Special Session: Reliability Assessment Recipes for DNN Accelerators

Mohammad Hasan Ahmadilivani, Alberto Bosio, Bastien Deveautour, Fernando Fernandes dos Santos, Juan David Guerrero Balaguera, Maksim Jenihhin, Angeliki Kritikakou, Robert Limas Sierra, Salvatore Pappalardo, Jaan Raik, et al.

Special Session: Reliability Assessment Recipes for DNN Accelerators

Mohammad Hasan Ahmadilivani[1], Alberto Bosio[2], Bastien Deveautour[2], Fernando Fernandes dos Santos[3], Juan-David Guerrero-Balaguera[4], Maksim Jenihhin[1], Angeliki Kritikakou[3], Robert Limas Sierra[4], Salvatore Pappalardo[2], Jaan Raik[1], Josie E. Rodriguez Condia[4], Matteo Sonza Reorda[4], Mahdi Taheri[1], and Marcello Traiola[3]

[1]Tallinn University of Technology, Tallinn, Estonia
[2]Ecole Centrale de Lyon, CPE Lyon, INL, Ecully, France
[3]Univ Rennes, CNRS, Inria, IRISA - UMR 6074, F-35000 Rennes, France
[4]Politecnico di Torino, Turin, Italy

***Abstract*—Reliability assessment is mandatory to guarantee the correct behavior of Deep Neural Network (DNN) hardware accelerators in safety-critical applications. While fault injection stands out as a well-established, practical and robust method for reliability assessment, it is still a very time-consuming process. This paper contributes with three recipes for optimizing the efficiency of the reliability assessment: a) hybrid analytical and hierarchical FI-based reliability assessment for systolic-array-based DNN accelerators; b) mixing techniques for the reliability assessment of in-chip AI accelerators in GPUs; c) reliability assessment of DNN hardware accelerators through physical fault injection. The experimental results demonstrate the efficiency of the proposed methods applied to their target DNN HW accelerator platforms.**

***Index Terms*—deep neural networks, approximate computing, fault simulation, error emulation, reliability, resiliency assessment**

I. Introduction

Deep Neural Networks (DNNs) are a powerful tool assisting different aspects of human life, e.g., healthcare, transportation, security, IoT and edge applications [1] [2]. They are characterised by the extremely complex computational kernels (i.e., several giga operations per second) and the high amount of parameters to be transferred from and to the memory (i.e., in the order of gigabytes). To achieve target energy efficiency and high performance, several types of DNN Hardware Accelerators (DNN-HAs) for the execution of DNN kernels were proposed. [3].

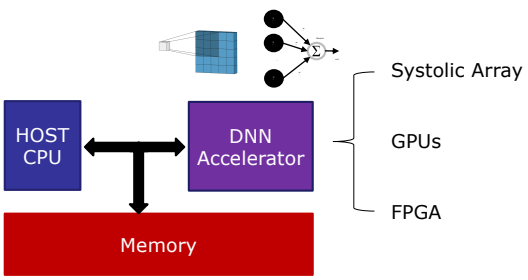


Fig. 1: DNN HW accelerator system

*The authors are sorted in alphabetic order.

Fig. 1 sketches a system-level view of a DNN-HA. The latter is controlled by a microprocessor host and has to access the main memory to retrieve data. The most widely used DNN-HAs can be classified as Systolic Array (i.e., TPU), GPU and FPGA. Independently from the type, employing DNN accelerators in safety-critical applications raises hardware reliability concerns. For example, in compliance with the ISO 26262 functional safety standard for road vehicles, the FIT (Failures In Time) rate of particular hardware components has to be 10 failures in 1 billion hours of operation at maximum to meet the target safety integrity level, which necessitates very circumspect design [4], [5].

The reliability of DNN accelerators is boosted by their intrinsic ability to function correctly even in the presence of environment-related faults (soft errors, electromagnetic effects, temperature variations) or faults in the underlying hardware (manufacturing defects, process variations, nanoelectronics aging effects) [6]. DNNs are known to be resilient to faults due to their numerous interconnected layers and the ability to mask faults [7] [8]. Unfortunately, assessing the reliability of a DNN accelerator is not a trivial task [9], [10]: it depends on several factors, such as the training set, the data type, and the quality of the test set [11]. On top of that, we need to consider the hardware that performs the computations since specific platforms have specific faults [12].

There are three main methodologies for DNNs' reliability assessment radiation-based, platform-based, and simulation-based [13]:

- **Simulation-based:** The injection process is carried out without relying on the physical device finally running the NN. Moreover, depending on the abstraction level, they can be further ranked.
  - **Software Level:** The injections are performed on a high-level model of the NN, not considering any details of the actual hardware architecture.
  - **Hardware Level:** The injections are performed on a more accurate model of the NN that simulates the target hardware architecture. E.g., this can be described at the register transfer level (RTL) or gate level.
- **Platform-based:** The measurements and the analyses are

performed directly on a physical device that emulates the final implementation of a design using FPGAs or on physical platforms running the NN, e.g., CPUs and GPUs.
- **Radiation-based:** The reliability assessment is performed in the actual platform running the NN under assessment by means of external electromagnetic interference, such as ionizing particle incidence through accelerated radiation test campaigns.

The goal of this paper is to present advanced methods to assess the reliability of the three types of DNN-HAs, i.e., Systolic Array, GPU and FPGA, through different methodologies.

The paper is organized as follows: Section II discusses a hybrid approach for Systolic Array DNN accelerator. Section III presents a method for evaluating the impacts of faults in GPU by considering its low-level micro-architecture description and its functional operation. Section IV leverages neutron beam experiments to extract practical – and non-obvious – information about GPUs and FPGA DNN accelerators. Finally, Section V concludes the paper.

II. Hybrid Analytical and Hierarchical FI-based Reliability Assessment for Systolic-Array-Based DNN Accelerators

A. Motivation and Related Work

Simulation-based FI is less expensive in terms of equipment, but at the same time, it implies the most resource-intensive computations and is very time-consuming. On the other hand, analytical and hybrid approaches are proposed to reduce the complexity of exploiting FI for the reliability assessment of DNNs [12]. Analytical approaches attempt to provide mathematical approaches to estimate reliability, nonetheless, their evaluation accuracy is challenging to address and they are mostly hardware-agnostic. Whereas hybrid FI-analytical approaches can take advantage of both FI and analytical approaches in terms of scalability, accuracy and hardware-based analysis [12]. To our knowledge, there is no work assessing the reliability of Systolic Arrays (SAs) running DNNs using hybrid methods to accelerate the analysis process.

This section introduces a novel hybrid analytical and hierarchical simulation-based reliability assessment for systolic-array-based DNN accelerators based on FI. This methodology is tailored to significantly accelerate the fault injection process on systolic-array-based DNN hardware accelerators. The systolic-array core of the DNN accelerators is modeled using a Uniform Recurrent Equation (URE) system [14]. The proposed injection flow has been implemented based on an SA simulator [15], thus offering the advantage of being more precise than a hardware-agnostic tool, yet much faster than traditional RTL-level simulations. An analytical method [16] is used to prune the fault space to further optimize the tool and speed up the reliability assessment process.

B. Proposed method

The methodology for the proposed framework is illustrated in Fig. 2. After providing the trained network parameters and architecture, in Step 1, the fault list is generated. Possible fault locations can be defined by the user or can be a random fault list generated based on the network parameters by the framework.

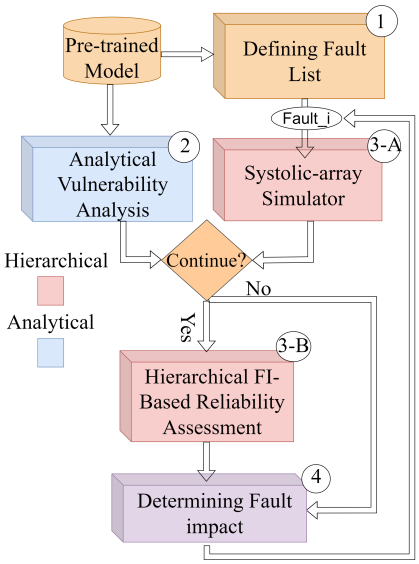


Fig. 2: The proposed methodology for hybrid analytical and hierarchical reliability assessment for systolic array DNN-HAs

In this work, we consider random transient faults in the registers of the systolic array's processing elements.

In Step 2, to prune the fault space, we adopt and extend the DeepVigor methodology presented in [16] that provides vulnerability analysis for DNNs and QNNs, respectively. To this end, we find error values for each output Feauture Map (FMap) that misclassifies the network output. Let $\delta_k^l(X_i)$ be an added positive or negative error value to an output FMap by a fault in the k-th neuron at layer l with input data X_i. For each neuron, we find the minimum positive and maximum negative $\delta_k^l(X_i)$ that misclassifies the output from the golden classification. This value is obtained for all input data X and aggregated over them. The aggregation leads to a vulnerability value range for each neuron, as shown in Fig. 3. It is labeled as follows:

- **Vulnerable** (red area): if a fault deviates the output of a neuron as in this range, it will certainly lead to misclassification for any input.
- **Non-Vulnerable** (green area): if a fault deviates the output of a neuron as in this range, it will not change the output classification for any input.
- **Semi-vulnerable** (grey area): if a fault deviates the output of a neuron as in this range, it might or might not lead to misclassification for any input.

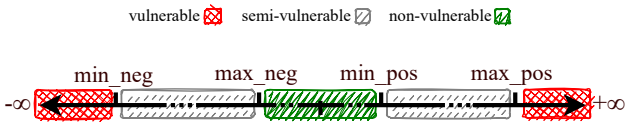


Fig. 3: Vulnerability ranges for fault space pruning

This analysis enables pruning the fault space in a way that we map the deviations at the erroneous outputs of neurons induced by fault to the obtained vulnerability ranges for neurons. *If the error corresponds to the red or green areas, we immediately classify them respectively as critical or non-critical, and do not continue the fault simulation.* Otherwise, if the error

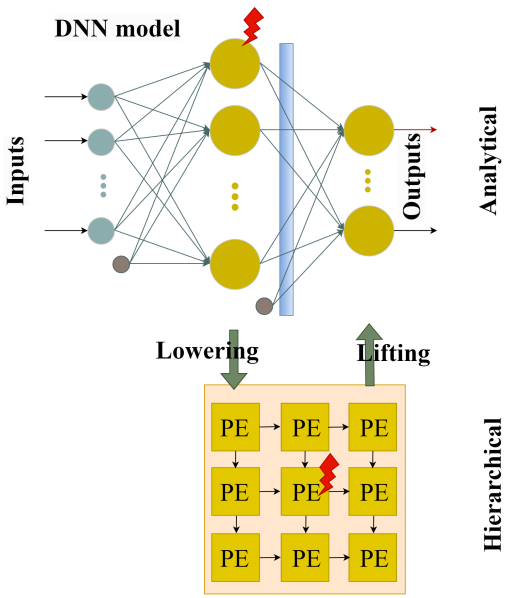


Fig. 4: Fault injection across hierarchical and analytical abstractions

corresponds to a semi-vulnerable range, the fault simulation is required to be performed.

In Step 3-A, the simulation is performed from the beginning of the DNN to the fault's location to obtain the erroneous output of the corresponding neuron. The error is mapped to the vulnerability values of the corresponding neuron (shown in Fig. 3). If the fault is pruned (corresponding to green or red areas), the fault impact is determined (Step 4). Otherwise, the simulation continues (Step 3-B) to obtain the fault impact (Step 4).

In Steps 3-A and 3-B, switching between high-level API and systolic-array simulator is done by solving the URE system mentioned before; this step is described later in this section and Fig. 4. In Step 4, the reliability of the network and the impact of the faults are reported by different metrics. After evaluating the impact of one fault, the next fault is selected for evaluation.

The hardware simulation in Step 3-A is based on a formalization of the problem to solve through a URE system. Such a system can describe the problem to be solved by the architecture through a set of recurrence relations. In our specific case, we are interested in solving the problem of matrix multiplication. We associate the following system to such a task.

$$\begin{aligned} c(i,j,k) &= c(i,j,k-1) + a(i,j-1,k) \times b(i-1,j,k) \\ a(i,j,k) &= a(i,j-1,k) \\ b(i,j,k) &= b(i-1,j,k) \end{aligned}$$

The generalization of this process is described in [14]. This equation system can be associated with actual hardware processing by projecting the iteration space to the physical space. The iteration space contains all the points (i,j,k) of the equation system. The physical space describes where (i.e., which processing element) and when (i.e., at what clock cycle) each computation happens in the real world. To achieve that, the iteration space is projected twice: the first time, the resulting points will correspond to the spatial arrangement of the processing elements; the second projection determines iso-

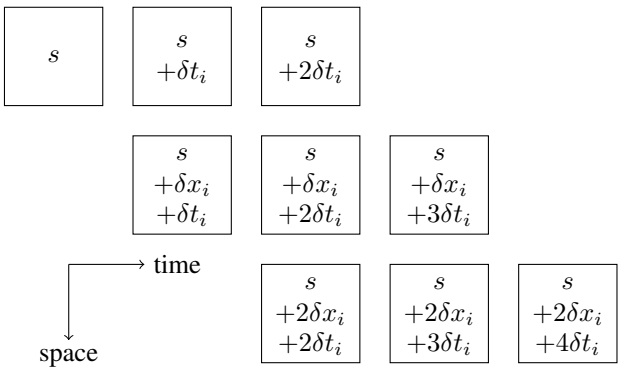


Fig. 5: Fault propagation in systolic array. When injecting element s, the fault is propagated in time (thus affecting elements $s+\delta t_i$ and $s+2\delta t_i$) and in space (forwarding the faulty value to neighboring elements $s+\delta x_i+\delta t_i$, $s+2\delta x_i+\delta t_i$ and so on).

temporal planes, identifying operations that are computed during the same clock cycle but on different processing elements; each plane corresponds to a different clock cycle. The space-projection matrix P and the temporal dimension vector π are used later.

In order to perform the simulation, it is sufficient to solve the system shown above. Nevertheless, this method gives the possibility of injecting faults in the values in a hardware-aware fashion. To achieve the injection, it is sufficient to change the values a, b and c at specific iterations (i,j,k). The faulty values must then be propagated to the following Processing Elements (PEs). Given the system of equations, the propagation can be computed easily, taking into account the transformation matrix. Figure 5 shows the concept. In this case, an injection in the element $s=(x,y,t)$ on the generic line i is done between times 0 and ∞. The injected elements are visible in the figure. Specifically, the fault will propagate in time, thus injecting also $s+\delta t_i$ and $s+2\delta t_i$. In the same way, this fault will propagate in the space to the element cascading from s. The value propagation only happens after each clock cycle, which means that the next injected element will be also displaced in time, thus injecting element $s+\delta x_i+\delta t_i$. In the same way, the latter will propagate to the following element on the following clock cycle, thus injecting element $s+2\delta x_i+2\delta t_i$ and so on. The injected points can be translated into iteration vectors i,j,k using the inverse transformation. Formally, it is sufficient to consider the injection as a function h applied to one of the three values, e.g. $a(i,j,k)=h(a(i,j-1,k)$

C. Experimental Results

To evaluate the methodology, experiments were performed using a 16-bit quantized LeNet-5 trained on the MNIST dataset. The network was injected with random transient faults (assuming that they are caused by Single-Event Upsets). More specifically, a fault affects a random bit in the weight register of a random processing element. The fault causes an error, which manifests itself by fixing the value of a bit to either 0 or 1. The target architecture is an Output-Stationary Systolic Array. In this experiment, faults are injected only in the first layer of the network. It is simulated using the URE system to obtain its output values. The output is compared against the vulnerability ranges obtained by the analytical approach, which determines whether the injected fault is vulnerable, semi-vulnerable or

non-vulnerable (see Fig. 3). If the fault happens to be semi-vulnerable, the simulation must be carried out until the end to determine whether the fault produced misprediction or not; in the other two cases, the output is pre-determined, and there is no need to complete the simulation until the end. It is worth mentioning that the vulnerability ranges are obtained in less than one minute on an NVIDIA 3090 GPU. This methodology allows to accelerate the process of fault injection simulations.

The produced fault list includes 964 random faults, each simulated with 100 random input images.

TABLE I: Channel-wise analysis of the fault injection speedup. The last column (su) indicates the percentage of vulnerable and non-vulnerable simulations with respect to the total.

channel	non-vulnerable	semi-vulnerable	vulnerable	speed-up (%)
0	13673	3127	0	81.38
1	12606	3594	0	77.81
2	13044	2956	0	81.52
3	10392	5508	0	65.35
4	11811	3589	0	76.69
5	13173	2927	0	81.82

Table I shows the number of simulations in each category. The simulations are grouped by channel for better understanding. The first column indicates the channel in which the faults are injected. The "non-vulnerable", "semi-vulnerable", and "vulnerable" columns indicate the number of simulations with the corresponding vulnerability. The "speed-up" column indicates the percentage of simulations that do not need to be fully performed because of the vulnerability ranges gathered in the previous step. More specifically, speed-up is the percentage of pruned simulations.

As observed in Table I, no fault was pruned as vulnerable using the analytical approach. This is because the analytical approach did not provide any red area for the first layer of LeNet-5, yet it provides an effective green area in which up to 81.52% of faults are pruned. Table I shows that consistently, for each layer, at least 65% of the simulations are predetermined using the analytical approach leading to a remarkable time-saving for simulations.

If we assume that x is the time needed to perform a single hardware simulation (computing a convolution between a kernel and a feature map), we would need a total time t for a single inference as follows:

$$t = x \times \sum K_{i,j,l} \qquad \forall (i,j,l) \in NN_f \subseteq NN$$

where K is always equal to 1 and indicates a single convolution and (i,j,l) is the triple $(fmap_{input}, fmap_{out}, \text{layer})$ in the neural network NN. NN_f represents subset of the convolutions affected by the fault f.

For example, in our LeNet-5, the set NN has 6 elements for the first layer: $(1,1,1),(1,2,1),\dots(1,6,1)$. The second layer has 6×16 elements: every FMap of the output of the first layer has to be convoluted with every kernel of the second layer, so we will have $(1,1,2),(1,2,2),\cdots(1,16,2),\cdots(2,1,2),\cdots(6,16,2)$. Note that the first layer only has one input FMap: the input image itself, this we only have 6 convolutions in that case. The subset NN_f corresponds to all those convolutions whose input FMap is different than the golden and thus needs to be recomputed, taking into account the errors introduced by the fault. In our case, with the type of fault chosen, we would have an error affecting only one value in the first channel. Such an error is propagated to the following layers, although it may be masked. Especially when considering permanent faults, we would be forced to simulate every convolution until the output to determine the criticality of the fault. In our setup, NN_f would have a total of 102, considering that our systolic array only computes convolutional layers. Using the analytical approach made it possible to interrupt the simulation just after the first layer. In the conducted fault injection campaign, about 76% of the total simulations were pre-determined, allowing us to simulate the systolic array only once in 76% of the total inferences. This gave us an average time per simulation $t' = x \times (0.24 \times 102 + 0.76) \approx 0.24 \times t$ and a total 86% of fault injection speed up.

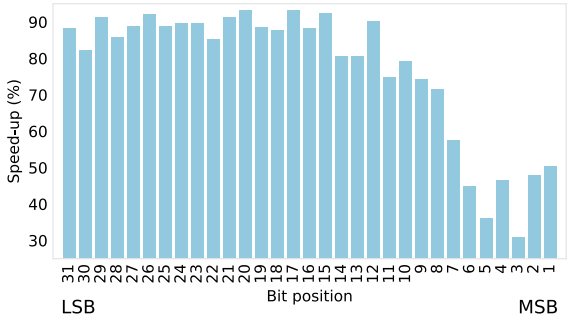


Fig. 6: Speedup per injected bit position

Figure 6 shows the speed-up in percentage concerning the injected bit position. It can be concluded that the most significant bits (0 to 7) are the ones that produce a smaller speed-up in general. Most of these faults fall in the "semi-vulnerable" category, thus it is necessary to propagate the error to infer the fault criticality. The least significant bits (14 to 31), on the other hand, provide great speed-up since the error is small enough for the simulation to be classified as "non-vulnerable".

Finally, the methodology is applied for SDC-1 (i.e., Silent Data Corruption leading to misclassification) analysis.

TABLE II: SDC1 computed over each channel. The second column (misclassified) shows how many "semi-vulnerable" simulations ended up misclassifying the output. The third column (SDC1) shows the metric computed over the whole batch.

channel	misclassified	SDC1 (%)
0	48	0.29
1	50	0.31
2	41	0.26
3	93	0.58
4	69	0.45
5	60	0.37

Using the data in table I, we could compute a lower bound corresponding to the number of "non-vulnerable" simulations. Since there is consistently no "vulnerable" simulation, the lower bound effectively corresponds to the complement of the speed-up. Furthermore, the SDC-1 was calculated for the "semi-vulnerable" faults. Table II shows the metric computed for the network under analysis. The second column shows the number of simulations that ended up in misclassification. The third column shows the metric itself. The network shows great resilience to this type of fault. In total, the SDC1 is 0.37%.

In conclusion, the presented approach is capable of significantly reducing the fault injection simulation time. This method will be extended to larger DNNs considering various fault models for obtaining reliability evaluation metrics and combining the assessment with selective hardening techniques [17], [18].

III. Mixing techniques for the Reliability Assessment of in-chip AI accelerators in GPUs

A. Motivation and Related Work

Modern GPUs are highly adopted platforms to deploy AI applications, since these devices efficiently exploit the programming flexibility and structural parallelism to speed up the execution of such data-intensive workloads [19]. In particular, modern AI algorithms are characterized by repetitive operations (e.g., convolutions) that are implemented by resorting to convolution mapping algorithms (e.g., GEMM, Winograd, or direct) [20]–[22]. For this purpose, modern generations of GPUs include specially optimized accelerators, called *Tensor Core Units* or TCUs, that resort to compacted 2D/3D arrays of *Dot-Product Units* (DPUs) [23]. TCUs comprise highly regular DPU structures to increase performance and are controlled in the GPU to exploit their implicit multi-threading parallelism [24]. Unfortunately, the vast density of transistors in modern accelerators (e.g., 100 billion) and the characterized amount of data-intensive operations in AI applications (e.g., tens of millions) impede the use of conventional strategies for the fault characterization of structures, as well as the reliability assessment of their running applications [25].

Classical strategies for GPUs, such as simulation- and emulation-based fault analyses, provide fine-grain characterizations and allow the identification of vulnerable structures under focused evaluations [26], [27]. However, evaluations on complete designs and large applications might involve unfeasible evaluation times. Other strategies, including software-based error propagation on GPUs, depend on the error models used, which might induce simplified evaluations and inaccurate analyses [28]. Thus, strategies to effectively characterize fault effects while providing an acceptable trade-off between performance and accuracy are still required.

In particular, the individual adoption of fault characterization strategies is insufficient and unfeasible for large hardware accelerators, such as GPUs with in-chip accelerators (TCUs). Some preliminary works evaluated faults in the TCU's micro-architecture and their propagation effects on the operation's outputs, considering number format impacts [29], and effects on the operand sizes [30]. However, these works did not consider the evaluation of large workloads, such as CNNs. Similarly, other works [31] evaluated the impact of TCUs in mixed-precision operations, indicating that TCUs seem to be more fault-sensitive than equivalent applications without TCUs. Unfortunately, these analyses hardly identified fault-sensitive structures inside TCUs.

Other works resort to software-based error propagation schemes to instrument the GPUs application's code and represent error corruptions in software from faults affecting the underlying system. Unfortunately, the accuracy of the method directly depends on the targeted units (mostly data path units) and the available error models to represent faults. In literature, some works [32] addressed the software-based characterization of large workloads (CNNs) in GPUs when errors affected functional units under software error models limited to random bit-flips. Authors in [33] analyzed the impact of errors in CNNs and developed error models to represent corruptions on applications, but neglecting the fine-grain micro-architecture of the underlying hardware. In [34], the authors explored a hybrid strategy to represent software errors from faults in GPU controllers. Unfortunately, their analyses were limited to a few structures with considerable evaluation times.

This work proposes a clever mixing method to effectively assess the reliability of TCUs in GPUs for permanent faults, considering the low-level micro-architecture description and their functional operation to determine error patterns, which are then used in the evaluation of large CNN workloads. Our method combines three strategies to provide an affordable trade-off between accuracy and performance evaluation.

B. Assessing the reliability of in-chip GPU accelerators

The proposed reliability assessment method for the TCUs in GPUs combines three strategies to provide accuracy while allowing feasible evaluation times when GPUs execute large applications, such as CNNs, as depicted in Figure 7. The first strategy (*focused low-level micro-architecture fault evaluation*) resorts to fine-grain evaluations of the main structures in TCUs (DPUs) in order to accurately characterize the fault propagation effects on the scalar operations in TCUs. Then, a *functional evaluation* characterizes fault effects on the array operations of TCUs, considering their interaction with other GPU structures. It must be noted that TCUs' execution depends on GPU's scheduling and are operated through GPU's machine instructions. Moreover, TCU cores are sequentially reused to operate matrix's fragments. Thus, the functional characterization allows the identification of accurate spatial fault propagation patterns during the complete execution of the TCUs. Finally, those scalar impacts and the array fault propagation patterns are combined to represent software errors effects on applications. In this case, we exploit an *application-level* error propagation strategy by using software-based error injection schemes to instrument the application's code (e.g. CNNs) with the identified error effects. The combination of both strategies (error patterns + software-based injection) allows the reliability assessment of large applications with accuracy under feasible times. The next subsections describe each strategy for the TCU's assessment.

1) Focused micro-architecture evaluation: consist of a low-level micro-architecture (Gate-level) evaluation of a DPU only focusing on its scalar operations and the fault propagation effects on the results. We use a commercial-grade logic simulator tool (*ModelSim* by *Siemens EDA*) to characterize the fault-free operation of the DPU with typical workloads (e.g., matrix tiles). The output values are used for comparison in the identification of fault effects. Then, a functional safety simulator (*ZOIX* by *Synopsis*) is adapted to exhaustively evaluate the propagation effects of permanent (stuck-at) faults in the DPU. In the characterization, one hardware fault is placed in the DPU and then all operations are independently evaluated, so allowing the identification of faults prone to corrupt individual DPU operations. We used normally distributed random matrix tiles with values in ranges from ± 1.0 to ± 100.0 to represent

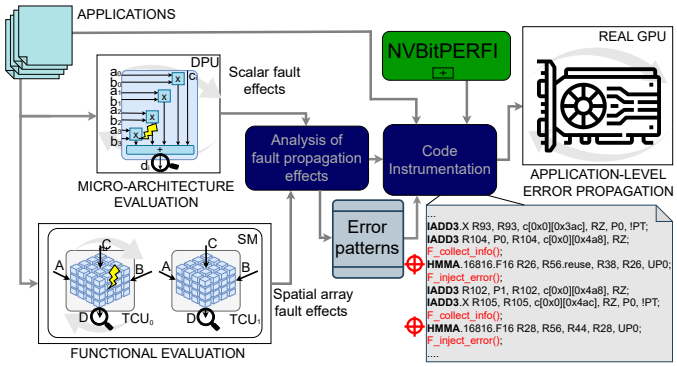


Fig. 7: A general scheme of the methodology for fault characterization of TCUs in GPUs.

CNN operations [35]. The output results are then evaluated to identify scalar corruption patterns. This analysis also allows us to identify vulnerable structures to faults for the further development of countermeasures and mitigation mechanisms.

The fault effects are categorized based on their impact on scalar results as follows: *i)* Silent Data Corruptions (SDCs) occurs when faults in the DPU alter the results. *ii)* Detected Unrecoverable Errors (DUEs) refer to faults that halt, collapse, or hang the DPU's operation, including faults generating Not-a-Number (NaN) and infinite (Inf) values, and *iii)* Masked effects encompass benign effects that leave the DPU's results unchanged despite the presence of faults.

2) Functional evaluation: consist on the characterization of the spatial corruption effects on the output arrays (matrix tiles) by faults in the TCUs. This evaluation aims to analyze the operational features in TCUs to process large matrix tiles, since TCU's are reused and managed by machine instructions. Moreover, this assessment considers the interaction with other GPU structures (i.e., schedulers and register files).

We resort to an instruction-accurate architectural simulator of the TCUs in GPUs (*PyOpenTCU*) [36] that integrates the scheduling and the memories as in real TCUs. A pin-level fault injector tool in *PyOpenTCU* supports the evaluation of hardware faults on the TCU's architecture. The injector tool places one or more faults in the inputs, outputs, or internal structures of the TCUs. Then, a complete set of input stimuli (matrix tiles) is evaluated using the set of sequential machine instructions for TCUs and the distribution of input operands among the threads and warps in the GPU. After the fault evaluation of the TCUs, the output results are compared with a fault-free operation using the same input stimuli. In our evaluation, we employ the same stimuli used in the micro-architecture evaluation from representative CNNs. The corruption effects are classified as SDCs, DUEs and masked effects.

3) Application-level error propagation: consist on the characterization of fault propagation effects on complete applications by resorting to a combined representation in software of the identified error patterns from the micro-architecture and functional evaluations of the TCUs (i.e., scalar and spatial errors patterns as *Error Masks*). Thus, *Error Masks* represent software errors by corrupting operations, and values are used to better describe the impacts of hardware faults while allowing acceptable evaluation times for large applications.

We adapt a Hardware-Injection through Program Transformation (HIPT) framework [37] (*Nvbit* [38]) to instrument the code of parallel programs for GPUs and inject software errors as permanent faults from the TCU's hardware. Our tool, called *NvBITPERFI* [34], represents errors by instrumenting the application's assembly code controlling the TCU cores (e.g., *MMA*) with custom functions that apply the previously identified scalar and spatial error patterns. The scalar error patterns are injected to corrupt output values to mimic the equivalent effect of a faulty TCU. In addition, the spatial error patterns are used to indicate the number of corrupted threads and warps, as well as the spatial distribution of the corruptions.

The error propagation evaluation starts with the execution of an error-free operation for the complete application (e.g., *CNN*) on a real GPU. The overall results are collected and stored as a reference for comparison. Then, *NvBITPERFI* instruments the application's code with one error pattern and starts its execution, collecting the output results for corruption classification. It is worth noting that the instrumented functions are used each time the TCUs are used in the application. The evaluation is repeated, injecting each one of the identified error patterns, and the error corruptions in the application are evaluated by comparing the results from the execution with errors and the reference one.

C. Experimental Results

For the experiments, we evaluated the typical configuration of a GPU with two TCUs per SM and 16 DPUs per TCU. In detail, the DPU operates four 16-bit wide inputs. The synthesis of the micro-architecture DPU resorts to a 15nm open-source technology library [39]. Furthermore, *PyOpenTCU* is configured to compute 16x16 input matrix tiles per warp, so a complete matrix tile requires a sequence of 8 *MMA* instructions.

For the fine-grain and functional fault characterizations, we used four typical input matrix tiles that account for a total of 4,096 input stimuli for the DPU core. One exhaustive fault injection campaign evaluates 82,912 faults on the DPU for each individual input stimulus. Similarly, four fault injection campaigns perform the exhaustive functional evaluation of the TCUs by injecting an overall of $2.29x10^5$ hardware faults (57,344 faults per campaign). Then, for the evaluation of the error propagation on the applications, we resort to three typical CNNs (*ResNet50*, *AlexNet*, and *MobileNetv3*) on the *ImageNet* data set by injecting 1,024 errors on the TCU core per CNN. These CNN models are deployed on TensorRT with TCU acceleration using FP16 data computations. All experiments were performed on a workstation HP Z2 G5 with an Intel Core i9-10800 CPU with 20 cores, 32BG of RAM, and one NVIDIA Ampere 3070ti GPU. In our experiments, we targeted the TCU_0 inside the SM_0 for the functional evaluation and the error propagation on a real GPU. In the end, the evaluation required around 415.6 hours, which accounts for 264hrs of fine-grain evaluation of the DPU, 128hrs of functional evaluation of the TCUs, and 23.6 hours for the error propagation on applications.

First, we analyze the fine-grain fault impacts on the scalar operations on the DPU. The results indicate that a considerable percentage of faults (around 87%) are prone to propagate and cause corruptions on the output results as SDCs, while other faults are masked (13%). Our evaluation did not identify scalar DUE effects, which can explained when considering that DPU units are used in the data path of a GPU. In addition, a cautious analysis of the results reveals that most DPU output corruptions

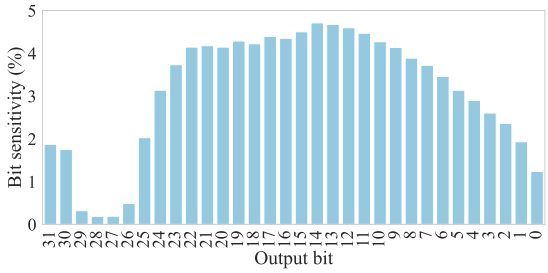


Fig. 8: Fault sensitivity on the outputs of the DPU operations.

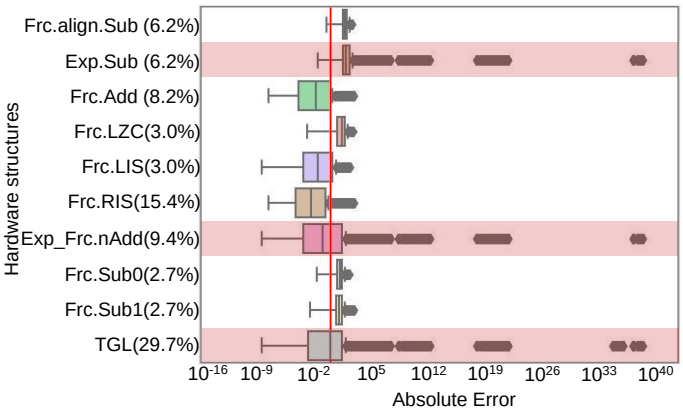


Fig. 9: Distribution of errors among the structures of the ADD inside the DPU.

mostly caused one bit-flip (around 46%), two bit-flips (about 18%) and three bit-flips (in around 8%). Then, we analyzed the scalar impacts as errors corrupting bit-fields on the DPU operations. Figure 8 illustrates the distribution of errors on the output bit-fields, indicating that most effects in the DPU operations are corruptions on the mantissa part (bits 22-13) and the lower part of the exponents (bits 30, 25-23). Interestingly, corruptions in the exponent fields (around 7% of all observed errors) are responsible for large-error magnitudes and prone to propagate effects on the application. In contrast, corruptions in the mantissa or fraction (around 93% of errors) only caused effects with magnitudes lower than 1.0 and might be neglected as critical error sources in most CNN scenarios. An additional evaluation reveals that infrastructures calculating the exponent of an operation for the sub-units of addition (*ADD* or adder tree) and multiplication (*MUL*) in the DPU (around 14.6% of the DPU's area) are highly vulnerable and prone to cause large-magnitude errors, as depicted in the distribution of errors (*Exp.Sub*, *Exp_Frc.nAdd*, and *TGL*) for the DPU's ADD in Figure 9. We observed that for both structures (ADD and MUL) produced similar error distributions with around 48% of the corruptions caused by the ADD and around 13% produced by faults inside any of the 4 MUL cores in the DPU.

The functional evaluation aims to indicate the distribution of spatial corruption effects among the DPUs in the TCU. Figure 10 reports the error patterns and their spatial distribution for an output matrix tile (16x16). The results indicate that a faulty DPU in a TCU can corrupt from 1 to 8 output elements. The spatial distribution of errors in terms of relative distance between corrupted elements, and the localized effects (i.e., left-part of the output) is explained by a deterministic distribution of matrix fragments to operate among the TCUs and its scheduling

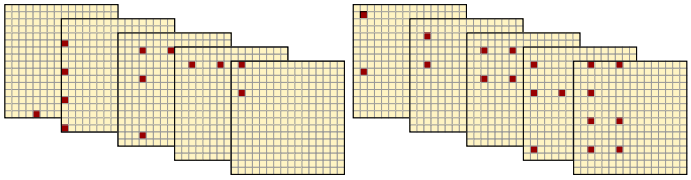


Fig. 10: Distribution of output's corruptions by a faulty TCU.

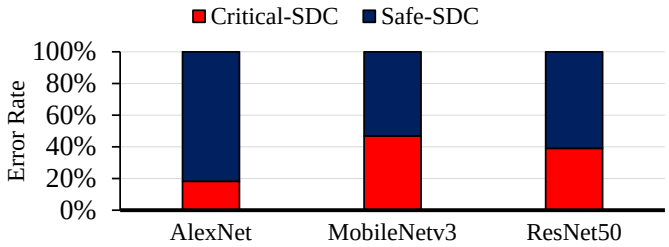


Fig. 11: Corruption effects on the CNNs by TCU hardware faults.

policy, respectively. An analysis of the intermediate results (during the *MMA* instruction's execution) reveals that the sequential use of TCUs contributes to accumulate errors and increases in one the number of corrupted bits on the output value.

Finally, both sets of error patterns (*scalar* and *spatial*) are used to instrument the applications. This evaluation considers one error injected per application execution. Thus, an error (i.e., error mask) is only applied on those threads using the instructions to access a targeted unit (TCU_0). Figure 11 illustrates the fault rate on the outputs of the evaluated CNNs. The results indicate that all the application level errors induce Silent Data Corruption (SDC) effects, but a considerable percentage of error effects (from 18% to 45%) critically caused wrong outcomes at the CNN's outputs. These results suggest that accumulated errors from the reuse of faulty TCUs are highly prone to corrupt several operations in CNNs and propagate their effects, so increasing the probability of data corruption on the outputs. Since errors affect several elements in the layer's outputs, the implicit resilience of a CNN is affected by these corruption effects. In fact, we identified that during the inference of one image, around 25% of the kernels are reused, and from those kernels, around 80% rely on the TCU computations. These features of the CNN implementation cause that a faulty TCU corrupt multiple layers by the accumulation of error during the CNN operation. Indeed, the results show that the average error activation rate reaches ≈90,000, ≈ 300,000, and ≈1 million for *AlexNet*, *ResNet50*, and *MobileNet*, respectively.

An analysis of the workload's distribution in the GPU shows that each kernel uses fixed configurations to distribute their tasks among the SMs and use the TCUs. The workload distribution in the GPU influences the error propagation when a faulty TCU is in the system, which indicates that a clear understanding of micro-architecture and the algorithms (i.e., convolutions) used to map CNNs in the GPU's TCUs are required to analyze and develop effective countermeasures and mitigation mechanisms correctly. The other parameter contributing to the error propagation in CNNs is the total amount of SMs; since the CNNs are distributed among all TCUs in a system, their error vulnerability is proportionally inverse to the number of TCUs and SMs in a system.

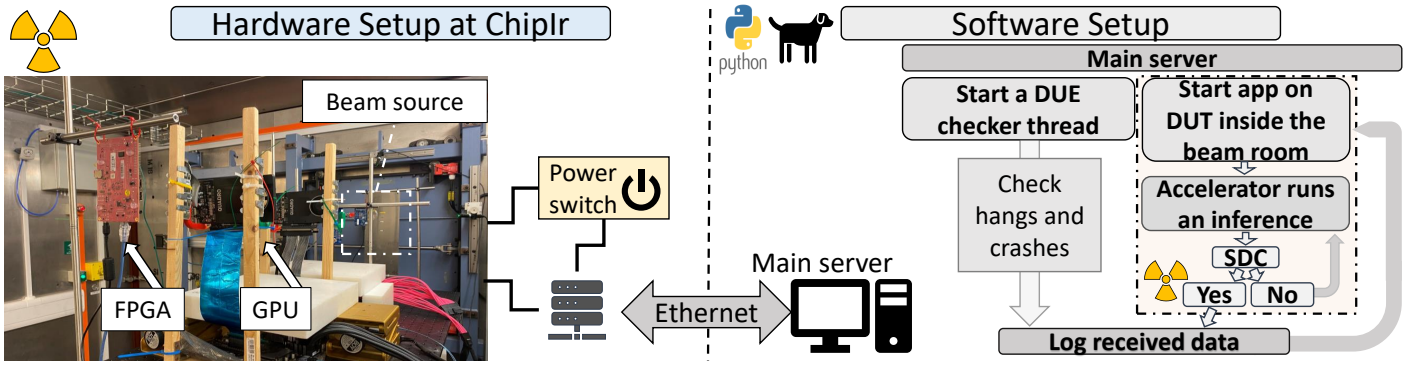


Fig. 12: Beam setup mounted at ChipIR beam line and the software setup.

IV. Reliability Assessment of DNN Hardware Accelerators Through Physical Fault Injection

A. Motivation and Related Work

Evaluating the sensitivity of DNN accelerators for transient events is crucial for identifying errors that compromise DNN accuracy. Fault simulation allows researchers to pinpoint specific fault sites and track how faults propagate through both the hardware and the application [37], [40], [41]. This can be implemented at various levels of abstraction. Low-level hardware fault simulations (e.g., gate level) offer detailed insights but are time-consuming, while higher-level hardware or software simulations (e.g., microarchitectural or software) provide faster but less accurate results. Additionally, selecting appropriate fault models for simulation is critical to prevent inaccurate conclusions [42]. In contrast, physical fault injection using radiation exposes the system to a beam of ionizing particles, offering a means to estimate realistic error rates. However, this method does not support tracking fault propagation, as faults are identified only when they lead to failures [43], [44].

By performing radiation experiments, recent studies have demonstrated that DNNs accelerators are highly susceptible to transient faults induced by radiation [31], [45]–[47]. Recent research data suggests that while GPUs are mostly affected because of the high amount of available resources [48], [49] and the possibility of having multiple output elements corrupted, which undermines DNN reliability [46], [50], FPGAs, are mostly affected due to the programable hardware characteristics, where a transient fault can change the configuration memory and change the circuit [51]–[53].

In this section, we show how we can extract practical – and non-obvious – information from neutron beam experiments with GPUs and FPGAs running DNNs.

B. Neutron Beam Experiments

Our experiments were performed at the ChipIR facility within the Rutherford Appleton Laboratory in the UK. The setup within the ChipIR facility is depicted in Figure 12. ChipIR delivers a neutron beam with an energy spectrum similar to atmospheric neutrons [54]. The neutron flux available reaches approximately $3.5\times10^6 n/(cm^2/s)$, which is nearly eight orders of magnitude greater than the terrestrial flux at sea level (13 $neutrons/(cm^2 \cdot h)$ [55]).

Considering the low terrestrial neutron flux, encountering more than one corruption during program execution in a realistic application is unlikely. Our experiments are meticulously designed to uphold this characteristic, ensuring observed error rates remain below one failure per 2,000 executions. Each DNN configuration undergoes testing for a minimum of 3 effective hours, excluding setup, result checking, initialization, and recovery from DUE time.

1) System Under Test: We selected the TUL PYNQ-Z2 board based on the 28nm Xilinx Zynq-7000 SoC and the NVIDIA Tesla V100 GPU based on a 12nm Volta microarchitecture as hardware platforms.

For the GPU experiments, we target an evaluation to measure the reliability of multiple floating-point precisions of a General Matrix Multiplication (GEMM), such as FP16, FP32, and FP64 from the cuBLAS libraries [56]. The choice of multiple GEMM configurations is guided as they are the core of the state-of-the-art DNNs. As a study case, we also exposed an object detection DNN, YOLOv3 [57], with two precisions, FP16 and FP32, with the same setup as used in [58].

For the FPGA experiments, we created five ANN accelerators using the HLS4ML tool [59]. HLS4ML is an open-source tool that can translate machine learning models described with libraries such as Pytorch and TensorFlow into HLS-compatible high-level-description parametric models, which can be configured, synthesized, and implemented to run on an FPGA. We used a 2-layer ANN to classify handwritten digits on the MNIST dataset and varied the number of multipliers used per layer. Table III shows the number of multipliers used for each evaluated FPGA configuration. As the number of multipliers increases, the clock cycles decrease, but the area also increases. As discussed in the next section, a larger area may increase the failure rate.

2) Experimental Setup: Figure 12 presents an overview of the experimental setup. We developed a software watchdog composed of Python scripts, which operates on the server computer located externally to the beam room. This watchdog oversees the DUT by monitoring its activity, executing programs,

TABLE III: Configurations generated by HLS4ML by varying the number of multipliers used in each layer. 39,200 multiplications are performed in the first layer and 500 in the second.

	Multipliers Used (layer 1× layer 2)	Clock Cycles
Opt_1	1x1	39,804
Opt_2	2x2	19,960
Opt_3	10x5	4,154
Opt_4	50x20	947

logging events, and effectively recovering from any device malfunctions. When the program becomes unresponsive, it is terminated and restarted based on a predefined timeout period. Notably, the timeout duration for each code is meticulously set, extending up to 10× the anticipated execution time, tailored to the specific requirements of the code's execution duration.

For every DUT configuration, we designate a host system that is responsible for managing the DNN accelerator hardware. For the GPU setup, the host system is an x86 Intel i3 CPU, whereas for FPGA SoC configurations, the ARM CPU on the PYNQ SoC is employed. Following each iteration performed within the device, the host compares the kernel's output against a predefined constant golden value. Any discrepancy between the actual output and the golden value prompts the host to relay pertinent information back to the server outside the beam room, before initiating a restart of the process. It is important to emphasize that our error rate analysis only considers discrepancies observed in the kernel output.

The experiments conducted at ChipIR allow the estimation of the *Failure In Time (FIT)* rates, representing the error rate of a given accelerator. FIT rate is directly derived from the application *cross-section* (σ), calculated by dividing the number of observed errors by the received particle fluence ($neutrons/cm^2$). This cross-section is then multiplied by the $flux$ under which the system operates to yield the FIT rate.

While the FIT rate is valuable for estimating system failure rates, it overlooks the execution time variations stemming from different HLS and GPU hardware optimizations. To address this, we compute the *Mean Executions Between Failures (MEBF)* to gauge how many correct executions can be performed before a failure occurs, as defined by equation 1.

$$MEBF = \frac{MTBF}{ExecutionTime} \tag{1}$$

Where the $MTBF$ is the inverse of the cross-section multiplied by the $flux$ under which the device will operate (i.e., $MTBF = FIT^{-1} = (\sigma * flux)^{-1}$), and the $ExecutionTime$ is the application execution time.

C. Experimental Results

Figure 13 shows experimentally measured FIT rate and MEBF. Both FIT and MEBF are presented relative to the configuration exhibiting the worst performance for each code. Specifically, we present the experimental data relative to FP64 GEMM and FP32 YOLOv3 for the GPU and Opt_1 for the FPGA. The data in Figure 13 is presented with a 95% confidence interval on a Poisson distribution.

1) FIT: Smaller precision results in a lower overall FIT rate for both applications evaluated on GPUs (Figure 13a). Specifically, the FIT rate of FP32 is 21.5% lower than that of FP64, while the FIT rate of FP16 FIT is 38.43% lower than that of FP16. Similarly, for YOLOv3, the FIT rate of FP16 is 46.29% lower than that of FP32. Using smaller float precisions reduces the probability of a fault that leads to a failure (SDC or DUE) as it uses fewer hardware resources. Furthermore, compared to the FP32 version, the FP16 version of YOLOv3 results in fewer critical SDCs (misdetections) with 21.43% and 27.27%, respectively. The lower representation capacity of FP16 reduces the chances of very large corrupted values, which are known to impact the reliability of DNNs [45], [60], thus changing the detection result of YOLOv3.

The FPGA FIT rate exhibits different behavior compared to the GPU results. The Opt_2 FIT rate decreases compared to Opt_1, while the same does not happen for Opt_3 and Opt_4. Specifically, the FIT rate for Opt_2 is 82.81% of Opt_1, while the FIT rates for Opt_3 and Opt_4 are 98.35% and 107.74% of Opt_1 FIT rate, respectively. It is important to note that no built-in fault detection functionalities, such as scrubbing and/or the activation ECC, were utilized in any of the experiments. As a result, smaller accelerators running for longer might accumulate errors in the extensively reused hardware, and since they experience less DUE stopping the execution, faults can propagate to outputs, leading to more SDCs. Additionally, the DUE FIT increases for larger accelerators. The larger the accelerator area (i.e., more multipliers per layer), the higher the chance that a bit flip corrupts a critical circuit resource, leading to the application's hang or crash.

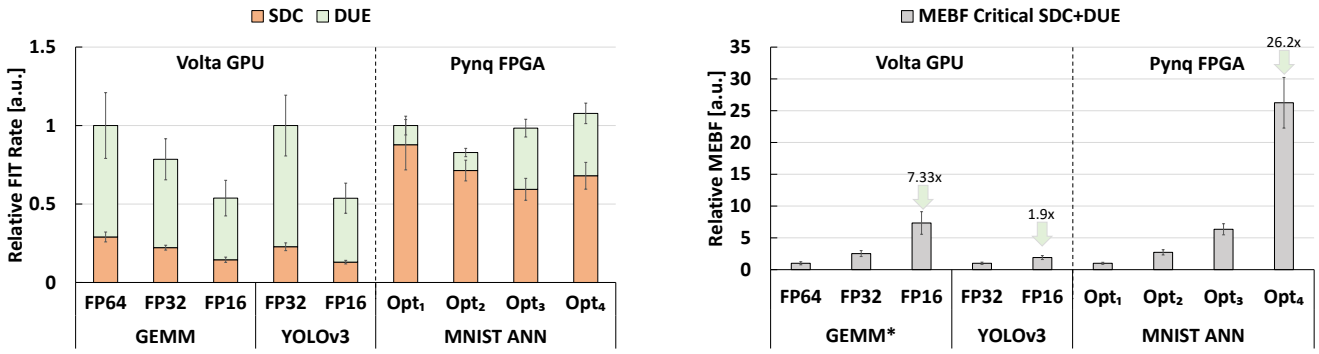
The Critical SDCs for the FPGAs experiments correspond to 18.26%, 21.92%, 28.52%, and 28.81% of the SDC FIT for Opt_1, Opt_2, Opt_3, and Opt_4, respectively. The chance of experiencing critical SDCs is higher for larger accelerators than smaller ones due to the usage of hardware resources. The amount of work done remains constant, but larger accelerators have more area to improve performance, which leads to a higher probability of critical computation at any given time.

2) MEBF: When comparing the ANN accelerators, relying on the FIT rate alone is insufficient, as it does not provide information on the application's execution time [61]. Therefore, we must correlate error rates with performance to make the comparison more holistic.

Figure 13b shows the MEBF for the different configurations we evaluate. For the GEMM codes on GPU, we used all SDCs and DUEs to calculate the MEBF, while for YOLOv3 and MNIST ANN, we used the Critical SDCs (misclassification or misdetection) and the DUE to calculate the MEBF. By using only the Critical SDCs (+DUEs) to calculate the MEBF, we can quantify the failures that will actually negatively affect the accelerator utilization (i.e., crash or wrong classification) if no fault mitigation is adopted.

Unsurprisingly, the GPU MEBF always increases when comparing the larger precisions GEMM FP64 and YOLOv3 FP32 to the lower precisions versions. For instance, the GPU MEBF for GEMM is 2.52× higher for FP32 and 7.33× higher for FP16 than FP64. Also, the YOLOv3 FP16 produces 1.89× more correct predictions than the FP32 version. Optimized hardware not only delivers higher performance but also ensures more correct operations than slower hardware.

The results of the FPGA configurations show a significant improvement trend from Opt_1 to Opt_4, in contrast to the FIT rate results. The MEBF demonstrates a linear correlation with the reuse parameters, and compared to Opt_1, it increases by 2.72× for Opt_2, 6.35× for Opt_3, and 26.25× for Opt_4. In general, the data suggests that fast and large accelerators are capable of performing more inferences between two failures than slow and small ones. However, when they fail, they are more likely to experience a critical SDC.



(a) (b)

Fig. 13: Experimentally measured relative SDC and DUE Failure In Time (FIT) rates, as well as the relative Mean Executions Between Failure (MEBF). FIT and MEBF values are expressed relative to the configuration exhibiting the worst performance. *While for the GEMM code, we consider all the SDCs on the MEBF calculation, in the YOLOv3 and MNIST ANN, we consider only the SDCs classified as a misclassification or misdetection.

V. Conclusions

The paper presents advanced methods to assess the reliability of the three types of DNN-HAs, i.e., Systolic Array, GPU and FPGA, through different tailored methodologies. These are a) hybrid analytical and hierarchical FI-based reliability assessment for systolic-array-based DNN accelerators; b) mixing techniques for the reliability assessment of in-chip AI accelerators in GPUs; c) reliability assessment of DNN hardware accelerators through physical fault injection. The experimental results demonstrated the efficiency of the proposed methods applied to their target DNN HW accelerator platforms.

Acknowledgments

This work was supported in part by the Estonian Research Council grant PUT PRG1467 “CRASHLES”, by Estonian-French PARROT project “EnTrustED”, by the ANR-21-CE24-0015 “RE-TRUSTING” project, and by the National Resilience and Recovery Plan (PNRR) through the National Center for HPC, Big Data and Quantum Computing.

References

[1] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, “Efficient processing of deep neural networks: A tutorial and survey,” *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.

[2] M. Taheri, “Dnn hardware reliability assessment and enhancement,” in *27th IEEE European Test Symposium (ETS)*, 2022.

[3] *Artificial Intelligence and Hardware Accelerators*. Springer International Publishing, 2023. [Online]. Available: http://dx.doi.org/10.1007/978-3-031-22170-5

[4] A. Nardi and A. Armato, “Functional safety methodologies for automotive applications,” in *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2017, pp. 970–975.

[5] M. Jenihhin, M. S. Reorda, A. Balakrishnan, and D. Alexandrescu, “Challenges of reliability assessment and enhancement in autonomous systems,” in *2019 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, 2019, pp. 1–6.

[6] M. Shafique, M. Naseer, T. Theocharides, C. Kyrkou, O. Mutlu, L. Orosa, and J. Choi, “Robust machine learning systems: Challenges, current trends, perspectives, and the road ahead,” *IEEE Design & Test*, vol. 37, no. 2, pp. 30–57, 2020.

[7] A. Bosio, P. Bernardi, A. Ruospo, and E. Sanchez, “A reliability analysis of a deep neural network,” in *2019 IEEE Latin American Test Symposium (LATS)*. IEEE, 2019, pp. 1–6.

[8] M. Taheri, N. Cherezova, M. S. Ansari, M. Jenihhin, A. Mahani, M. Daneshtalab, and J. Raik, “Exploration of activation fault reliability in quantized systolic array-based dnn accelerators,” *arXiv preprint arXiv:2401.09509*, 2024.

[9] M. H. Ahmadilivani, M. Barbareschi, S. Barone, A. Bosio, M. Daneshtalab, S. D. Torca, G. Gavarini, M. Jenihhin, J. Raik, A. Ruospo, E. Sanchez, and M. Taheri, “Special session: Approximation and fault resiliency of dnn accelerators,” in *2023 IEEE 41st VLSI Test Symposium (VTS)*, 2023, pp. 1–10.

[10] M. Taheri, M. Riazati, M. H. Ahmadilivani, M. Jenihhin, M. Daneshtalab, J. Raik, M. Sjõdin, and B. Lisper, “Deepaxe: A framework for exploration of approximation and reliability trade-offs in dnn accelerators,” in *24th International Symposium on Quality Electronic Design*. https://doi.org/10.48550/arXiv.2303.08226, 2023.

[11] M. Taheri *et al.*, “Appraiser: Dnn fault resilience analysis employing approximation errors,” in *2023 26th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS)*, 2023, pp. 124–127.

[12] M. H. Ahmadilivani *et al.*, “A systematic literature review on hardware reliability assessment methods for deep neural networks,” *ACM Computing Surveys*, vol. 56, no. 6, pp. 1–39, 2024.

[13] A. Ruospo *et al.*, “A survey on deep learning resilience assessment methodologies,” *Computer*, vol. 56, no. 2, pp. 57–66, 2023.

[14] P. Quinton, “Automatic synthesis of systolic arrays from uniform recurrent equations,” *ACM SIGARCH Computer architecture news*, vol. 12, no. 3, pp. 208–214, 1984.

[15] M. Taheri, M. Daneshtalab, J. Raik, M. Jenihhin, S. Pappalardo, P. Jimenez, B. Deveautour, and A. Bosio, “Saffira: a framework for assessing the reliability of systolic-array-based dnn accelerators,” 2024.

[16] M. H. Ahmadilivani, M. Taheri, J. Raik, M. Daneshtalab, and M. Jenihhin, “Deepvigor: Vulnerability value ranges and factors for dnns’ reliability assessment,” in *2023 IEEE European Test Symposium (ETS)*. IEEE, 2023, pp. 1–6.

[17] M. Taheri, N. Cherezova, S. Nazari, A. Rafiq, A. Azarpeyvand, T. Ghasempouri, M. Daneshtalab, J. Raik, and M. Jenihhin, “Adam: Adaptive fault-tolerant approximate multiplier for edge dnn accelerators,” *29th IEEE European Test Symposium, arXiv:2403.02936*, 2024.

[18] M. H. Ahmadilivani, M. Taheri, J. Raik, M. Daneshtalab, and M. Jenihhin, “Enhancing fault resilience of qnns by selective neuron splitting,” in *2023 IEEE 5th International Conference on Artificial Intelligence Circuits and Systems (AICAS)*. IEEE, 2023, pp. 1–5.

[19] B. Dally, “Hardware for deep learning,” in *IEEE Hot Chips 35 Symposium (HCS)*. IEEE Computer Society, 2023, pp. 1–58.

[20] X. Liu, J. Pool, S. Han, and W. J. Dally, “Efficient sparse-winograd convolutional neural networks,” *CoRR*, vol. abs/1802.06367, 2018.

[21] A. Vasudevan, A. Anderson, and D. Gregg, “Parallel multi channel convolution using general matrix multiplication,” in *2017 IEEE 28th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, 2017, pp. 19–24.

[22] A. Lavin and S. Gray, “Fast algorithms for convolutional neural networks,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[23] B. R. Boswell *et al.*, “Generalized acceleration of matrix multiply accumulate operations,” Jul. 2 2019, u.S. Patent 10,338,919.

[24] M. Raihan *et al.*, “Modeling deep learning accelerator enabled gpus,” in *IEEE Int. Symp. on Performance Analysis of Systems and Software (ISPASS)*, mar 2019, pp. 79–92.

[25] J. E. R. Condia, J.-D. Guerrero-Balaguera, F. F. Dos Santos, M. S. Reorda, and P. Rech, "A multi-level approach to evaluate the impact of gpu permanent faults on cnn's reliability," in *2022 IEEE International Test Conference (ITC)*, 2022, pp. 278–287.

[26] J. E. R. Condia, F. F. dos Santos, M. S. Reorda, and P. Rech, "Combining architectural simulation and software fault injection for a fast and accurate cnns reliability evaluation on gpus," in *2021 IEEE 39th VLSI Test Symposium (VTS)*, 2021, pp. 1–7.

[27] F. F. d. Santos, J. E. R. Condia, L. Carro, M. S. Reorda, and P. Rech, "Revealing gpus vulnerabilities by combining register-transfer and software-level fault injection," in *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2021, pp. 292–304.

[28] G. Papadimitriou and D. Gizopoulos, "Demystifying the system vulnerability stack: Transient fault effects across the layers," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, 2021, pp. 902–915.

[29] R. L. Sierra, J.-D. Guerrero-Balaguera, J. E. Rodriguez Condia, and M. Sonza Reorda, "Analyzing the impact of different real number formats on the structural reliability of tcus in gpus," in *2023 IFIP/IEEE 31st International Conference on Very Large Scale Integration (VLSI-SoC)*, 2023, pp. 1–6.

[30] R. Limas Sierra, J.-D. Guerrero-Balaguera, J. E. R. Condia, and M. Sonza Reorda, "Exploring hardware fault impacts on different real number representations of the structural resilience of tcus in gpus," *Electronics*, vol. 13, no. 3, 2024.

[31] P. M. Basso, F. F. d. Santos, and P. Rech, "Impact of tensor cores and mixed precision on the reliability of matrix multiplication in gpus," *IEEE Transactions on Nuclear Science*, vol. 67, no. 7, pp. 1560–1565, 2020.

[32] J.-D. Guerrero-Balaguera, R. L. Sierra, and M. S. Reorda, "Effective fault simulation of gpu's permanent faults for reliability estimation of cnns," in *2022 IEEE 28th International Symposium on On-Line Testing and Robust System Design (IOLTS)*, 2022, pp. 1–6.

[33] C. Bolchini, L. Cassano, A. Miele, and A. Toschi, "Fast and accurate error simulation for cnns against soft errors," *IEEE Transactions on Computers*, vol. 72, no. 4, pp. 984–997, 2023.

[34] J. D. Guerrero Balaguera, J. E. R. Condia, F. Fernandes Dos Santos, M. Sonza Reorda, and P. Rech, "Understanding the effects of permanent faults in gpu's parallelism management and control units," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '23, 2023.

[35] D. Mallasén, R. Murillo, A. A. D. Barrio, G. Botella, L. Piñuel, and M. Prieto-Matias, "Percival: Open-source posit risc-v core with quire capability," *IEEE Transactions on Emerging Topics in Computing*, vol. 10, no. 3, pp. 1241–1252, 2022.

[36] R. L. Sierra, J.-D. Guerrero-Balaguera, J. E. R. Condia, and M. Sonza Reorda, "Analyzing the impact of different real number formats on the structural reliability of tcus in gpus," in *2023 IFIP/IEEE 31st International Conference on Very Large Scale Integration (VLSI-SoC)*, 2023, pp. 1–6.

[37] T. Tsai, S. K. S. Hari, M. Sullivan, O. Villa, and S. W. Keckler, "Nvbitfi: Dynamic fault injection for gpus," in *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2021, pp. 284–291.

[38] O. Villa, M. Stephenson, D. Nellans, and S. W. Keckler, "Nvbit: A dynamic binary instrumentation framework for nvidia gpus," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '52, 2019, p. 372–383.

[39] M. Martins, J. M. Matos, R. P. Ribas, A. Reis, G. Schlinker, L. Rech, and J. Michelsen, "Open cell library in 15nm freepdk technology," in *Proceedings of the 2015 Symposium on International Symposium on Physical Design*, 2015, p. 171–178.

[40] J. E. R. Condia, B. Du, M. S. Reorda, and L. Sterpone, "Flexgripplus: An improved gpgpu model to support reliability analysis," *Microelectronics Reliability*, vol. 109, p. 113660, 2020.

[41] A. Mahmoud *et al.*, "Pytorchfi: A runtime perturbation tool for dnns," in *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*. IEEE, 2020, pp. 25–31.

[42] G. Papadimitriou and D. Gizopoulos, "Demystifying the System Vulnerability Stack: Transient Fault Effects across the Layers," in *IEEE ISCA*, 2021, p. 902–915. [Online]. Available: https://doi.org/10.1109/ISCA52012.2021.00075

[43] H. Quinn, "Challenges in testing complex systems," *IEEE Transactions on Nuclear Science*, vol. 61, no. 2, pp. 766–786, 2014.

[44] P. R. Bodmann, G. Papadimitriou, R. L. R. Junior, D. Gizopoulos, and P. Rech, "Soft error effects on arm microprocessors: Early estimations versus chip measurements," *IEEE Transactions on Computers*, vol. 71, no. 10, pp. 2358–2369, 2021.

[45] G. Li, S. K. S. Hari, M. Sullivan, T. Tsai, K. Pattabiraman, J. Emer, and S. W. Keckler, "Understanding error propagation in deep learning neural network (dnn) accelerators and applications," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2017, pp. 1–12.

[46] Y. Ibrahim, H. Wang, J. Liu, J. Wei, L. Chen, P. Rech, K. Adam, and G. Guo, "Soft errors in dnn accelerators: A comprehensive review," *Microelectronics Reliability*, vol. 115, p. 113969, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0026271420308003

[47] R. L. Rech Junior, S. Malde, C. Cazzaniga, M. Kastriotou, M. Letiche, C. Frost, and P. Rech, "High energy and thermal neutron sensitivity of google tensor processing units," *IEEE Transactions on Nuclear Science*, vol. 69, no. 3, pp. 567–575, 2022.

[48] D. Tiwari, S. Gupta, J. Rogers, D. Maxwell, P. Rech, S. Vazhkudai, D. Oliveira, D. Londo, N. DeBardeleben, P. Navaux, L. Carro, and A. Bland, "Understanding gpu errors on large-scale hpc systems and the implications for system design and operation," in *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, 2015, pp. 331–342.

[49] M. B. Sullivan, N. Saxena, M. O'Connor, D. Lee, P. Racunas, S. Hukerikar, T. Tsai, S. K. S. Hari, and S. W. Keckler, *Characterizing And Mitigating Soft Errors in GPU DRAM*, 2021, p. 641–653. [Online]. Available: https://doi.org/10.1145/3466752.3480111

[50] F. F. dos Santos, P. F. Pimenta, C. Lunardi, L. Draghetti, L. Carro, D. Kaeli, and P. Rech, "Analyzing and increasing the reliability of convolutional neural networks on gpus," *IEEE Transactions on Reliability*, vol. 68, no. 2, pp. 663–677, 2018.

[51] F. Libano, B. Wilson, J. Anderson, M. J. Wirthlin, C. Cazzaniga, C. Frost, and P. Rech, "Selective hardening for neural networks in fpgas," *IEEE Transactions on Nuclear Science*, vol. 66, no. 1, pp. 216–222, 2018.

[52] P. Maillard, Y. P. Chen, J. Vidmar, N. Fraser, G. Gambardella, M. Sawant, and M. L. Voogel, "Radiation-Tolerant Deep Learning Processor Unit (DPU)-Based Platform Using Xilinx 20-nm Kintex UltraScale FPGA," *IEEE Transactions on Nuclear Science*, vol. 70, no. 4, pp. 714–721, October 2023.

[53] M. Traiola, F. F. Dos Santos, P. Rech, C. Cazzaniga, O. Sentieys, and A. Kritikakou, "Impact of high-level-synthesis on reliability of artificial neural network hardware accelerators," *IEEE Transactions on Nuclear Science*, pp. 1–1, 2024.

[54] C. Cazzaniga and C. D. Frost, "Progress of the scientific commissioning of a fast neutron beamline for chip irradiation," *Journal of Physics: Conference Series*, vol. 1021, p. 012037, may 2018. [Online]. Available: https://doi.org/10.1088/1742-6596/1021/1/012037

[55] JEDEC, "Measurement and Reporting of Alpha Particle and Terrestrial Cosmic Ray-Induced Soft Errors in Semiconductor Devices," JEDEC Standard, Tech. Rep. JESD89A, 2006.

[56] C. Nvidia, "Cublas library," *NVIDIA Corporation, Santa Clara, California*, vol. 15, no. 27, p. 31, 2008.

[57] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *CoRR*, vol. abs/1804.02767, 2018. [Online]. Available: http://arxiv.org/abs/1804.02767

[58] F. Fernandes dos Santos, C. Lunardi, D. Oliveira, F. Libano, and P. Rech, "Reliability evaluation of mixed-precision architectures," in *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2019, pp. 238–249.

[59] J. Duarte *et al.*, "Fast Inference of Deep Neural Networks for Real-Time Particle Physics Applications," *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 305–335, February 2019.

[60] L. Roquet, F. Fernandes dos Santos, P. Rech, M. Traiola, O. Sentieys, and A. Kritikakou, "Cross-Layer Reliability Evaluation and Efficient Hardening of Large Vision Transformers Models," in *Design Automation Conference (DAC)*, San Fracisco, United States, Jun. 2024. [Online]. Available: https://hal.science/hal-04456702

[61] G. Reis, J. Chang, N. Vachharajani, S. Mukherjee, R. Rangan, and D. August, "Design and evaluation of hybrid fault-detection systems," in *32nd International Symposium on Computer Architecture (ISCA'05)*, 2005, pp. 148–159.