Hardware-Aware Software-Based Fault Injection Platform for DNN Accelerators

# POSTER: Hardware-Aware Software-Based Fault Injection Platform for DNN Accelerators

Eleonora Vacca
eleonora.vacca@polito.it
Politecnico di Torino
Turin, Italy

Federico Buccellato
federico.buccellato@polito.it
Politecnico di Torino
Turin, Italy

## ABSTRACT

Ensuring reliable execution of Deep Neural Networks (DNNs) is crucial for safety-critical applications. Traditional software-based approaches fail to capture real-world fault scenarios, overlooking accelerator datapath effects. We propose a hardware-aware, software-based fault injection platform that emulates systolic array processing, enabling effective fault propagation analysis without the overhead of time-consuming methods.

## CCS CONCEPTS

• **Hardware** → **Robustness**; **Safety critical systems**;

## KEYWORDS

DNN, Reliability, Fault Injection, Systolic Arrays

**ACM Reference Format:**
Eleonora Vacca and Federico Buccellato. 2025. POSTER: Hardware-Aware Software-Based Fault Injection Platform for DNN Accelerators. In *Proceedings of May 28–30, 2025 (CF '25).* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3719276.3727952

## 1 INTRODUCTION

The reliability evaluation of deep neural network models is a critical aspect of deploying artificial intelligence applications in real-world scenarios. Most evaluations rely on software-based fault injection techniques, such as TensorFlow FI[4]. These methodologies introduce faults at high abstraction levels, typically flipping bits in DNN weights and bias or corrupting inter-layer inputs and outputs. However, these approaches remain independent of the underlying hardware platform, assessing only the robustness of the model itself. However, the response of a DNN model to faults is significantly shaped by the characteristics of the hardware accelerator executing it, as it defines how faults propagate and impact the DNN outcome. While lower-level fault injection techniques, such as radiation testing[3] [7] or HDL simulation environments[2], provide more precise evaluations, they are often impractical due to high cost and long execution times, respectively. Another common approach to reliability evaluation involves using FPGA-based fault emulation[1] [6]. This technique flips bits in the configuration memory (CRAM) content to emulate faults within the circuit netlist. A significant drawback of FPGA-based fault injection is that faults are typically injected randomly, without a clear guarantee of targeting the specific resources associated with the implemented circuit. Even when using information such as the essential bits provided by AMD Xilinx—which represent a subset of configuration bits more likely to be relevant to the design—there is still an overwhelming number of bits to consider. Indeed, the encoded nature of the bitstream means that users lack explicit knowledge of which bits correspond to specific hardware modules. As a result, injecting faults into a specific module to observe its behavior is not straightforward and requires extensive reverse engineering of the FPGA bitstream, making it an impractical approach for targeted fault analysis. In this paper, we propose a hardware-aware software-based fault injection platform that emulates the execution of DNNs on systolic arrays-based accelerators while incorporating hardware-level fault modeling. By bridging the gap between high-level software-based fault injection and low-level hardware execution, our approach enables a fast, accurate, and scalable reliability evaluation of DNN models.

## 2 PROPOSED APPROACH

The proposed framework evaluates the reliability of DNNs by considering their execution on a systolic array (SA) accelerator, incorporating the accelerator's datapath in fault propagation analysis. This is achieved through a detailed modeling of DNN computation within the SA while simulating the effects of permanent faults. An overview of the implemented platform is in Fig. 1. The platform receives as input a trained DNN model and produces as output a model of computation (MoC) embedding user defined fault models. To produce the MoC, the first step consists of translating the DNN layers into a proper sequence of General Matrix Multiplication (GEMM) operations to match the SA datapath, characterized by a 2D array of processing elements (PEs). Hence, for each DNN layer, the weights are extracted, as well as information on the input/output shape of the layer. Additionally, operation tiling to accommodate the SA size must be considered to fully emulate an SA. This involves partitioning the weight matrices into smaller submatrices based on the SA size, a key parameter introduced by the user, mimicking real hardware execution constraints. Operation tiling typically introduces operations overhead, as well as an additional point of failure in the hardware execution, since the results of subsequent submatrices need to be recombined employing additions. Once the operation tiling is performed, the MoC can be finalized, adding the desired fault model. To introduce faults, the framework generates a fault map based on user specifications. For instance, if the user wants to evaluate the effect of a permanent fault on the $PE_{i,j}$ , the


*CF '25, Cagliari, Italy,*

ACM ISBN 979-8-4007-1528-0/2025/05
https://doi.org/10.1145/3719276.3727952

fault map ensures that all computations routed through this PE are consistently impacted. This fault map is then applied at every layer execution step, and for every layer of the model, allowing a realistic propagation of hardware faults into the model execution. Currently, faults are modeled as stuck-at at the input, output, or weight. The framework supports several fault injection modes. The user mode, where the user has to provide fault location and type of fault (stuck-at 0 or stuck-at 1), the basic mode, where at each iteration a single PE is picked randomly and either its input or output or weight data is selected as fault source. The exhaustive mode, where all the PEs are impacted progressively, once affecting the input data, once the weights, and once the output. Finally, the accumulation mode where the number of corrupted PEs increases at each iteration.

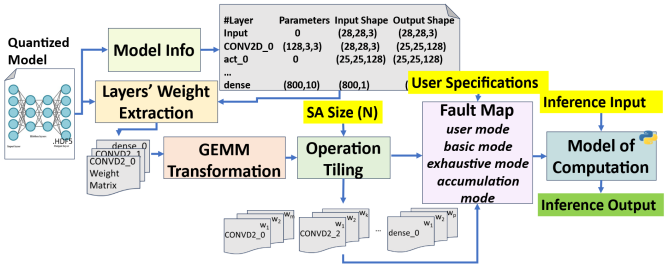


**Figure 1: The implemented fault injection platform.**

## 3 EXPERIMENTAL RESULTS

The developed platform is compared with two baseline methods: an FPGA-based hardware fault emulation approach, and a classical software-based fault injection technique. For the FPGA-based campaign, we implemented an open-source TPU core[5] equipped with a $14 \times 14$ systolic array (SA). Consequently, in our proposed platform, we implemented the MoC using the same SA size.

As benchmark applications, we targeted two CNN models on the MNIST digit and CIFAR-10 datasets. The injection campaign consisted of 5,000 fault injections. In the FPGA-based approach, the fault is modelled as a bit-flip in the configuration memory (CRAM), causing a structural fault in the datapath. In our platform, the fault is modelled as a stuck-at fault at the weight resource of a single processing element (PE), while in the software method, 0.5% of the total possible bit-flip faults across the CNN weights are considered.

Injection results are reported as the percentage of faults inducing a misclassification and those causing a silent data corruption (SDC), i.e., a deviation in the top-1 class score that does not compromise the classification output. Results are shown in Table 1.

The software-based methodology underestimates realistic fault rates, as its misclassification rate is significantly lower than that of the other approaches. This suggests that while software-based bit-flip models can provide a preliminary assessment of transient faults on data, they fail to capture datapath influence. In contrast, our MoC introduces nearly 4× and 2× higher misclassification rates than the FPGA-based method for CIFAR-10 and MNIST, respectively. This is due to the randomness of FPGA-based fault injection, where bitstream details are unknown, leading to less precise targeting of SA resources. Our approach, however, directly models faults in the SA core, effectively evaluating datapath-induced fault propagation.

**Table 1: Comparison results over 5,000 fault injections for our proposed platform, the FPGA-based and software FI methods.**

| Fault Inj. Mode | CIFAR-10 | | MNIST | |
|---|---|---|---|---|
| | Misclass. (%) | SDC (%) | Misclass. (%) | SDC (%) |
| Our MoC | 60.29 | 81.41 | 19.68 | 81.56 |
| FPGA-based | 15.24 | 17.00 | 11.38 | 24.22 |
| Software | 6.32 | 37.28 | 3.62 | 31.10 |

To validate our results, we compared MNIST accuracy after fault injection with the study in[2], which implemented a $256 \times 256$ SA on a 45 nm PDK library and injected gate-level faults. Their reported accuracy (around 60–80% for 0.5% faulty PEs) aligns with our MoC, which achieved 75% accuracy under similar conditions. This confirms our approach's consistency with low-level fault injection while avoiding its time-consuming nature.

## 4 CONCLUSION

We propose a software-based fault injection platform for DNN reliability assessment, using a MoC to emulate benchmark execution on SAs. By comparing it with traditional fault injection methods, we demonstrate its effectiveness in modeling fault propagation and its impact on accuracy.

## REFERENCES

[1] E. Vacca et al. 2024. ZOR: Zero Overhead Reliability Strategies for AI Accelerators. In *IEEE Interregional NEWCAS Conference*. https://doi.org/10.1109/NewCAS58973.2024.10666350

[2] J. J. Zhang et al. 2018. Analyzing and mitigating the impact of permanent faults on a systolic array based neural network accelerator. In *2018 IEEE 36th VLSI Test Symposium (VTS)*. https://doi.org/10.1109/VTS.2018.8368656

[3] P. R. Bodmann et al. 2024. Neutrons Sensitivity of Deep Reinforcement Learning Policies on EdgeAI Accelerators. *IEEE Transactions on Nuclear Science* (2024).

[4] Z. Chen et al. 2020. TensorFI: A flexible fault injection framework for TensorFlow applications. In *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*.

[5] J. Fuhrmann. 2018. Implementierung einer Tensor Processing Unit mit dem Fokus auf Embedded Systems und das Internet of Things. http://hdl.handle.net/20.500.12738/85276

[6] E. Vacca, G. Ajmone, and L. Sterpone. 2023. RunSAFER: A Novel Runtime Fault Detection Approach for Systolic Array Accelerators. In *2023 IEEE 41st International Conference on Computer Design (ICCD)*. Washington, DC, USA, 596–604. https://doi.org/10.1109/ICCD58817.2023.00095

[7] 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. https://doi.org/10.1016/j.microrel.2022.114778