POLITECNICO DI TORINO
Repository ISTITUZIONALE

On-Hardware Resilience Analysis of DPU-Accelerated CNNs on FPGA-Based Systems

Original
On-Hardware Resilience Analysis of DPU-Accelerated CNNs on FPGA-Based Systems / Buccellato, F., De Sio, C., Azimi, S., Sterpone, L.. - ELETTRONICO. - (2025), pp. 34-41. (28th Euromicro Conference Series on Digital System Design Salerno (ITA) September 10th-12th, 2025) [10.1109/DSD67783.2025.00017].

Availability:
This version is available at: 11583/3002680 since: 2026-03-04T15:49:54Z

Publisher:
IEEE

Published
DOI:10.1109/DSD67783.2025.00017

Terms of use:

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

Publisher copyright
IEEE postprint/Author's Accepted Manuscript

©2025 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collecting works, for resale or lists, or reuse of any copyrighted component of this work in other works.

(Article begins on next page)

13 September 2026

On-Hardware Resilience Analysis of DPU-Accelerated CNNs on FPGA-Based Systems

Federico Buccellato, Corrado De Sio, Sarah Azimi, Luca Sterpone
Dipartimento di Automatica e Informatica
Politecnico di Torino
Turin Italy
{name.surname}@polito.it

***Abstract*— In recent years, Reconfigurable SoCs have emerged as a high-performance solution for embedded systems, addressing the increasing complexity of neural networks, balancing performance, cost, and adaptability. Flexible hardware accelerators, such as AMD's Deep Learning Processing Units (DPUs), enable efficient computation across various domains, including safety-critical applications. However, soft errors remain a significant reliability concern, especially in harsh environments like space, where radiation-induced corruption of configuration memory poses a significant threat to FPGA-based systems. Most research on the reliability and robustness of deep learning models against soft errors has focused on application-level analyses, with comparatively little attention paid to architectural hardware faults. This paper introduces a resilience evaluation framework targeting AMD's state-of-the-art DPU, comparing traditional application-level fault injection with hardware-aware fault injection performed on an actual hardware platform, a Kria KV260. Applying this methodology, we evaluated fourteen different deep neural network architectures and demonstrated that hardware-aware fault injections reveal critical vulnerabilities that application-only approaches fail to detect. Moreover, we investigated the source of different faults at the hardware level, enabling the identification of architectural resources that are more susceptible to errors. These insights are valuable to support the development of more robust deployment strategies and mitigation techniques tailored to FPGA-based deep learning accelerators.**

Keywords—Fault injection, deep neural network, DPU, FPGA, reliability

I. INTRODUCTION

Neural networks have become a primary solution for solving complex prediction and classification problems, and they are now widely adopted in fields such as robotic control [1], automotive systems [2], object recognition [3][4], natural language processing [5][6], and aerospace applications [7]. As a result of this trend, researchers and industry have generated a steady stream of new architectures and solutions, each tailored to specific requirements. However, the growing computational demand of modern models, such as deep convolutional neural networks, has introduced significant challenges, especially when targeting embedded applications, where conventional system-on-chip (SoC) designs are often inadequate for these workloads.

Reconfigurable SoCs have emerged as a high-performance solution by combining general-purpose processors with programmable logic, offering a flexible and efficient platform for deploying neural networks and tailoring hardware architecture to specific computational patterns.

The Deep-Learning Processing Unit (DPU) is a specialized hardware accelerator provided by AMD, designed to be deployed in FPGAs and Reconfigurable SoCs as an engine for high-performance inference tasks [8]. Thanks to the possibility of supporting DPU with software control and custom hardware, these platforms have attracted and enabled the integration of neural networks in safety-critical domains, such as automotive systems and space missions [9][10]. However, as neural networks become increasingly prevalent in these critical systems, a comprehensive and systematic study of their reliability and resilience becomes essential.

Soft errors are one of the main concerns for SRAM-based reconfigurable systems, as they can silently corrupt configuration memory defining the hardware architecture. These faults can be induced by radiation or disturbances and compromise the functionality of the systems. Therefore, understanding the effect of such faults in neural networks and assessing their resilience to soft errors is a crucial step toward their deployment in mission-critical applications.

Different solutions and approaches exist for evaluating the reliability of neural networks and inference platforms. When considering radiation-induced soft errors, which are one of the main sources of failures for reconfigurable systems operating in space, radiation testing is a realistic evaluation method that exposes the device to radiation, allowing evaluation of the reliability of the system against the effects caused by interaction with a device and radiation [11]. Although these tests accurately reproduce the real failure mechanisms induced by particles, they are significantly expensive in terms of both cost and infrastructure. Another way to assess the reliability of neural-network-based applications relies on simulation-level methodologies, where faults are injected during RTL simulation to observe their effect on the system. However, such a solution presents multiple limitations. Access to the RTL description is often unavailable, and the high simulation time usually makes it unfeasible to perform a comprehensive and systematic analysis of the target application, a problem that is exacerbated by the growing complexity of models and accelerators. For these reasons, most current analyses rely on studies on application-level fault injection. In application-level fault injections, disturbances are injected in the model describing the neural network architecture, affecting parameters such as weights or intermediate states between layers to measure their impact on inference accuracy [12][13]. However, these analyses do not involve the computational platform, neglecting the system's actual response to hardware faults. While application-level fault injection may offer some insights into the intrinsic robustness of neural network models, the interaction between the model and the underlying hardware architecture could hide more

XXX-X-XXXX-XXXX-X/XX/$XX.00 ©20XX IEEE

complex details. As a result, the conclusions drawn may not reflect the real-world behaviour of Deep-Neural Network (DNN) deployed on specialized hardware such as a DPU. Still, there is currently a strong reliance on application-level analyses, primarily because they are the easiest to carry out, even though these methods may not provide a reliable assessment.

The main contribution of this research is to highlight and bridge this gap. The paper focuses on evaluating how traditional application-level software reliability analyses compare to hardware-level evaluations of neural networks. Such results are achieved by exploiting Reconfigurable SoCs. Reconfigurable SoCs serve a dual purpose. First, they are a state-of-the-art platform for efficiently deploying neural networks, thanks to integrating general-purpose processors with programmable logic and specialized accelerators such as DPUs. Second, reconfigurable SoCs provide a controllable and observable environment for studying the behaviour of neural networks running on hardware under fault conditions. Their intrinsic configurability allows for direct manipulation of architectural components and facilitates the emulation of faults without the need for destructive testing or expensive radiation campaigns. This approach enables us to observe how faults propagate through the hardware and software stack, affecting the inference accuracy. By directly comparing the results of application-level and on-hardware fault injection campaigns on fourteen different CNN architectures, we assess the limitations of abstract-level analyses and demonstrate the added value of hardware-aware methodologies. Our results reveal critical differences in fault manifestation and model vulnerability, emphasizing the importance of platform-specific evaluation, particularly in the context of safety-critical applications, where overlooking hardware-induced behaviours leads to underestimated risks. Additionally, we highlight how hardware-induced behaviours and fault manifestation differ significantly from software-level assumptions. Ultimately, this work aims to provide a deeper understanding of fault effects in DPU-accelerated neural networks.

The rest of the paper is organized as follows. Section II reviews related work on fault-tolerance evaluation of deep-learning models on hardware accelerators. Section III describes the technical background. Section IV details our fault-injection methodology. Section V reports the experimental results. Finally, Section VI concludes the paper and outlines directions for future work.

II. Related Works

In recent years, neural networks have been widely adopted in many applications running on different hardware platforms across diverse fields, including safety-critical systems. This broad deployment has underscored the need for reliability studies, especially in contexts where errors could have catastrophic consequences. Numerous studies have been dedicated to analysing reliability in this setting, many employing application-level fault injection to keep the analysis as much as possible independent of the hardware platform, in order to obtain generalized results on the impact of faults on inference accuracy. The authors of [12] presented Ares, a framework for empirical analysis based on a software-based approach working at the algorithmic level, thus providing an initial evaluation layer that can guide more in-depth microarchitectural investigations involving the hardware layer. Similarly, the authors of [13] use the Darknet infrastructure to simulate soft errors in number representation independently of the hardware layer and evaluate their impact on inference accuracy.

However, purely application-based analyses seem only partially to capture the dynamics of the underlying hardware substrate. A few works tried to move the analysis to a lower abstraction level for simple network models and hardware accelerators. Authors in [14] assess that application-level studies may overlook key hardware behaviours, limiting the precision of their results, and propose a pipeline-style, multi-level environment that accelerates RTL simulations of DNNs and enables more realistic fault injections if the hardware description is available. In [15], the authors investigated the reliability of custom HLS implementations of convolutional layers on Reconfigurable SoC platforms by emulating hardware faults, yielding microarchitectural insights not accessible through purely software methods. Likewise, in [16], fault-injection experiments were carried out to assess the resilience of binary networks under single- and multi-bit upsets, revealing distinct architectural responses. Finally, in [17], the combination of fault injection and neutron irradiation on a multilayer perceptron deployed on FPGAs evaluated the reliability of this network at the full-model level and at the level of individual layers.

However, neural network models have grown larger and more computationally demanding in recent years. Specialized accelerators have been developed to enable their efficient execution on both common devices and reconfigurable platforms. These devices, such as Tensor Processor Units (TPUs), GPUs, and AMD's DPUs for FPGA-based systems, are designed to optimize neural network inference to meet the performance and energy constraints of real-world implementations. With the introduction of these specialized accelerators, research into their reliability has also emerged to understand how they behave when subjected to faults [18]. For example, in [19], researchers investigated the susceptibility of NVIDIA Kepler, Maxwell, and Pascal GPUs to bit-flip errors during YOLO, Faster R-CNN, and ResNet executions, uncovering the role of streaming-multiprocessor topology in error rates. Another study examining accelerator behaviour [20] uses a hierarchical fault-injection approach to assess single-event effects (SEEs) on deep-learning models accelerated by various FPGA DPUs, analyzing performance trends, inference accuracy, and in-orbit fault rates.

Despite these efforts, relatively few studies offer an extensive evaluation of how faults affect specific state-of-the-art accelerator designs for reconfigurable systems, such as AMD's DPU. Moreover, the extent to which application-level fault injection can serve as a reliable proxy for neural network models deployed on real-world hardware remains unclear. To fill this gap, the present work proposes a comprehensive analysis of fourteen neural networks implemented on AMD's DPU, the state-of-the-art accelerator for reconfigurable systems developed by AMD. This work presents a methodology for evaluating the reliability of such systems, combining traditional application-level fault injection with a hardware-level investigation. The comparison enables a deeper assessment of accelerator resilience and provides insight into the validity and

limitations of application-level analyses in capturing real system behaviour.

III. Technical Background

In this section, we provide a technical background by first describing the Deep Learning Processing Unit [8] and then the two key frameworks used in our analysis, PyXEL [21] and Vitis AI [22]. This background provides base information for understanding in detail the characteristics of the accelerator used, explaining the tools that enabled us to deploy neural networks on the DPU, and conducting a resilience analysis of the accelerator.

A. Deep-Learning Processing Unit

AMD's Deep Learning Processing Unit is a specialized hardware accelerator offered as a pre-implemented obfuscated overlay for FPGA and SoC platforms. It is the standard *de facto* general-purpose solution to accelerate neural network models for inference on AMD reconfigurable systems. Although its detailed hardware architecture remains confidential, the DPU is fully configurable to meet varying performance and resource requirements and exposes a parallel computation paradigm via dedicated pipelines for convolution, pooling, activation, and matrix multiplication. An integrated instruction engine orchestrates on-chip data movement and execution scheduling without CPU intervention, leveraging an optimized memory hierarchy that buffers intermediate results on-chip while storing model weights and bulk data in external DDR. High-speed interconnects minimize external memory transfers and latency, and hardware-level parallelism maximizes throughput for neural network inference. By abstracting away, the hardware design and simplifying deployment, the DPU delivers high-performance, low-latency inference on reconfigurable devices.

B. PyXEL

PyXEL [21] is a tool that enables a detailed fault-injection process and analysis on reconfigurable systems, able to analyze how errors affecting the configuration memory affect the implemented netlist. Specifically, PyXEL works directly with the FPGA's Configuration RAM (CRAM), which stores the device's configuration bits responsible for defining the netlist of the circuit implemented on the FPGA resources, such as logic functions of Look-Up Tables (LUT), interfaces with memory and DSPs, and interconnections. Modifying or corrupting these bits, emulating radiation-induced Single-Event Upsets, can reproduce functional faulty behaviour caused by radiation-induced faults in reconfigurable systems. PyXEL not only provides precise control over fault injection targets, enabling realistic and fine-grained emulation of errors within the CRAM and evaluation of their impact on hardware functionality, but it also allows detailed analysis of the resources affected and logic cells associated with the netlist, helping users identify critical vulnerabilities within the implemented design.

C. Vitis AI

Vitis AI is the software stack developed by AMD for optimizing and accelerating neural networks on adaptive SoCs, FPGAs, and Alveo accelerator cards. This tool allows users to import models from frameworks such as TensorFlow, PyTorch, and Caffe and perform inference on programmable devices through the Deep Learning Processing Unit. To achieve this, Vitis AI employs a dedicated quantizer and compiler: the quantizer applies per-channel INT8 quantization and calibration to maintain accuracy, while the compiler directly maps supported operations such as convolution, pooling, and softmax into the DPU hardware cores. Any unsupported operations are offloaded to the ARM host or implemented through HLS-synthesized modules. By abstracting hardware-software co-design, Vitis AI significantly reduces deployment time, simplifying the integration and usage of neural networks on the reprogrammable devices. The combination of Vitis AI software stack and DPU hardware acceleration currently represents the standard de facto for running neural-network-based solutions on AMD reconfigurable devices.

IV. Methodology

In this section, we illustrate the methodology developed to evaluate the reliability of neural networks at the hardware and application levels against specific fault models. We detail and describe the fault models, discussing the associated fault mechanisms. The flow of producing the neural network models to deploy on hardware for the fault campaigns is discussed as well. Finally, after presenting our hardware-level methodology, we report the application-level analysis proposed to highlight the differences between the two evaluation approaches.

A. Fault Models: Soft Errors and Permanent Faults

When evaluating the robustness of systems, the fault models play a central role. This work focuses on soft errors and permanent errors. Permanent errors are also referred to as hardware faults. Soft errors are faults caused by different sources based on the specific domain considered. The most common source of soft errors in space is space radiation. In particular, a Single Event Upset (SEU) results from energetic particles striking sensitive nodes in the device, which cause a bit flip in a memory cell. These errors are soft in the sense that the functionality of the memory element is not compromised, and the memory cell can be written with a new value and work correctly. Differently, a hardware or permanent fault often arises from physical defects, such as conditions leading to stuck-at-0 or stuck-at-1 in logic cells or interconnects, due to issues such as manufacturing imperfections, aging, or total ionizing dose. However, in FPGA-based reconfigurable systems, these two concepts partially overlap. The configuration memory itself defines the device's functional architecture, so a soft error, like a bit flip in configuration frames of CRAM, produces a persistent alteration of the implemented circuit until the bit is rewritten. Since CRAM is usually written at boot or to change the system functionality, the boundary between transient and permanent faults is less marked in FPGA platforms. Due to this characteristic, we found reconfigurable hardware systems particularly well suited for evaluating the effect that permanent hardware faults via soft error injection can produce. Indeed, by introducing SEUs in configuration bits, one can emulate a wide variety of hardware defects in a fully controlled manner. Furthermore, CRAM soft errors are the dominant source of malfunctions in FPGA-based accelerators, occurring orders of magnitude more frequently than physical hardware failures, because SEUs induced by radiation or electrical disturbance are far more common in

typical operating environments. Emulating such faults, therefore, serves the dual purpose of assessing both hardware-level defect scenarios and the principal error mechanism affecting FPGA systems.

However, since many adopted approaches for assessing the robustness of neural network models focus on soft errors [23] at higher levels of abstraction, considering only the neural network model structure, we also included in our work an analysis methodology targeting this fault model and a methodology to enable a direct comparison. In particular, unlike faults that affect the hardware structure, these abstracted soft errors impact internal data processed by the system, such as neural network weights, biases, or activations. This class of faults is frequently used in resilience studies due to the ease of injection in software-level frameworks and the ability to emulate the effects of transient disturbances without requiring access to low-level hardware internals.

B. Fault Emulation Methodology Overview

The steps of the proposed fault emulation methodology are illustrated in Figure 1. The first step of the methodology is dedicated to analyzing the accelerator's configuration data. The purpose is to identify the used resources in the programmable hardware part of the reconfigurable SoC to perform a fault injection targeting the used resources. Since the DPU is an obfuscated IP, neither the netlist nor the hardware implementation details within the programmable logic are available. To solve these issues, we relied on PyXEL to explore the configuration data structure and extract information on the configuration resources used for the overlay, thereby precisely identifying the section of configuration memory where the DPU is mapped and allowing us to focus subsequent corruption operations on the accelerator. Such information on the coordinates of the section of the configuration data, in terms of frames and offsets, composes the fault location space to explore.

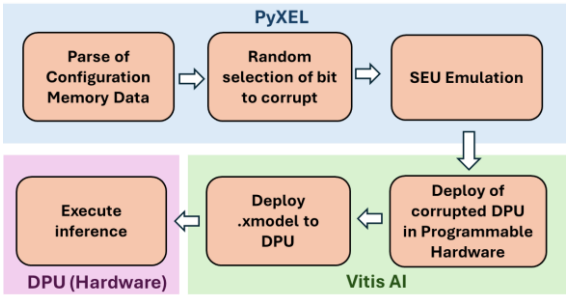

Fig. 1: Single-bit flip injection workflow in the DPU bitstream.

Once the configuration memory analysis is complete, the fault emulation campaign is performed by statistical fault injection [24] relying on the PyXEL toolkit support. To do this, we first randomly select a bit to corrupt within the relevant section to produce a possible error in the hardware structure of the accelerator. The specific bit is corrupted in the configuration data, and a faulty configuration bitstream is generated from the faulty configuration data. Such a configuration file is used to program the programmable hardware subsequently. Once the altered configuration is loaded and the programmable hardware is prepared, we upload a neural network model explicitly compiled for the DPU accelerator and perform inference. The process of creating this neural network model and the fault classification procedure, aimed at evaluating the impact of the fault, is described in greater detail in the following subsections.

C. DPU Neural Network Model Design Flow

To run neural networks on the target acceleration platform, the original models needed to be adapted into a version to be deployed in the DPU architecture. This is necessary since models, typically designed for execution on general-purpose computing systems, are not natively supported by the reconfigurable accelerator. The Vitis AI toolkit provides the toolchain to achieve the conversion. The flow consists of two core components: a quantizer, responsible for both quantization and calibration, and a compiler, which generates the final executable code optimized for the DPU. As illustrated in Figure 2, the conversion flow follows a structured pipeline. The Vitis AI Quantizer transforms all floating-point tensors, including weights, biases, and activations, into an 8-bit integer format. This step significantly reduces computational complexity and memory consumption, while preserving the numerical information necessary for accurate inference. To mitigate the effects of quantization, a post-training calibration is then performed using a representative dataset, allowing the adjustment of scale factors to minimize precision loss and preserve model accuracy. Following the quantization phase, we fed the model to the Vitis AI Compiler targeting the DPU accelerator version. The compiler conducts a static analysis of the model, identifies supported operations, and generates an .xmodel file that encodes the execution graph in a format optimized for hardware deployment. Each operation is mapped directly into the accelerator's computational resources, ensuring efficient execution. The compiled model is then deployed to the reconfigurable device and used in fault-injection experiments conducted under corrupted configurations.

D. Application-Level Comparison

To enable a systematic and fair comparison between the hardware-level and application-level evaluation methodologies commonly used in robustness analysis, a dedicated infrastructure was developed to perform controlled fault injection directly within the neural model at the application level. By injecting faults at the application level, isolating the network's response to data-level corruptions is possible independently of the underlying hardware effects. This allows the analysis to focus purely on the model's internal computational behaviour, providing a clear reference point for interpreting results obtained through hardware-level fault campaigns.

A custom framework based on TensorFlow was implemented to simulate internal faults during inference. The framework takes as input a model saved in .h5 format, corresponding to a standard floating-point neural network, and a configuration specifying whether the fault targets the weights or the activations. Once the model is loaded, the framework analyses the computational graph and identifies injection points according to the selected fault type.

For weight faults, the system detects all layers containing learnable parameters and randomly selects one bit to corrupt. In particular, a single scalar element is extracted from the chosen weight tensor and converted into its 32-bit binary representation. A single random bit is then flipped. This operation emulates a low-level corruption that may occur in memory or during parameter access, providing insight into the

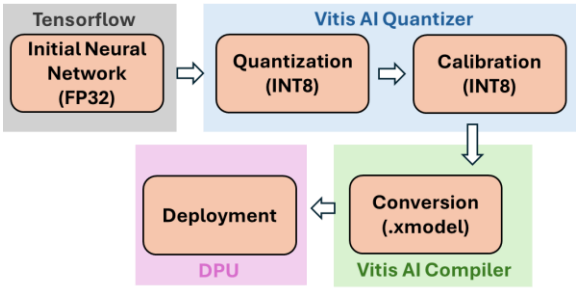

Fig. 2: Conversion workflow from FP32 model to DPU model using Vitis AI.

sensitivity of the model to fine-grained perturbations in its trained parameters.

Regarding the emulation of faults affecting activation, inner states, or datapath, the framework randomly selects an intermediate layer and modifies the model by inserting a custom post-layer operation. This additional operation acts as a wrapper during inference, intercepting the activation tensor and forcing a specific bit position to remain fixed to the logical value 1. This simulates a persistent stuck-at-1 fault in the dynamic data path of the network. Such an approach reflects transient computation errors that manifest during activation propagation and allows for controlled, repeatable testing of fault effects on signal flow within the network.

In both fault scenarios, the output is a corrupted model that is evaluated on a fixed test dataset. The predictions are then compared to those of the fault-free reference model. This process detects the functional impact of injected faults at the application level and permits a direct comparison with the behaviour observed in hardware-level fault injection experiments.

V. Experimental Analysis

In this section, we present a detailed description of the analysis of the resilience of neural networks executed on the DPU accelerator. The entire experimental campaign has been structured into four subsections: a description of the experimental setup, the metrics adopted to evaluate the impact of faults, the analysis of results obtained at the hardware level, and finally, a direct comparison with the application-level analysis.

A. Experimental Setup

The entire hardware-level experimental campaign was conducted using Kria KV260 platforms. Kria is a development board based on the Zynq UltraScale+ MPSoC that supports edge applications and AI workloads. The SoC on the board integrates a quad-core ARM Cortex-A53 processor alongside programmable logic, enabling acceleration through precompiled hardware overlays provided by the vendor. The most performant hardware DPU overlay provided by AMD for Kria is a DPU B4096, a processing unit optimized for INT8 inference capable of efficiently executing key operations such as convolutions, pooling, and fully connected layers. The architecture of this accelerator includes a highly parallel array of computing engines, an internal instruction fetch/unit and scheduler, and dedicated memories and data movers that manage memory accesses between the programmable logic and main memory. The DPU B4096 allows the DPU to execute inference workloads with low latency and high throughput, making it suitable for real-time AI applications. Furthermore, the precompiled overlay is integrated with the Vitis AI development environment and Vitis AI runtime layer, which facilitates compilation, deployment, and runtime control of neural networks, thus enabling seamless use and experimentation with different models and configurations.

TABLE I: CNN Model Specifications

Model	Parameters (M)	Depth	Input
EfficientNetB0	5.3	132	224x224x3
InceptionResNetV2	55.9	449	299x299x3
InceptionV3	23.9	189	299x299x3
MobileNet	4.3	55	224x224x3
MobileNetV2	3.5	105	224x224x3
MobileNetV3	4.0	155	224x224x3
ResNet50	25.6	107	224x224x3
ResNet50V2	25.6	103	224x224x3
ResNet101	44.7	209	224x224x3
ResNet101V2	44.7	205	224x224x3
ResNet152	60.4	311	224x224x3
ResNet152V2	60.4	307	224x224x3
VGG16	138.4	16	224x224x3
VGG19	143.7	19	224x224x3

While the hardware-level analysis requires running on the actual hardware, the application-level experiments are independent of the hardware platform. Both the application-level campaign for weight and activation fault injections, were executed independently on a workstation equipped with an Intel Core i9 processor, 32 GB of RAM, and an NVIDIA GeForce RTX 4070 GPU. The entire experiment was conducted within a TensorFlow-based environment.

As the set of benchmark models, we selected 14 pretrained neural network models for image classification. The models are trained on the ImageNet dataset and used for image classification tasks. Table I details their model architecture, such as the number of parameters, depth, and input size. To ensure consistency and comparability of the results, all evaluations were carried out using a shared test set of 500 images, randomly sampled from the original dataset. The same networks and test sets were employed across all experimental campaigns, both at the hardware and application levels.

B. Experimental Campaign

Experimental analysis is based on fault injection campaigns to evaluate the reliability of neural network models. Each neural network model was assessed by running it on DPU against SEU in CRAM. This analysis is referred to as hardware-level fault injection. It acts directly on the accelerator's hardware architecture through manipulation of the DPU configuration data. The same models have also been evaluated at the application level, simulating faults within the neural network model, explicitly targeting the weights and the activations.

Both hardware and application-level campaigns are based on statistical fault injections. Each campaign involved 5,000 fault injections per network, a number sufficient to obtain a statistically meaningful estimation of average error rates with a 95% confidence interval, as per [24]. The fault location selected for injection for the hardware-level analysis was shared among neural network models, which means that all network models in the benchmark have been subjected to the same hardware faults.

Hardware-level fault injections involving the actual hardware system have been conducted directly on the board. To speed up the fault injection campaigns, we used three

different Kria KV260 platforms, each configured with the same hardware and software stack, supporting fully automated campaign execution. PyXEL and Vitis AI frameworks run directly on the SoC processor system. PyXEL performed configuration memory manipulation and the configuration of the DPU, while Vitis AI handled the execution of neural network models directly on the accelerator. The entire process, from bitstream analysis and corruption to inference, was carried out autonomously on the boards themselves, without needing a host computer.

The application-level analysis was carried out separately on a single machine equipped with an Intel Core i9 and an NVIDIA GeForce RTX 4070 GPU, which was used to accelerate the execution of the fault injection campaigns. All experiments were performed within a TensorFlow environment.

The average time required for each fault injection was measured during the campaigns, including both the injection and model inference phases. As reported in Table II, timing varied depending on the approach. For hardware-level injections, the process required approximately 0.2 seconds for bitstream modification, 9.4 seconds for device reconfiguration, and 0.9 seconds for inference, resulting in a total average of around 10.5 seconds per injection. In contrast, application-level campaigns, which do not involve reconfiguration, required an average of 0.7 seconds for injection and 2.1 seconds for inference, with a total average time of 2.8 seconds per injection. It is important to note that these values represent averages, as execution times may vary slightly depending on the specific neural network, especially the inference time.

TABLE II: AVERAGE FAULT INJECTION TIMES

	Data Corruption	**Configuration**	**Inference**
Hardware-Level	~0,2s	~9,4s	~0,9s
Application-Level	~0,7s	No-Configuration	~2,1s

C. Metrics

To evaluate each fault's impact, the networks' output has been classified into categories. The proposed categories describe both variations in the predicted class and in the associated confidence scores.

During each fault injection experiment, a single fault was emulated at a specific location. In hardware-level campaigns, this involved flipping a bit in the configuration memory, whereas in application-level campaigns, it corresponded to a bit flip in a tensor element (e.g., weight or activation). For every injected fault, the system performed 500 inferences, one for each image in the test set. The predicted class and associated confidence score were recorded for each inference performed by the faulty system. These outputs were then compared to the results obtained from the fault-free system. Each prediction is categorized to assess whether it matched the fault-free classification, retained the same predicted class but with a change in confidence score, or resulted in a misclassification. Based on the classification associated with each outcome of the network under a specific fault, that fault was assigned to one of the following impact categories. A fault was classified as **Masked (M)** if it did not affect any of the 500 outcomes, both in terms of prediction score and final classification of any outcome. It was considered **Low Degradation (LD**) if it altered only the confidence score while maintaining the correct class for all 500 outcomes. **A Medium Degradation (MD)** was assigned if the fault caused a misclassification rate of up to 10% (i.e., from 1 to 50 misclassifications), while **Strong Degradation (SD)** indicated a misclassification rate above 10%. Finally, cases where the device halted or crashed were labelled as **Timeout (T).** These metrics enabled a consistent and accurate description of fault effects, allowing for a direct comparison between hardware-level and application-level fault analysis.

D. Experimental Results of the DPU Accelerator

The hardware-level fault injection campaigns experiment focused on evaluating the fault resilience of the DPU accelerator during neural network inference. The results of this analysis are summarized in Table III, which reports the percentage distribution of fault outcomes across different hardware resources, along with an overall view of the system's fault response. The data show that a significant number of injected faults, about 86.07%, result in masking at the hardware level, meaning they do not produce any observable effect on the output of the inference process. This masking level is consistent with typical results observed in faults affecting the configuration memory, where the percentage of faults propagated to the outcome often falls within the 1% and 15% range. This is primarily because many CRAM bits do not directly control active logic, even when using the associated resource or tile. Consequently, a bitflip affecting one of these bits is unlikely to alter the functional behaviour of the design under test. Additionally, mechanisms such as electrical and logic masking further reduce the propagation of faults to the output, contributing to the overall high masking rate. However, the remaining 13.93% of faults are unmasked and lead to deviations in inference results. These include medium to severe output degradations, which may compromise the system's reliability in safety-critical applications.

By analysing the impact across different hardware components, we observe comparable levels of vulnerability. In particular, flip-flops, interfaces, lookup tables, and interconnections exhibit non-marginal rates of unmasked faults, with medium-to-severe degradation occurring in 11.2%, 15.3%, 14.7%, and 17.1%. These findings may help pinpoint the most sensitive regions of the DPU architecture, offering practical insights into which areas may require targeted protection or fault-tolerance mechanisms to improve system robustness.

Regarding the specific hardware component affected, the effects of unmasked faults exhibit a consistent trend across

TABLE III: DPU RESOURCE RESPONSES TO BIT-FLIP FAULTS

Resource	**M (%)**	**WD (%)**	**MD (%)**	**SD (%)**	**T (%)**
BRAM Interface	100	0	0	0	0
FF	88,3	0,4	4,5	6,7	0
Interconnection	81,9	0,3	3,9	13,2	0,7
Interface	83,4	0,6	4,5	10,8	0.6
IO	100	0	0	0	0
LUT	84,7	0,2	11,4	3,3	0,3
Other	82,2	0.3	6,6	10,4	0,6
Faults-injection effects on DPU	86,07	0,28	5,92	6,61	1,12

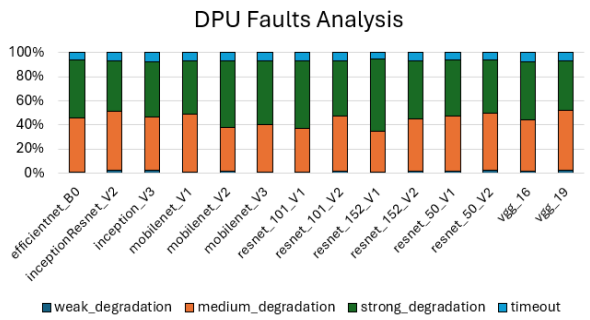

Fig. 3: Overview of fault-injection effects on the DPU accelerator for neural-network inference.

different neural networks. As illustrated in Figure 3, medium degradation is the most frequent outcome, followed by strong degradation and, less frequently, timeouts. These results emphasize that unmasked faults often lead to noticeable deviations in output. Moreover, the similarity in the error distribution patterns across all tested networks suggests that the DPU's fault response is mainly independent of the model architecture, highlighting a uniform and architecture-independent behaviour.

E. Application vs. Hardware Fault Tolerance Comparison

To allow a direct and fair comparison between different methodologies, we extended the analysis by performing fault injection at the application level for the models evaluated in hardware. We compared the two analyses on the same 14 neural networks and evaluated robustness on the same test set using the same evaluation metrics of the hardware-level experiments. A summary of the results for the two application-level campaigns is shown in Table IV. The first clear result from the comparison is the substantial difference in fault masking. While the DPU was able to mask 86.07% of injected faults, the application-level tests showed much lower masking rates. However, this characteristic is intrinsic to the specific test methodology, as already discussed when commenting masked faults for the hardware-based campaign.

To better characterize the system behaviour in the presence of unmasked faults, Figures 3 and 4 report the distribution of fault effects, considering only those cases in which the fault produced an observable impact. This analysis reveals how each environment reacts once a fault bypasses the internal masking mechanisms. On the DPU, unmasked faults frequently led to medium (5.92%) or strong (6.61%) degradation, and in some cases to a complete system timeout (1.12%), indicating that hardware-level faults can result in more severe consequences. In contrast, application-level fault

TABLE IV: SUMMARY OF MEAN FAULT-INJECTION EFFECTS ON DPU AND APPLICATION-BASED INFERENCE

	M (%)	**WD (%)**	**MD (%)**	**SD (%)**	**T (%)**
DPU – Hardware Faults	86.07	0.28	5.92	6.61	1.12
Application Layer – Weights Faults	0.13	50.14	24.51	25.22	0
Application Layer – Activation Faults	22.76	58.02	12.68	6.54	0

injections showed only a minor impact on the output, with most cases resulting in low degradation, 50.14% for weight faults and 58.02% for activation faults.

A further relevant observation emerges when analysing how individual neural networks respond to injected faults. As illustrated in Figure 4, application-level injections result in highly variable behaviour depending on the network architecture, indicating a strong dependency on the model's internal structure. In contrast, Figure 3 shows that faults injected into the DPU produce a consistent error distribution across all networks tested, not dependent on the specific network model. This suggests that hardware-level fault effects are largely model-independent, while application-level resilience is more tightly coupled to the particular design of the neural network. Considering how hardware fault campaigns model the authentic computation architecture in more detail, these findings suggest that fault injection performed exclusively at the application level tends to mischaracterize fault impact severity. Additionally, it suggests an inherent fault tolerance capability of some neural network models that has no match in real hardware analysis. To achieve a comprehensive and realistic evaluation of neural network resilience, it is therefore essential to incorporate hardware-level fault injection, which captures low-level effects. Additionally, application-level approaches may lead to baseless conclusions.

VI. CONCLUSIONS AND FUTURE WORKS

This work presented a fault-resilience analysis of deep neural networks executed on an FPGA-based platform, focusing on the DPU accelerator. The study compared hardware-level and application-level fault injection strategies. The results showed that application-level campaigns tend to underestimate the overall impact of faults. In contrast, hardware-level campaigns are fundamental to providing a more accurate view of hardware behaviour under fault conditions.

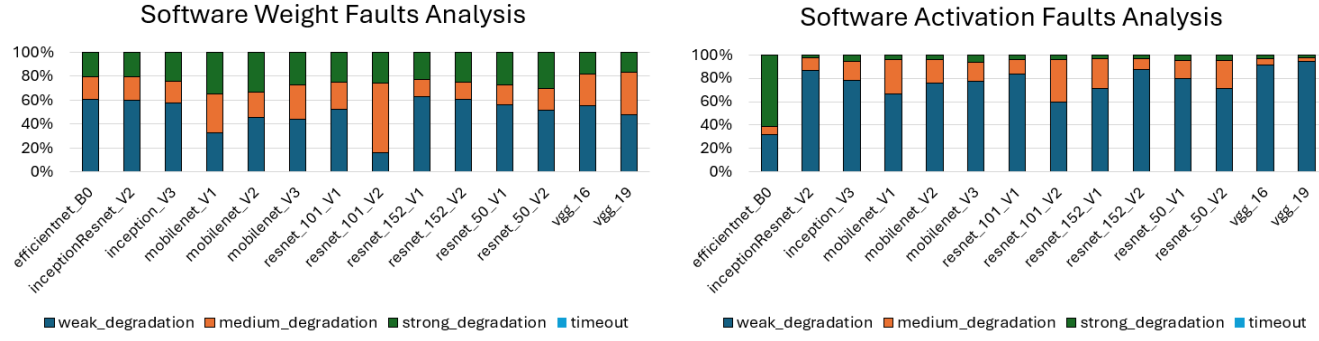

Fig. 4: Overview of fault-injection effects on application-based neural-network inference.

Future work will focus on further investigating the effects of faults injected at different levels of abstraction, with particular attention to the distinction between configuration-level and data-level corruption. This analysis is essential, as it may demonstrate that application-level fault injection could still serve as a valid approximation when modelling data-level faults only, thus offering a possible trade-off between evaluation accuracy and implementation complexity.

ACKNOWLEDGMENT

This research is part of the project "SPACE IT UP!", under Contract No. 2024-5-E.0. – CUP: I53D24000060005. The authors acknowledge the valuable support provided for the implementation and development of the project.

REFERENCES

[1] Redmon and A. Angelova, "Real-time grasp detection using convolutional neural networks," in 2015 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2015, pp. 1316–1322.

[2] O. I. Abiodun, A. Jantan, A. E. Omolara, K. V. Dada, N. A. E. Mohamed, and H. Arshad, "State-of-the-art in artificial neural network applications: A survey," Heliyon, vol. 4, 2018, Art. no. e00938.

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

[4] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei, "Large-scale video classification with convolutional neural networks," in Proceedings of the IEEE conference on Computer Vision and Pattern Recognition, 2014, pp. 1725–1732.

[5] L. Deng, J. Li, J.-T. Huang, K. Yao, D. Yu, F. Seide, M. Seltzer, G. Zweig, X. He, J. Williams et al., "Recent advances in deep learning for speech research at microsoft," in 2013 IEEE International Conference on Acoustics, Speech and Signal Processing. IEEE, 2013, pp. 8604–8608.

[6] Y. Kim, "Convolutional neural networks for sentence classification," arXiv preprint arXiv:1408.5882, 2014

[7] National Academies of Sciences, Engineering, and Medicine, Thriving on Our Changing Planet: A Decadal Strategy for Earth Observation from Space. Washington, DC: The National Academies Press, 2018.

[8] Xilinx. DPUCZDX8G for Zynq UltraScale+ MPSoCs Product Guide, PG338 (Version 4.1), Advanced Micro Devices, Inc., January 23, 2023

[9] A. Cratere *et al*., "Efficient FPGA-Accelerated Convolutional Neural Networks for Cloud Detection on CubeSats," in *IEEE Journal on Miniaturization for Air and Space Systems*, doi: 10.1109/JMASS.2025.3533018.

[10] L. Ioannou and S. A. Fahmy, "Network Intrusion Detection Using Neural Networks on FPGA SoCs," *2019 29th International Conference on Field Programmable Logic and Applications (FPL)*, Barcelona, Spain, 2019, pp. 232-238, doi: 10.1109/FPL.2019.00043.

[11] E. Vacca, et al. "Failure rate analysis of radiation tolerant design techniques on SRAM-based FPGAs", Microelectronics Reliability, 2022, doi.org:10.1016/j.microrel.2022.114778.

[12] B. Reagen et al., "Ares : A framework for quantifying the resilience of deep neural networks," in Proc. 55th ACM/ESDA/IEEE Des. Automat. Conf., 2018, pp. 1–6.

[13] A. Ruospo, A. Bosio, A. Ianne, and E. Sanchez, "Evaluating convolutional neural networks reliability depending on their data representation," in Proc. 23rd Euromicro Conf. Digit. System Des., 2020, pp. 672–679.

[14] A. Ruospo, A. Balaara, A. Bosio, and E. Sanchez, "A pipelined multi-level fault injector for deep neural networks," in Proc. IEEE Int. Symp. Defect Fault Tolerance VLSI Nanotechnol. Syst., 2020, pp. 1–6.

[15] C. De Sio, S. Azimi and L. Sterpone, "FireNN: Neural Networks Reliability Evaluation on Hybrid Platforms," in IEEE Transactions on Emerging Topics in Computing, vol. 10, no. 2, pp. 549-563, 1 April-June 2022, doi: 10.1109/TETC.2022.3152668.

[16] B. Du, S. Azimi, C. de Sio, L. Bozzoli, and L. Sterpone, "On the reliability of convolutional neural network implementation on SRAMbased FPGA," in 2019 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), October 2019, pp. 1–6.,

[17] F. Benevenuti, F. Libano, V. Pouget, F. L. Kastensmidt, and P. Rech, "Comparative analysis of inference errors in a neural network implemented in SRAM-based FPGA induced by neutron irradiation and fault injection methods," in 2018 31st symposium on Integrated Circuits and Systems Design (SBCCI), August 2018, pp. 1–6.

[18] E. Vacca et al., "Analyzing the SEU-induced Error Propagation in Systolic Array on SRAM-based FPGA", 2023 23rd European Conference on Radiation and Its Effects on Components and Systems (RADECS), pp. 1-4, doi: 10.1109/RADECS59069.2023.10767017.

[19] F. F. D. Santos et al., "Analyzing and increasing the reliability of convolutional neural networks on GPUs," IEEE Trans. Rel., vol. 68, no. 2, pp. 663–677, Jun. 2019.

[20] S. Sabogal and A. George, "A Methodology for Evaluating and Analyzing FPGA-Accelerated, Deep-Learning Applications for Onboard Space Processing," 2021 IEEE Space Computing Conference (SCC), Laurel, MD, USA, 2021, pp. 143-154, doi: 10.1109/SCC49971.2021.00022

[21] L. Bozzoli, C. De Sio, L. Sterpone and C. Bernardeschi, "PyXEL: An Integrated Environment for the Analysis of Fault Effects in SRAM-Based FPGA Routing," *2018 International Symposium on Rapid System Prototyping (RSP)*, Turin, Italy, 2018, pp. 70-75, doi: 10.1109/RSP.2018.8632000.

[22] AMD/Xilinx, "Vitis AI User Guide (UG1414)," Version 3.5, Xilinx, 28 Sep. 2023. https://docs.amd.com/r/en-US/ug1414-vitis-ai

[23] Cristiana Bolchini, Luca Cassano, Antonio Miele, "Resilience of deep learning applications: A systematic literature review of analysis and hardening techniques", Computer Science Review, Volume 54, 2024, 100682, ISSN 1574-0137, https://doi.org/10.1016/j.cosrev.2024.100682.

[24] A. Ruospo et al., "Assessing Convolutional Neural Networks Reliability through Statistical Fault Injections," 2023 Design, Automation & Test in Europe Conference & Exhibition (DATE), Antwerp, Belgium, 2023, pp. 1-6, doi: 10.23919/DATE56975.2023.10136998.