relies on the mathematical properties of the SA datapath rather than on the semantic meaning of the data. In particular, the detection mechanism

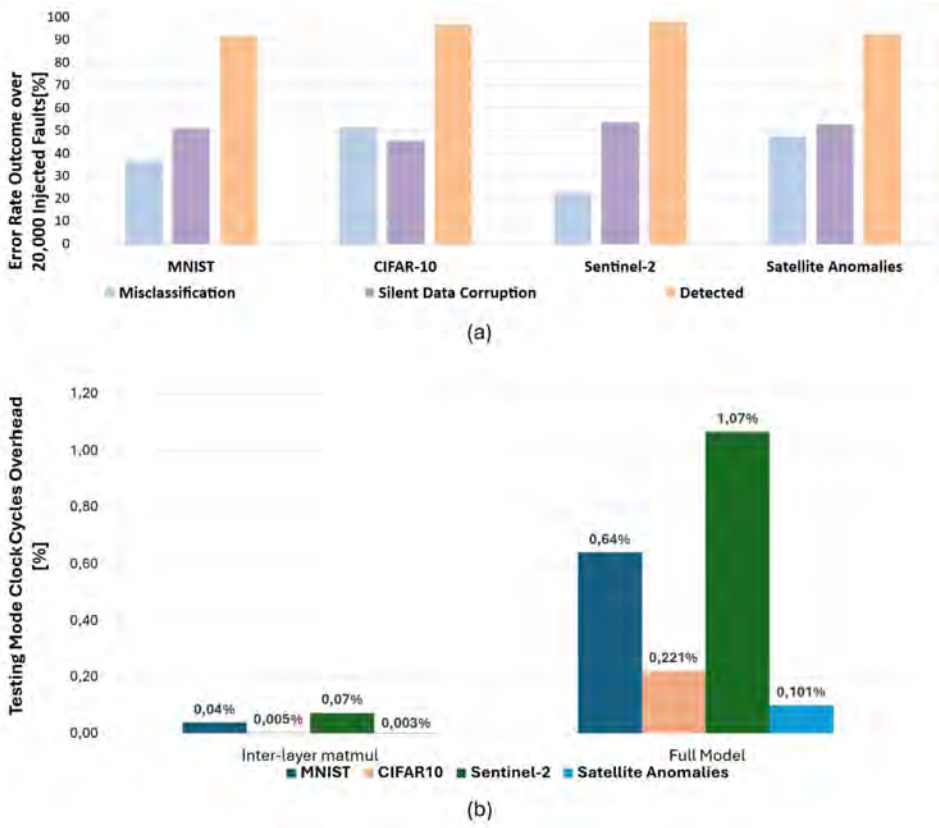


Fig. 6. (a) Fault injection results over 20,000 injected faults and (b) Maximum and minimum clock cycles overhead.

operates on the full computational capability of the SA, propagating and checking all bits involved in each MAC operation and fully exploiting the inherent parallelism of the operands. In contrast, the neural network output for each layer is subject to rounding and activation functions, which, in quantized inference, reduce intermediate results from 32-bit values back to int8 values. As a consequence, numerical errors that may be masked or truncated at the model output level can still be detected at the intermediate computation level by the proposed methodology for all the tested models, preventing fault accumulation and enabling timely corrective actions. Regarding performance overhead, it is important to highlight that it is related to the inference execution time, which is different for different inference tasks, in particular, 0.23 ms for MNIST and 1.8 ms for CIFAR10, 0.32 ms for Sentinel-2, and 29.7 ms for Satellite Anomalies, respectively, when operating at a 100 MHz clock frequency. Additionally, the fault detection mechanism introduces an overhead, but this overhead depends on how many instructions the developer wants to execute in testing mode. Reminding that the testing procedure introduces 3 clock cycles of penalty, the execution overhead is 3 clock cycles $*$ number of matmul instructions per layer $*$ number of layers, in the worst case, i.e., when all the matrix multiplication in all the DNN's layers are executed in testing mode. If the test mode is activated only during the last matmul of the layer, we have an overhead of 3 clock cycles $*$ number of layers while still detecting faults in each layer computation. The minimum and maximum relative overhead for the benchmark models is reported in Figure 6(b).

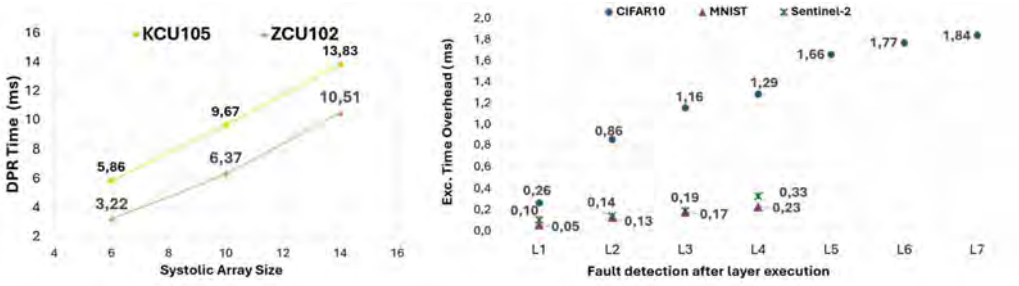


Fig. 7. (a) TPU DPR time related to the SA size for the KCU105 and ZCU102 boards; (b) the execution time overhead in full board reconfiguration, depending on when the fault is detected during the inference.

For all the benchmark applications, the overhead is negligible, and it decreases as the model complexity increases. The disadvantage of having a coarser fault detection is that in case of a fault, we need to recompute the computation of the whole layer instead of repeating only the last matrix multiplication, as will be discussed further.

The advantage regarding system downtime strictly relates to the amount of time required to apply DPR to the accelerator compared to the time required for reconfiguring the full device. The time required for DPR depends on the size of the SA since the larger the accelerator, the higher the DPR time, while the time for complete reconfiguration depends on the device size since larger devices require longer reconfiguration times. Specifically, the DPR time resulted in scaling linearly with the SA size, as shown in Figure 7(a). For instance, a DPR time of 14 ms is necessary to complete the reconfiguration of a 14×14 SA, the maximum SA size allowed by TinyTPU architecture. In contrast, considering the KCU105 device, its full board reconfiguration requires almost 13 s. Therefore, when mapping the accelerator to KCU105, we can benefit of an 872.1× reduction in time exploiting DPR. When implementing the platform on the ZCU102 development board, slightly different results were observed compared to the KCU105. For a 14×14 SA, the partial reconfiguration time decreased from 13.83 ms to 10.51 ms. This improvement is mainly due to the smaller partial bitstream size and differences in how partial reconfiguration is handled on the two devices. Specifically, on the ZCU102, the DPR process does not require uploading the target reconfigurable region with a "clearing bitstream" prior to loading the correct partial bitstream into the configuration memory. Consequently, despite the ICAP interface operating at the same speed on both the ZCU102 and KCU105, the absence of this additional step results in faster reconfiguration on the ZCU102.

Furthermore, the full board reconfiguration of the ZCU102 requires approximately 15 s, leading to an overall reduction time factor of 1327.2× when exploiting partial reconfiguration on the UltraScale+ device. Finally, reducing the SA dimensions from 14×14 to 10×10 and 6×6 results in partial reconfiguration times of 6.37 ms and 3.22 ms, respectively, for the ZCU102. These values are comparable in order of magnitude to those measured for the corresponding configurations on the KCU105 implementation.

In the case of a full reconfiguration, the program must restart the inference process from the beginning, assuming it can retrieve the input data again. Consequently, the execution overhead depends on the timing of the fault. In the best-case scenario, the fault is detected during the execution of the first instruction in the first layer of the DNN. Here, the system discards the result after executing the first instruction, reconfigures the board, and then completes the full inference process starting from the beginning. Conversely, in the worst-case scenario, the fault is detected at the end of the inference process, requiring the entire process to be repeated twice. In

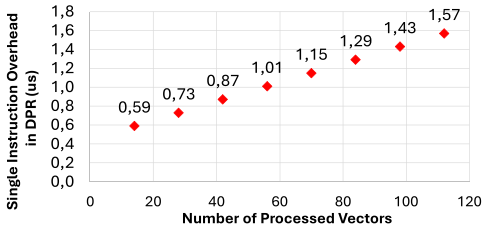


Fig. 8. Faulty instruction overhead in DPR as a function of processed vectors [12].

addition, some time is also required by the NEORV to boot before starting computation. Figure 7(b) illustrates the overhead as a function of the fault's timing for the benchmark CNNs. The Satellite Anomaly Detection benchmark is excluded from the chart because its overhead is roughly an order of magnitude higher than the other workloads, which would obscure the lower values. However, its trend is identical: the later the fault occurs, the higher the overhead. Specifically, detecting a fault in the first layer adds approximately 4 ms, detection around the middle of the model (e.g., the 7th layer) adds nearly 20 ms, and detection at the end requires re-running the full inference, resulting in an overhead of about 30 ms.

When DPR is enabled, the inference process does not start from the beginning after the reconfiguration. When the whole model is executed in testing mode, i.e., every matmul of each layer is executed in testing mode, the process resumes from the last correct instruction. This approach avoids discarding and executing computations that were correct again. The testing mode ensures the correctness of results up to the fault occurrence. The recovery of the execution from the last correct instructions is achieved by excluding the TPU input/output buffer from the DPR process. The buffer is hardened with ECC, preserving the TPU context during reconfiguration while protecting it from SEU.

The cost of the recovery procedure depends on the cost of the failed instruction, particularly the number of vectors processed by the instruction. Figure 8 illustrates the time overhead associated with executing the faulty instruction as a function of the processed vector count, ranging from a matrix multiplication with the size of operands of 14×14 (matching the TPU dimensions) to operands up to eight times larger. However, the additional time required for processing the same instruction after DPR, compared to a scenario where no fault is detected, is below 2 µs, which is negligible considering that the entire inference process for the simplest CNN model is 100 times longer.

However, if the layers are not fully executed in testing mode, for instance, only the first and last instructions are checked to ensure correct inter-layer processing, the program must restart from the last correctly executed layer. Consequently, the total execution time with DPR differs from the previous case and depends on the layers' complexity and the model architecture itself, as explained further.

Figure 9 illustrates a quantitative comparison of inference execution scenarios, focusing on the closest comparable cases involving testing instructions at each layer's entry and exit points, under full reconfiguration versus DPR. In the chart, the No-Fault scenario serves as the baseline, representing the execution with two testing instructions per layer with no fault detected; hence, the inference is executed with no interruptions. The other scenarios depict the total execution time when a fault occurs during different layers, which have different computational costs. These values account only for the computational overhead caused by restarting the program, excluding reconfiguration time. This exclusion is due to the significantly longer duration of full device reconfiguration—on the order of seconds—making it incomparable to the other measured values.

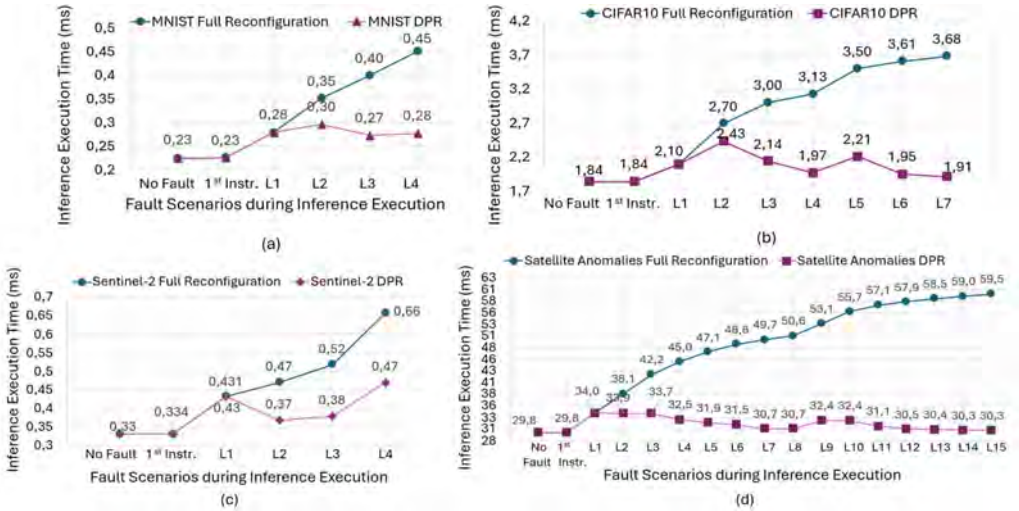


Fig. 9. Total inference execution time considering fault occurrences at different stages of the inference process with and without DPR for MNIST (a) [12], CIFAR10 (b) [12], Sentinel-2 (c) and Satellite Anomalies Detection (d).

The graphs show that the inference execution overhead introduced by partial or full re-execution of the program during inference exhibits similar trends for both CNN benchmarks. If the fault occurs during the execution of the first layer, the computational overhead is the same for both DPR and full reconfiguration. In this case, the only differentiating factors between the two techniques are the total reconfiguration time, which depends on the device's characteristics (e.g., size, ICAP port speed), and the partial reconfiguration time, which is influenced by the size of the accelerator.

Starting from the second layer, the use of DPR becomes significantly more advantageous. DPR allows the computations completed before the fault to be preserved, enabling the process to resume from the last successfully executed layer. In contrast, with full reconfiguration, the overhead increases as more layers are processed, as all computations must be restarted. For DPR, the worst-case scenario involves re-executing the most complex layer. In the benchmarks under study, this occurs in the second layer for both MNIST and CIFAR-10, leading to an increase in inference time of nearly 30% in both cases. For Satellite Anomalies and Sentinel-2, instead, it appears in the first layer, introducing approximately 30% and 40% overhead, respectively. For all evaluated cases, the proposed approach introduces an overhead that is significantly lower than that of the worst-case full-device reconfiguration, which incurs a more than 96% increase in execution time. As expected, when executing longer and more complex models with extended inference times, the relative cost of reconfiguration becomes progressively more affordable, further amplifying the advantage of the proposed method over full reconfiguration. Nevertheless, the focus of this work is not to amortize reconfiguration time through longer workloads, but to validate the correctness, reliability, and effectiveness of the proposed fault detection and recovery mechanism. Hardware fault injection through CRAM corruption is inherently time-consuming, and we therefore prioritized injecting a large number of faults individually to ensure that faults are consistently detected and that the associated reconfiguration behaves as intended. Executing longer inference models would not provide additional validation benefits, since the computational structure remains invariant across different models: both simple and complex networks are ultimately decomposed into sequences of matrix multiplications executed on the SA. Consequently, executing 10 or 100 matrix multiplications

in testing mode affects only the total execution time, without changing the fault detection behavior or the reconfiguration mechanism itself.

Finally, DPR offers the advantage of significantly shorter reconfiguration times—on the order of milliseconds—compared to full reconfiguration, which can take several seconds for a medium-sized device. These benefits further highlight the efficiency of DPR in mitigating inference overheads during fault recovery.

8.4 Accumulated Fault Injections

To evaluate the robustness of an FPGA-based design for space applications, accumulated fault injection campaigns are commonly employed. Unlike traditional single-fault injection experiments, in which one SEU is emulated in the CRAM of the device and corrected before the new injection, the accumulation-based approach introduces a fault in the configuration memory without correction, allowing it to accumulate until a system failure is observed. This approach gives valuable information regarding the system's fault tolerance, enabling the estimation of the number of SEUs that the system can tolerate before failure. Moreover, by integrating these results with the environmental analysis, the MTTF of the RePAIR platform can be estimated, as discussed in the following section.

Due to the significantly higher number of faults required in the accumulated fault injection campaign with respect to the single-SEU experiments, the conventional bitstream corruption technique, which involves modifying and reloading the bitstream for each injection, would be extremely time-consuming. To address this limitation, we adopted an alternative strategy based on the SEM IP core provided by AMD-Xilinx. In addition to its primary role as an error correction and detection module, the SEM-IP includes built-in support for fault injection and error classification, provided the exact coordinates of the target configuration bits are known. Although AMD-Xilinx provides partial metadata such as the **Essential Bits Data (EBD)** and **Essential Bits Content (EBC)** files, which indicate the criticality of specific CRAM bits, there is no direct mapping available between FPGA resources and their corresponding configuration bits, making it challenging to generate the addresses required by the SEM-IP to perform the injections. To address this challenge, several tools have been developed to support configuration memory analysis and bitstream manipulation. Among these, we selected PyXEL [43], which enabled us to analyze the bitstream and combine EBD and EBC data to extract the coordinates of both essential and non-essential configuration bits associated with the RePAIR platform.

To enable SEM-IP-based fault injections, minor design modifications were also required. Specifically, both the DFX controller, used in the RePAIR platform for the partial reconfiguration, and the SEM-IP core require access to the CRAM via the ICAP. To manage concurrent access to this shared resource, we implemented a custom arbiter module in VHDL. This arbiter grants default ICAP control to the DFX controller and temporarily delegates access to the SEM-IP core during injection events. It is important to highlight that both the SEM-IP core and the arbiter were excluded from the actual fault injection analysis, as they serve only as validation infrastructure and are not part of the RePAIR platform under evaluation. To automate the fault injection and result classification process, a custom Python-based control platform was developed. This software communicates with the SEM-IP core via UART, issues injection commands, and collects and classifies the outcomes of the inference procedure.

Accumulated-fault injection campaigns are inherently time-intensive, since numerous random bit flips must be accumulated before manifesting a failure, and each injected fault requires the evaluation of the benchmark application to run to completion, we selected two representative workloads for the experimental campaign. The first is MNIST digit classification, a widely used benchmark in state-of-the-art studies, and the second is the Sentinel-2 application, which closely reflects realistic onboard satellite operations. Each experiment was carried out on both the KCU105

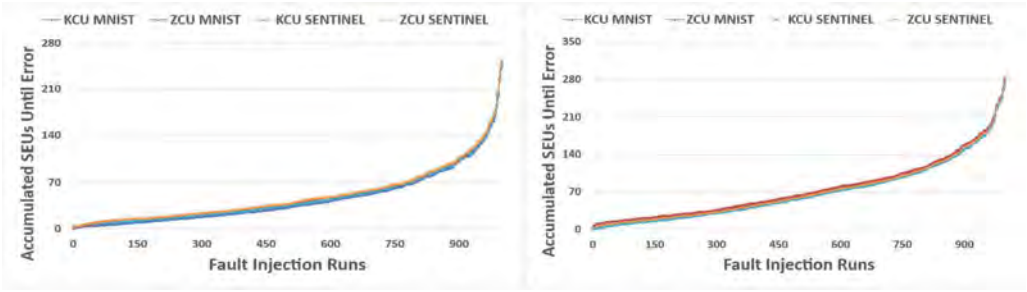


Fig. 10. TPU accumulated fault injection runs results for 1,000 runs targeting (a) essential bits; (b) random bits.

and ZCU102 platforms, enabling us to assess how the same application behaves across different devices and underlying technologies. Given the large number of critical bits involved, exceeding ten million, exhaustive testing was infeasible. Using a Monte Carlo approach and assuming a confidence level of at least 95%, each fault injection campaign was therefore limited to 1,000 runs. Figure 10 summarizes the results of the fault injection campaigns conducted on the TPU, considering two scenarios: injections targeting only the essential configuration bits (Figure 10(a)) and injections targeting configuration memory bits within the TPU area (Figure 10(b)). The results show that the TPU achieves high levels of reliability even when essential bits are selectively targeted, highlighting the robustness of the proposed approach even under worst-case conditions. As expected, when the injections are extended to all configuration bits, the overall reliability improves further with an average of around 30 more tolerated failure before failure with respect to the essential bits case. In this case, more than 550 runs required at least 50 injected faults before a system failure was observed, demonstrating the strong fault-tolerance capabilities implemented inside the TPU.

Moreover, the results show no significant differences between the outcomes obtained on the KCU105 and the ZCU102 boards. This behavior is explained by the fact that fault injection campaigns emulate only the effects of faults, without any consideration regarding the source and its nuclear interaction with the target device. Specifically, radiation-induced SEU are tightly linked to the device technology and by the energy carried by the particle. Since this two informations are omitted in fault injection, the RePAIR platform either implemented on KCU105 or ZCU102 mostly behaves the same under emulated radiation induced bitflip in CRAM. Any change, when present, may be to different resource mapping or routing that may change from one implementation and the other. As consequence, to derive realistic predictions of system reliability, it is necessary to complement the fault injection results with an environmental analysis that accounts for technology-dependent sensitivity. Such analysis is essential to quantify the benefit of adopting one technology respect to another. The details of this environmental analysis are presented in the following section.

Additionally, we further classified the observed events into the following categories:

- —TPU Halt: The Neorv32 cannot communicate with the TPU.
- —DPR Event: The injected fault triggers a DPR event in the TPU, and no error propagates to the final output. In this case, the error has been detected, corrected and the result of the inference is correct.
- —Undetected Fault: The injected fault, while producing a different inference result from the golden reference, is not detected by the implemented detection mechanism, but the RePAIR platform is still responding.

Table 6. Fault Classification

	Essential (%)				Random Bits (%)			
	MNIST		Sentinel-2		MNIST		Sentinel-2	
	KCU105	ZCU102	KCU105	ZCU102	KCU105	ZCU102	KCU105	ZCU102
TPU Halt	3.4	4.7	2.5	4.9	3.2	4.3	2.9	4.0
DPR Event	73.6	71.5	75.7	72.5	73.9	71.9	71.3	71.5
Undetected Fault	21	20.6	20	19.1	20.7	21.1	21.9	21.2
DPR Error	2	3.2	1.8	3.5	2.2	2.7	3.9	3.3

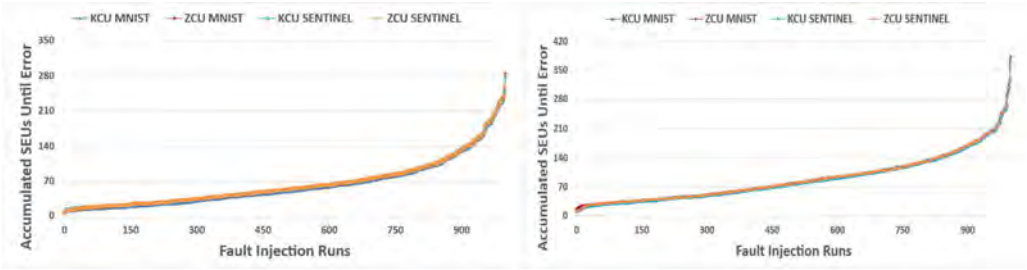


Fig. 11. RePAIR accumulated injection runs results for 1,000 runs targeting (a) essential bits; (b) random bits.

—DPR Error: Persistent error even after DPR, requiring full board reconfiguration.

Table 6 reports the outcome of this classification. Each entry in the table is expressed as a percentage of occurrence with respect to the total of 1,000 runs. As expected, most injected faults result in errors being detected by the TPU's custom fault detection mechanism. In these cases, a DPR is triggered by the processor, after which the TPU resumes operation and produces a correct inference result.

Only in a very limited number of cases does the TPU halt completely, preventing the processor from communicating with the accelerator. Such failures are most likely associated with errors occurring in the control logic or within the AXI interface connecting the NEORV32 processor to the TPU. Finally, another small fraction of cases corresponds to situations where DPR is correctly triggered but fails to restore the system to a functional state, resulting in an incorrect inference outcome. In such cases, a full board reconfiguration is triggered to restore the entire system's functionalities.

Figure 11 presents the results of the accumulated fault injection runs performed on the entire RePAIR platform, considering both cases where only the essential configuration bits were targeted and where all configuration memory bits within the platform were targeted. Similarly to what was observed in the TPU-only case, the system demonstrates higher resilience when random configuration bits are targeted. In this scenario, around 30 additional injected faults are required to cause a system failure compared to the essential-bit scenario. It is important to note, however, that even in the random-bit case, the injected faults were limited to the configuration memory region corresponding to the RePAIR implementation. In a real-case scenario, where radiation effects can impact the entire configuration memory, the platform's tolerance to such faults is expected to be even higher. However, in this study, the injection area was constrained in order to focus on validating only the performance of the RePAIR platform itself.

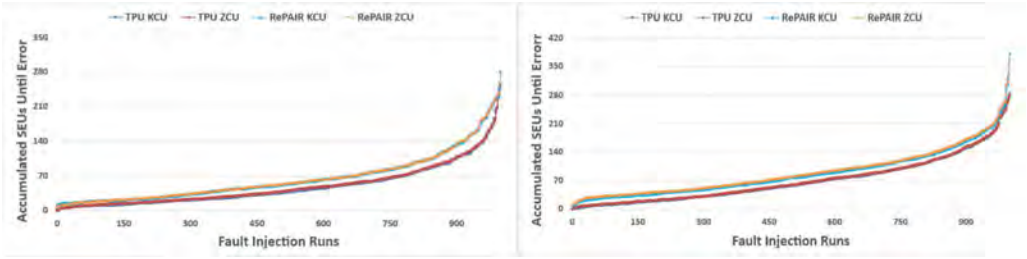


Fig. 12. Comparison between RePAIR and TPU accumulated injection runs results targeting (a) essential bits; (b) random bits.

Table 7. Heavy-Ion SEU Cross-Section Parameters as per [42]

Technology	LETth (MeV/mg/cm^2)	xSAT (cm^2/bit)	W	S
20 nm CMOS	0.8	2.0E-09	27.0	0.88
16 nm FinFET	0.3	5.4E-10	15.7	2.3

Finally, Figure 12 highlights the results of the fault injection campaigns conducted on the TPU alone with those obtained for the complete RePAIR platform while executing the Sentinel-2 benchmark. In this case, the RePAIR platform consistently outperforms the TPU-only configuration, with an average improvement of about 20 additional tolerated faults before failure in both the essential-bit and random-bit campaigns. Through this analysis, it is possible to estimate the average number of accumulated SEUs required to cause a failure in the RePAIR platform for the two devices. When injections are restricted to essential bits, the system tolerates, on average, 65 accumulated SEUs on both boards. When the fault model is extended to include all configuration memory bits within the RePAIR region, the average tolerance increases to 89 SEUs for the KCU105 and 91 SEUs for the ZCU102. These values are then combined with the environmental analysis data to estimate the MTTF of the RePAIR platform when implemented across the two different FPGA technologies.

8.5 SEU Estimation for Different Space Environments

As discussed in Section 7, it is crucial to correlate the failure rate obtained from fault-injection campaigns—where radiation-induced SEUs are emulated as CRAM bit flips—with environment-driven estimates of the actual particle flux capable of inducing SEUs during a space mission. The daily number of SEUs experienced by a satellite is strongly influenced by two primary factors: the device technology and the orbital parameters of the mission.

In this work, since the RePAIR platform was implemented on both the KCU105 (20 nm CMOS FPGA) and the ZCU102 (16 nm FinFET FPGA), we performed two complementary analyses. First, we evaluated how the underlying technology affects the MTTF of the platform when executing the same workload in a given space environment. To this end, we extracted radiation sensitivity data from [42], where both technologies were characterized under heavy-ion irradiation and the SEU cross-section per bit was fitted with a Weibull distribution, whose parameters are reported in Table 7.

Next, we modeled the space environment corresponding to the orbital parameters of Sentinel-2. Since Sentinel-2 operates in LEO, where the Earth's magnetosphere provides significant shielding, we also considered the case of EUTELSAT, a telecommunications satellite in GEO, in order to assess

Table 8. Benchmark Satellites' Trajectory Parameters

Satellite	Perigee (km)	Apogee (km)	Inclination (∘)	Orbital Period (min)	Orbit Type
SENTINEL-2	788	790	98.56	101	LEO
EUTELSAT	35,778	35,811	0.056	1,436	GEO

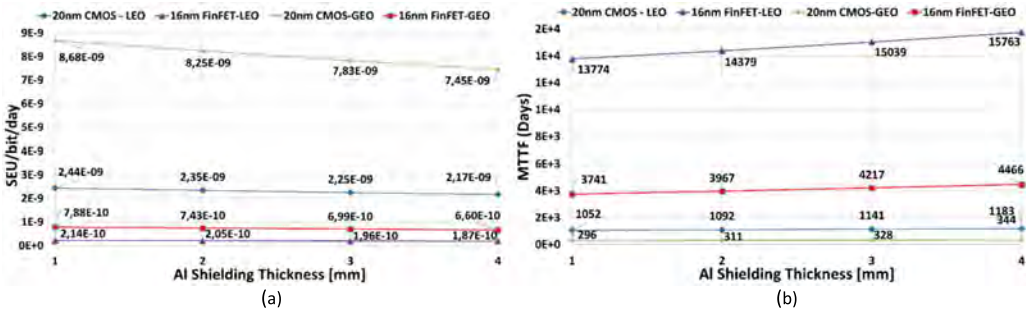


Fig. 13. (a) Estimated SEUs in configuration memory for different technology and space orbit and (b) Estimated MTTF for the RePAIR platform under different technology implementations and space orbits.

the SEU rate under GEO conditions. The orbital parameters, summarized in Table 8, were used as input to the OMERE tool [44] to simulate a one-year mission scenario for each satellite. Since the SEU cross-section is dominated by heavy-ion interactions, we focused on the GCR flux as the main particle population in orbit. The estimated flux was then combined with the cross-section data (considered separately for each technology) to run the SEE prediction engine in OMERE. Acknowledging the strong modulation of GCR flux by solar activity, we adopted the solar minimum GCR model to capture the worst-case heavy ion radiation environment, hence the worst-case heavy ion-induced SEUs affecting the platform. The estimated SEU/bit per day of mission for the two technologies is provided in Figure 13(a). The SEU rate is shown as a function of aluminum shielding thickness, which is typically used to estimate the equivalent shielding provided by satellite mechanical structure in radiation hardness assurance procedures [52].

Results show that the GEO environment leads to significantly higher SEU rates than LEO, with GEO values exceeding LEO by a factor of about 3–4 across all shielding levels. Technology also plays a major role: 16 nm FinFET consistently exhibits nearly an order of magnitude lower SEU sensitivity than 20 nm CMOS, in both LEO and GEO. The effect of increasing aluminum shielding from 1 mm to 4 mm is modest, resulting in reductions of only ∼10–15%, confirming that high-energy particles in GEO can still penetrate typical shielding thicknesses. Overall, these results highlight that while shielding provides limited mitigation, technology scaling (FinFET vs. CMOS) has a much stronger influence on SEU resilience, particularly in the harsher GEO environment. It should also be noted that this analysis only considers the contribution of heavy ions present in the near-Earth environment; the additional impact of high-energy protons, both solar and galactic in origin, is not included and could further increase the overall SEU rate.

8.6 MTTF Estimation

To estimate the MTTF of the RePAIR platform, the SEU rate per single bit of the CRAM is combined with the total number of CRAM bits associated with the RePAIR implementation on the two target devices, KCU105 and ZCU102. The RePAIR platform requires 34,677,324 bits on the KCU105 and

30,871,616 bits on the ZCU102 to be implemented. By multiplying the per-cell SEU rate by these totals, it is possible to estimate the number of SEUs that would affect the entire platform over the course of one day in each considered space environment.

Once the SEU rate for the platform is derived, it is combined with the average number of accumulated SEUs before system failure (as described in the previous section). This combination enables the estimation of the platform's MTTF due to heavy-ion induced upsets in both GEO and LEO environments. The results are summarized in Figure 13(b). According to the MTTF estimation, the implementation in 16 nm FinFET technology operating in LEO exhibits the highest MTTF. However, it must be stressed that this result only accounts for SEUs in CRAM caused by heavy-ion interactions. In LEO, the heavy-ion contribution is strongly mitigated by the Earth's magnetic field, and FinFET devices inherently show reduced sensitivity to heavy ions compared to planar CMOS, leading to an artificially high MTTF estimate.

In reality, the radiation environment in LEO is dominated by trapped protons and electrons, whose fluxes are orders of magnitude higher than those of heavy ions. These particles are the main drivers of failures in LEO but vanish in GEO. Since SEU cross-section data for protons is not available for the considered devices, their contribution could not be included in this analysis. This means that the effect of protons, whether originating from GCRs or from solar events during periods of high activity, is missing. Protons primarily induce SEUs through indirect ionization, and because their flux is much higher than that of heavy ions across most space orbits, their overall impact on reliability is considerably greater. As a result, including proton-induced upsets for both devices and environments would significantly lower the MTTF values reported here.

Therefore, while the presented MTTF values provide a useful first-order estimate of the heavy-ion contribution, they should be interpreted as an optimistic upper bound. A comprehensive evaluation of platform reliability requires the inclusion of proton-induced upsets, and ultimately, a direct exposure to ionizing particles through a radiation test campaign.

9 Conclusions

This research introduces RePAIR, a highly reliable and high-performance FPGA-based platform specifically designed for DNN execution in safety-critical applications. The platform integrates the NEORV32 RISC-V core, enhanced with TMR to guarantee fault tolerance, and TinyTPU, a SA-based accelerator extended with custom ISA instructions that enable runtime fault detection during inference. RePAIR goes beyond simple fault detection by also supporting fault correction through DPR of the DNN accelerator, a process orchestrated by the RISC-V core. To validate its effectiveness, the platform was implemented on two FPGA boards fabricated with different process technologies, 16 nm FinFET and 28 nm CMOS, and evaluated under hardware fault emulation. Fault injection was performed directly on the configuration memory, introducing both single-bit upsets and accumulated multi-bit errors to reproduce realistic failure scenarios expected when deployed in space. Experimental results demonstrated a 94% runtime fault detection accuracy across the benchmark application, with a worst-case computational overhead of only 0.6%. Furthermore, leveraging DPR to correct corrupted modules drastically reduced recovery time, achieving nearly a 900× reduction in system downtime on medium-to-large-scale FPGA devices. Finally, the platform's MTTF was estimated by combining the characterized radiation environments in GEO and LEO with the measured sensitivity of the two technologies. Results show that RePAIR exhibits remarkable fault tolerance, with reliability benefits arising from the combined effect of SEU detection during inference and DPR-based recovery. This synergy ensures that the platform maintains operational continuity even under harsh radiation conditions, confirming its suitability for safety-critical space deployments.

References

[1] Andrew Waterman and Krste Asanovic. 2017. *The RISC-V Instruction Set Manual, Volume I: User-Level ISA, Document Version 2.2*. RISC-V Foundation.

[2] D.-Z. Li, H.-R. Gong, and Y.-C. Chang. 2018. Implementing RISCV system-on-chip for acceleration of convolution operation and activation function based on FPGA. In *Proceedings of the 2018 14th IEEE International Conference on Solid-State and Integrated Circuit Technology (ICSICT)*, 1–3. DOI: https://doi.org/10.1109/ICSICT.2018.8564810

[3] Héctor Martínez, Sandra Catalán, Adrián Castelló, and Enrique S. Quintana-Ortí. 2024. Parallel GEMM-based convolutions for deep learning on multicore ARM and RISC-V architectures. *Journal of Systems Architecture* 153 (2024), 103186. DOI: https://doi.org/j.sysarc.2024.103186

[4] Xingbo Wang, Chenxi Feng, Bingzhen Chen, Qi Wang, Yucong Huang, and Terry Tao Ye. 2024. RV-GEMM: Neural network inference acceleration with near-memory GEMM instructions on RISC-V. In *Proceedings of the 21st ACM International Conference on Computing Frontiers (CF '24)*, 302–305. DOI: https://doi.org/10.1145/3649153.3649181

[5] N. P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. 2017. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, 1–12.

[6] Fabiano Libano, Paolo Rech, and John Brunhaver. 2023. Efficient error detection for matrix multiplication with systolic arrays on FPGAs. *IEEE Transactions on Computers* 72, 8 (2023), 2390–2403.

[7] S. Burel, Adrian Evans, and Lorena Anghel. 2021. MOZART: Masking outputs with zeros for architectural robustness and testing of DNN accelerators. In *Proceedings of the 2021 IEEE 27th International Symposium on On-Line Testing and Robust System Design (IOLTS)*, 1–6.

[8] E. Vacca, Sarah Azimi, and Luca Sterpone. 2024. ZOR: Zero overhead reliability strategies for AI accelerators. In *Proceedings of the 2024 22nd IEEE Interregional NEWCAS Conference (NEWCAS)*, 248–252. DOI: https://doi.org/10.1109/NewCAS58973.2024.10666350

[9] P. E. Dodd, M. R. Shaneyfelt, J. R. Schwank, and J. A. Felix. 2010. Current and future challenges in radiation effects on CMOS electronics. *IEEE Transactions on Nuclear Science* 57, 4 (2010), 1747–1763.

[10] R. F. Hodson, D. Morgan, R. L. Ladbury, Y. Chen, M. Bay, and J. Zinchuk. 2019. *Radiation Single Event Effects (SEE) Impact on Complex Avionics Architecture Reliability, NASA/TM-2019-220269 (NESC-RP-17-01211)*. NASA.

[11] AMD. 2018. Soft Error Mitigation Controller v4.1 (PG036). AMD.

[12] G. Cora, E. Vacca, C. De Sio, S. Azimi, and L. Sterpone. 2025. RePAIR: Reconfigurable platform for AI Resilience within RISC-V ecosystem. In *Proceedings of the Applied Reconfigurable Computing. Architectures, Tools, and Applications (ARC)*. R. Giorgi, M. Stojilović, D. Stroobandt, P. Brox Jiménez, and Á. Barriga Barros (Eds.), Lecture Notes in Computer Science, Vol 15594. Springer. DOI: https://doi.org/10.1007/978-3-031-87995-1_5

[13] S. Nolting and Contributors. 2023. The NEORV32 RISC-V processor. *Zenodo*. DOI: https://doi.org/10.5281/zenodo.8260609

[14] Jonas Fuhrmann. 2018. Implementierung einer Tensor Processing Unit mit dem Fokus auf Embedded Systems und das Internet of Things. Retrieved from http://hdl.handle.net/20.500.12738/8527

[15] Yonghua Zhang, Hongxu Jiang, Xiaojian Liu, Haiheng Cao, and Yu Du. 2022. High-efficient MPSoC-based CNNs accelerator with optimized storage and dataflow. *The Journal of Supercomputing* 78, 3 (2022), 3205–3225. DOI: https://doi.org/10.1007/s11227-021-03909-y

[16] Kaiyuan Guo, Lingzhi Sui, Jiantao Qiu, Jincheng Yu, Junbin Wang, Song Yao, Song Han, Yu Wang, and Huazhong Yang. 2018. Angel-Eye: A complete design flow for mapping CNN onto embedded FPGA. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 37, 1 (2018), 35–47. DOI: https://doi.org/10.1109/TCAD.2017.2705069

[17] A. Sanchez-Flores, L. Alvarez, and B. Alorda-Ladaria. 2023. Accelerators in embedded systems for machine learning: A RISCV view. In *Proceedings of the 2023 38th Conference on Design of Circuits and Integrated Systems (DCIS)*, 1–6. DOI: https://doi.org/10.1109/DCIS58620.2023.10335969

[18] E. Parisi, Alberto Musa, Simone Manoni, Maicol Ciani, Davide Rossi, and Francesco Barchi. 2024. TitanCFI: Toward enforcing control-flow integrity in the root-of- trust. In *Proceedings of the 2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 1–6. DOI: https://doi.org/10.23919/DATE58400.2024.10546873

[19] P. R. Nikiema, Alessandro Palumbo, Allan Aasma, Luca Cassano, Angeliki Kritikakou, and Ari Kulmala. 2023. Towards dependable RISC-V cores for edge computing devices. In *Proceedings of the 2023 IEEE 29th International Symposium on On-Line Testing and Robust System Design (IOLTS)*, 1–7. DOI: https://doi.org/10.1109/IOLTS59296.2023.10224862

[20] C. Gewehr, L. Luza, and F. G. Moraes. 2024. Hardware acceleration of crystals-Kyber in low-complexity embedded systems with RISC-V instruction set extensions. *IEEE Access* 12 (2024), 94477–94495. DOI: https://doi.org/10.1109/ACCESS.2024.3416812

[21] Z. Wang, Haochen Du, Baoyi Han, Yanqing Xu, Xiaonan Tang, and Yang Zhou. 2024. RTPE: A high energy efficiency inference processor with RISC-V based transformation mechanism. In *Proceedings of the 2024 IEEE 6th International Conference on AI Circuits and Systems (AICAS)*, 297–301. DOI: https://doi.org/10.1109/AICAS59952.2024.10595923

[22] J. Wei, L. Zhang, Z. Yu, and D. Liu. 2020. Design space exploration for heterogenous SoC integrated with matrix accelerator. In *Proceedings of the 2020 IEEE 2nd International Conference on Circuits and Systems (ICCS)*, 40–43.

[23] G. Pagonis, V. Leon, D. Soudris, and G. Lentaris. 2023. Increasing the fault tolerance of COTS FPGAs in Space: SEU Mitigation Techniques on MPSoC. In *Proceedings of the Applied Reconfigurable Computing. Architectures, Tools, and Applications(ARC)*. F. Palumbo, G. Keramidas, N. Voros, and P. C. Diniz (Eds.), Lecture Notes in Computer Science, Vol. 14251. Springer. DOI: https://doi.org/10.1007/978-3-031-42921-7_15

[24] Jorge Cano-Páez, Luis Entrena, Mario García-Valderas, and Almudena Lindoso. 2025. Architecture for error detection and recovery in MPSoCs: A hypervisor approach using dynamic partial reconfiguration. *IEEE Transactions on Nuclear Science* 72 (2025), 2636–2644. DOI: https://doi.org/10.1109/TNS.2025.3534431

[25] A. Pérez, Alfonso Rodríguez, Andrés Otero, David González Arjona, Álvaro Jiménez-Peralo, and Miguel Ángel Verdugo. 2020. Run-time reconfigurable MPSoC-based on-board processor for vision-based space navigation. *IEEE Access* 8 (2020), 59891–59905. DOI: https://doi.org/10.1109/ACCESS.2020.2983308

[26] Zhen Gao, Lini Yuan, Jingyan Wang, Qiang Liu, Javier Conde, Pedro Reviriego, Shulin Zeng, Yu Wang, Shanshan Liu, Fabrizio Lombardi, et al. 2025. Robustness against faults in configuration memories of FPGA-based LLMs. *IEEE Transactions on Circuits and Systems for Artificial Intelligence* 2 (2025), 162–173. DOI: https://doi.org/10.1109/TCASAI.2025.3552735

[27] Kuang-Hua Huang and J. A. Abraham. 1984. Algorithm-based fault tolerance for matrix operations. *IEEE Transactions on Computers* C-33, 6 (1984), 518–528. DOI: https://doi.org/10.1109/TC.1984.1676475

[28] J. Kim, H. Lee, J. Park, and S. Kang. 2022. ZOS: Zero overhead scan for systolic array-based AI accelerator. In *Proceedings of the 2022 19th International SoC Design Conference (ISOCC)*, 360–361. DOI: https://doi.org/10.1109/ISOCC56007.2022.10031441

[29] S. Zhan, Dmitriy Bekker, Jeffrey Boye, Laurel Funk, Matthew Gile, and Michael Hoffmann. 2021. The design and verification of the DART single board computer FPGA. In *Proceedings of the 2021 IEEE Aerospace Conference*, 1–11. DOI: https://doi.org/10.1109/AERO50100.2021.9438523

[30] D. M. Marcos and A. Valverde Carretero. 2023. DHS architecture for HERA deep space mission. In *Proceedings of the 2023 European Data Handling & Data Processing Conference (EDHPC)*, 1–6. DOI: https://doi.org/10.23919/EDHPC59100.2023.10396073

[31] E. Vacca, S. Azimi, and L. Sterpone. 2022. Failure rate analysis of radiation tolerant design techniques on SRAM-based FPGAs. *Microelectronics Reliability* 138 (2022), 114778. DOI: https://doi.org/j.microrel.2022.114778

[32] Adria B. de Oliveira, Lucas A. Tambara, Fabio Benevenuti, Luis A. C. Benites, Nemitala Added, Vitor A. P. Aguiar, Nilberto H. Medina, Marcilei A. G. Silveira, and Fernanda L. Kastensmidt. 2020. Evaluating soft core RISC-V processor in SRAM-Based FPGA under radiation effects. *IEEE Transactions on Nuclear Science* 67, 7 (2020), 1503–1510. DOI: https://doi.org/10.1109/TNS.2020.2995729

[33] S. Azimi, C. De Sio, A. Portaluri, D. Rizzieri, and L. Sterpone. 2022. A comparative radiation analysis of reconfigurable memory technologies: FinFET versus bulk CMOS. *Microelectronics Reliability* 138 (2022), 114733. DOI: https://doi.org/j.microrel.2022.114733

[34] D. I. Moldovan. 1983. On the design of algorithms for VLSI systolic arrays. *Proceedings of the IEEE* 71, 1 (1983), 113–120. DOI: https://doi.org/10.1109/PROC.1983.12532

[35] S.-J. Jou and C.-W. Jen. 1988. The design of a systolic array system for linear state equations. In *Proceedings of the International Conference on Systolic Arrays*, 275–284. DOI: https://doi.org/10.1109/ARRAYS.1988.18068

[36] Sau-Gee Chen, Jiann-Cherng Lee, and Chieh-Chih Li. 1994. Systolic implementation of Kalman filter. In *Proceedings of the 1994 Asia Pacific Conference on Circuits and Systems* (*APCCAS '94*), 97–102. DOI: https://doi.org/10.1109/APCCAS.1994.514531

[37] Biagio Peccerillo, Mirco Mannino, Andrea Mondelli, and Sandro Bartolini. 2022. A survey on hardware accelerators: Taxonomy, trends, challenges, and perspectives. *Journal of Systems Architecture* 129 (2022), 102561. DOI: https://doi.org/10.1016/j.sysarc.2022.102561

[38] Mehdi Safarpour, Reza Inanlou, and Olli Silven. 2022. Algorithm level error detection in low voltage systolic array. *IEEE Transactions on Circuits and Systems II: Express Briefs* 69, 2 (2022), 569–573.

[39] Hayoung Lee, Jihye Kim, Jongho Park, and Sungho Kang. 2023. STRAIT: Self-test and self-recovery for AI accelerator. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 42 (2023), 3092–3104.

[40] AMD. 2025. *Vivado Design Suite User Guide: Dynamic Function eXchange (UG909)*. AMD.

[41] AMD. 2020. *Dynamic Function eXchange Controller v1.0 Product Guide (PG374)*. AMD.

[42] D. S. Lee, Michael King, William Evans, Matthew Cannon, Andrés Pérez-Celis, and Jordan Anderson. 2018. Single-event characterization of 16 nm FinFET xilinx UltraScale+ devices with heavy ion and neutron irradiation. In *Proceedings of the 2018 IEEE Radiation Effects Data Workshop (REDW)*, 1–8. DOI: https://doi.org/10.1109/NSREC.2018.8584313

[43] C. De Sio, S. Azimi, L. Sterpone, D. M. Codinachs, and F. Decuzzi. 2023. PyXEL: Exploring bitstream analysis to assess and enhance the robustness of designs on FPGAs. In *Proceedings of the 2023 19th International Conference*

on Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design (SMACD), 1–4. DOI: https://doi.org/10.1109/SMACD58065.2023.10192116

[44] A. Varotsou, Nathalie Chatry, Pierre-François Peyrard, Françoise Bezerra, Anne Samaras, and Eric Lorfèvre. 2011. Shielding geometry effect on SEE prediction using the new OMERE release: JASON-2 mission case study. In *Proceedings of the 2011 12th European Conference on Radiation and Its Effects on Components and Systems*, 849–853. DOI: https://doi.org/10.1109/RADECS.2011.6131315

[45] M. Kruglanski, Neophytos Messios, Erwin De Donder, Emmanuel Gamby, Stijn Calders, and Laszlo Hetey. 2009. Last upgrades and development of the space environment information system (SPENVIS). In *Proceedings of the 2009 European Conference on Radiation and Its Effects on Components and Systems*, 563–565. DOI: https://doi.org/10.1109/RADECS.2009.5994715

[46] E. Vacca, Corrado De Sio, Sarah Azimi, and Luca Sterpone. 2024. On assessing the robustness of RISC-V soft cores for space systems by mission-tailored SEU analysis. In *Proceedings of the 2024 31st IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, 1–4. DOI: https://doi.org/10.1109/ICECS61496.2024.10848907

[47] P. Helber, B. Bischke, A. Dengel, and D. Borth. 2019. EuroSAT: A novel dataset and deep learning benchmark for land use and land cover classification. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing* 12, 7 (2019), 2217–2226. DOI: https://doi.org/10.1109/JSTARS.2019.2918242

[48] K. Kotowsk, Christoph Haskamp, Jacek Andrzejewski, Bogdan Ruszczak, Jakub Nalepa, Daniel Lakey, Peter Collins, Aybike Kolmas, Mauro Bartesaghi, Jose Martinez-Heras, et al. 2024. European space agency benchmark for anomaly detection in satellite telemetry. arXiv:2406.17826. Retrieved from https://arxiv.org/abs/2406.17826

[49] F. Buccellato, D. Nicolini, E. Vacca, C. De Sio, and L. Sterpone. 2025. AI-powered anomaly detection for satellite telemetry. In *Proceedings of the 22nd ACM International Conference on Computing Frontiers (CF '25)*. ACM, New York, NY, 222–223. DOI: https://doi.org/10.1145/3719276.3727953

[50] O. Chatzopoulos, M. Trakosa, H. D. Dixit, S. Sankar, and D. Gizopoulos. 2026. Measuring the unmeasurable: Demystifying silent data corruptions in AI accelerators through microarchitectural modeling. *IEEE Micro* 46 (2026), 55–65. DOI: https://doi.org/10.1109/MM.2025.3646859

[51] O. Chatzopoulos, M. Trakosa, and D. Gizopoulos. 2025. Accurate analysis of silent data corruptions in programmable AI accelerator microarchitectures. In *Proceedings of the 2025 IEEE 31st International Symposium on On-Line Testing and Robust System Design (IOLTS)*, 1–4. DOI: https://doi.org/10.1109/IOLTS65288.2025.11117132

[52] K. Lemière, R. Benacquista, A. Varotsou, J. Guillermin, M. Ruffenach, and M. Rizzo. 2025. Investigation on the limits of the ray-tracing method applied on dose analysis for radiation hardness assurance. *IEEE Transactions on Nuclear Science* 72 (2025), 2480–2489. DOI: https://doi.org/10.1109/TNS.2025.3540078

Received 28 August 2025; revised 14 January 2026; accepted 22 March 2026