SAFFIRA A Framework for Assessing the Reliability of Systolic-Array DNN Accelerators

Journal of Circuits, Systems, and Computers

SAFFIRA
A Framework for Assessing the Reliability of Systolic-Array DNN Accelerators

Salvatore Pappalardo[1], Nicolò Bellarmino[2], Bastien Deveautour[3], Alberto Bosio[1]
[1]*Ecole Centrale de Lyon, INSA Lyon, CNRS,*
Universite Claude Bernard Lyon 1, CPE Lyon, INL, UMR5270, 69130 Ecully, France
[2]*Politecnico di Torino, Italy*
[3]*Nantes Université, France*
[1]{*name.surname*}*@ec-lyon.fr,* [2]*nicolo.bellarmino@polito.it* [3]*bastien.deveautour@univ-nantes.fr,*

Mahdi Taheri[4], Masoud Daneshtalab[4], Jaan Raik[4,5], Maksim Jenihhin[4]
[4]*Tallinn University of Technology, Tallinn, Estonia*
[5]*Mälardalen University, Västerås, Sweden*

Received (Day Month Year)
Revised (Day Month Year)
Accepted (Day Month Year)
Published (Day Month Year)

Systolic array has emerged as a prominent architecture for Deep Neural Network (DNN) hardware accelerators, providing high-throughput and low-latency performance essential for deploying DNNs across diverse applications. However, when used in safety-critical applications, reliability assessment is mandatory to guarantee the correct behavior of DNN accelerators. 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 addresses the time efficiency issue by introducing a novel hierarchical software-based hardware-aware fault injection strategy tailored for systolic array-based DNN accelerators. A uniform Recurrent Equations system is used for software modeling of the systolic-array core of the DNN accelerators. The approach demonstrates a reduction of the fault injection time up to 3× compared to the state-of-the-art hybrid (software/hardware) hardware-aware fault injection frameworks and more than 2000× compared to RT-level fault injection frameworks — without compromising the accuracy from the application level. Additionally, we introduce novel reliability metrics to better evaluate the robustness of a deep neural network system. The performance of the framework is studied on state-of-the-art DNN benchmarks.

Keywords: Hardware Accelerator; Systolic Array; Deep Neural Networks; Fault Simulation; Reliability; Resilience Assessment; Reliability Metrics.

1. Introduction

Evaluating the reliability of a Deep Neural Network (Deep Neural Network (DNN)) is a complex task, as its robustness is depends on multiple factors, such as the training dataset, the data type, and the quality of the test set[1]. Furthermore, it is essen-

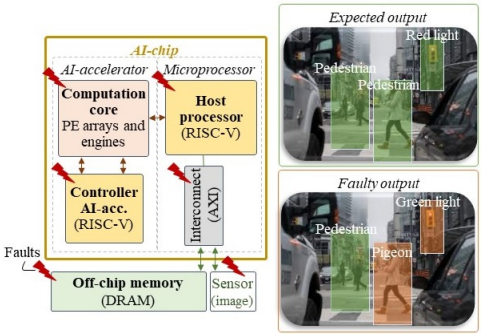


Fig. 1: DNN accelerator hardware reliability threats[5]

tial to account for the hardware executing the computations [2], as different platforms are prone to distinct types of faults[3]. Numerous studies have demonstrated that hardware faults can significantly compromise the performance of DNNs[4]. Consequently, there has been a growing research focus on assessing and improving the reliability of DNNs. An example of faulty hardware is illustrated in Figure 1, which highlights potential fault locations within a DNN inference engine. This example emphasizes the importance of reliability assessment for DNNs.

However, evaluating the reliability of DNNs remains a challenging task[6], often addressed through Fault Injection (FI). There are three primary FI methodologies for assessing DNN reliability: irradiation-based, platform-based, and simulation-based approaches [7]. Among these, simulation-based FI is the most widely used within the research community, as it is less resource-intensive and more cost-effective compared to the others[2]. Simulation-based FI offers some advantages, such as the ability to precisely model various fault scenarios to evaluate their impact on Deep Neural Networks (DNN), eliminating the need for extensive hardware resources and lengthy design processes. Moreover, it provides full control over network parameters and architecture.

On the other hand, simulating DNN hardware accelerators for FI is computationally intensive and often requires considerable time to execute a single inference[8]. To address this, this paper presents a novel simulation flow and FI methodology designed to significantly expedite the injection process on systolic-array-based DNN hardware accelerators. The systolic-array core of these accelerators is modeled using a Uniform Recurrent Equations (Uniform Recurrent Equations (URE)) framework. The proposed injection flow has been implemented as an open-source tool named **SAFFIRA**, which stands for **Systolic Array simulator Framework for Fault Injection based Reliability Assessment**. While simulation-based FI is typically performed either without accounting for the underlying hardware or using RTL (Register-Transfer Level) simulations, which are notoriously resource-intensive and time-consuming, SAFFIRA leverages a Systolic Array (SA) simula-

tor, providing a balance between precision and efficiency—offering greater accuracy than hardware-agnostic tools while being significantly faster than traditional RTL-level simulations. Experimental results demonstrate a reduction in FI time of up to 3× compared to state-of-the-art hybrid (software/hardware) hardware-aware fault injection frameworks, and up to 2000× compared to RTL-level fault injection frameworks—without sacrificing accuracy in evaluating DNN and Hardware reliability.

The key contributions of this paper are as follows:

- Introducing a hierarchical **methodology** for hardware-accurate reliability assessment of SA through a novel simulation-based FI approach, by modeling systolic arrays using the URE system;
- Presenting an **open-source tool** that implements the aforementioned methodology;
- Introducing a new **metrics** for reliability assessments of DNNs;
- Evaluating the performance of the framework on state-of-the-art DNN benchmarks.

The rest of this paper is organized as follows. Section 2 presents the related works. Section 3 presents the proposed fault injection flow for SA. Section 5 shows the experimental setup and results. Section 7 concludes the paper.

2. Related Works

This section discusses previous works targeting DNNs reliability assessment by using simulation-based FI.

2.1. *Hardware-Agnostic FI Tools*

Hardware-agnostic FI tools work without considering the underlying hardware. Some of these tools are capable of directly injecting faults into DNNs models. For instance, PyTorchFI[9] and TensorFI[10] can inject faults into DNN models implemented in PyTorch, TensorFlow, and Keras. These open-source frameworks can inject both permanent and transient faults into weights and activations, with specified error rates, allowing for the evaluation of accuracy loss.

Furthermore, to improve efficiency, additional FI tools have been introduced. For example, BinFI[11] is an extension of TensorFI designed to identify critical bits in DNNs. Another tool, LLTFI[12], can inject transient faults into specific instructions of DNN models implemented in either PyTorch or TensorFlow.

2.2. *Hardware-Aware FI Tools*

Hardware-aware FI tools perform fault injection in software while considering the underlying hardware through abstract models of the DNN hardware accelerator.

In a paper from 2023[13], the authors used an RTL model of a SA to conduct their experiments. Similarly, in a study conducted by aziz et al.[14], the authors map

a DNN onto the RTL implementation of the accelerator, studying the effects of transient faults in memory and datapath with high accuracy. In these studies, FI is performed in software, while all relevant parameters are integrated with the corresponding hardware components. In [15], the authors implemented both the DNN and the fault injector in software, inspired by an FPGA-based DNN accelerator. Additionally, in [16], the DNN and FI are implemented in Keras, with the architecture of a SA accelerator considered for a fault-tolerant design. Similarly, in [17], the authors evaluate their proposed reliability improvement technique for memories in TensorFlow, injecting transient faults into the weights. In [18], PyTorch is used to implement the DNN, with transient faults injected into activations (datapath or MAC units) and weights (memory) in the context of the SA accelerator model. Goldenstein et al.[19] also use PyTorch and inject faults through a custom framework called TorchFI, which targets the outputs of CONV and FC layers of the network.

The effect of permanent faults in the outputs of the Processing Elements (PEs) of Systolic Arrays was studied in 2022[20], where the model of the accelerator is based on implementing the DNN within the N2D2 framework[21]. Additionally, Hoang et al.[22] use PyTorch to study permanent faults in Multiply and ACcumulate (MAC) units of an accelerator during training, with the goal of improving reliability during inference. In a study from 2021[23], the authors developed a Keras-based accelerator simulator to investigate the impact of permanent faults on the accelerators' on-chip memory, injecting permanent faults into activations and weights. The weight remapping strategy in memory to mitigate the effects of permanent faults is evaluated using Ares[24]. Zhao et al.[25] used SCALE-Sim[26], a systolic CNN accelerator simulator, to study the impact of permanent faults in PEs and computing arrays in systolic array-based accelerators.

Similar to hardware-independent platforms, faults are injected based on Bit-Error Rate (BER) or fault rate, with experiments repeated to achieve a 95% confidence level and a 1% error margin[16]. The main drawbacks of existing reliability assessment methods for DNNs can be summarized as follows:

- There is no software FI framework for hardware-aware platforms, suggesting a potential for DNN accelerator simulators to be exploited or developed for reliability assessment;
- Many FI studies evaluate reliability by assessing accuracy loss and fault classification, with some considering FIT (Failure In Time) [27]. However, there is a pressing need to introduce DNN-specific metrics for reliability evaluation. In this work, we introduce a new metric called faulty distance to provide a deeper understanding of network resilience.

3. Proposed Methodology

The proposed methodology for the SAFFIRA framework is depicted in Fig. 2. The process begins with providing the trained network parameters and architecture. In step one, the fault list is generated, with possible fault locations either defined by the

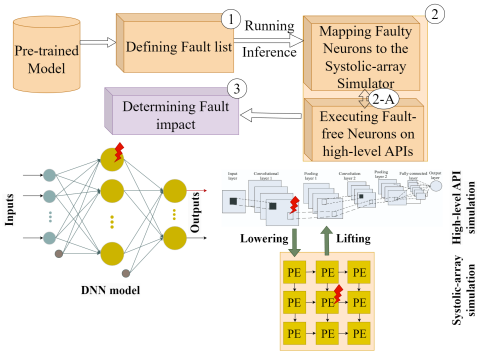


Fig. 2: SAFFIRA methodology[28]

user or randomly determined based on the network parameters by the framework. Faults can be configured as transient or permanent, targeting various activations of the DNN.

In step two, the fault injection campaign is executed within the systolic-array simulation environment in Python, while the remaining network computations are performed using a high-level API (e.g., PyTorch) to enhance efficiency. The transition between the high-level API and the systolic-array simulator (step 2-A) is managed using the Lowering and Lifting Strategy (LoLif), detailed in subsection 3.1.2.

Finally, in step three, the framework reports the reliability of the network and the impact of the faults using various metrics.

SAFFIRA supports multiple data representations, including fixed-point, integer, and floating-point formats. Additionally, the framework accommodates various mapping scenarios relevant to systolic-array architectures, such as output-stationary and weight-stationary mappings. These flexibilities enable researchers to adapt the framework to diverse applications and customize reliability assessments to meet specific hardware requirements.

3.1. *Hardware simulations*

SAFFIRA is a SA model based on the URE system. A SA can be generated to solve problems defined by a URE system[29]. In the case of SAFFIRA, the URE system corresponds to matrix multiplication, which is the primary operation executed on the SA for DNNs processing. The following subsection provides the formal details required to simulate this operation, followed by the fault injection process in this context. Additionally, strategies specifically tailored to DNNs are discussed.

3.1.1. *Mathematical formalism*

A URE system[29] is defined over an integer lattice L_n, consisting of points p in the n-dimensional Euclidean space E_n. The objective is to solve a system of equations associated with variables $x_1(p), x_2(p), \ldots, x_m(p)$ for all points $p \in R$, where $R \subseteq L_n$.

This system can be either uni-variate or multi-variate. In this work, only the uni-variate case is considered, resulting in a system of the following form:

$$\begin{aligned} x_1(p) &= f[x_1(p - w_1), \cdots, x_m(p - w_p)], \\ x_2(p) &= x_2(p - w_2), \\ &\vdots \\ x_m(p) &= x_m(p - w_p). \end{aligned} \tag{1}$$

The points $p - w_{i_k}$ belong to L_n, and the vectors w_k are constants independent of p, which is why this type of dependence is referred to as *uniform dependence*. Each equation $x_i(p)$ depends on the points $p - w_{i_k}$.

Quinton proposes a strategy to model a SA starting from the problem to be solved[29]. Specifically, they outline three key steps:

(1) Find a URE system for the problem to solve.
(2) Find a timing function compatible with the dependencies of the URE system.
(3) Find an allocation function to map the URE onto a finite architecture.

The main idea is to project the space E_n twice: the first time, the resulting points will correspond to the spatial arrangement of each PE. The second projection determines iso-temporal planes, identifying operations that are computed during the same clock cycle but on different PEs; each plane corresponds to a different clock cycle. The space-projection matrix P and the temporal dimension vector π are used later.

3.1.2. *Convolutions*

The strategy explained above opens the possibility to implement a variety of algorithms as a systolic array. Based on the literature, it is possible to perform a convolution as a matrix multiplication[30]. The experiments shown below are performed using a systolic array to perform the matrix multiplication $C = A \times B$. The associated URE is the following.

$$
\begin{aligned}
c(i,j,k) &= c(i,j,k-1) + a(i,j-1,k) \times b(i-1,j,k) \\
a(i,j,k) &= a(i,j-1,k) \\
b(i,j,k) &= b(i-1,j,k) \\
&\text{initial conditions} \\
a(i,0,k) &= a_{ik}, \quad \forall i,k \\
b(0,j,k) &= b_{kj}, \quad \forall k,j \\
c(i,j,0) &= 0, \quad \forall i,j
\end{aligned}
\tag{2}
$$

where $p = (i,j,k) \in R \subseteq L_n$, in which i, j and k assume values between 1 and $N1$, $N2$, $N3$ respectively. $N1$, $N2$ and $N3$ are problem parameters such that $A \in \mathbb{R}^{N_1,N_3}, B \in \mathbb{R}^{N_3,N_2}, C \in \mathbb{R}^{N_1,N_2}$.

When performing a convolution, the input matrices must be *reshaped* to ensure that the output of the SA corresponds to the convolution result. In this paper, this concept is referred to as LoLif. Specifically, to compute a convolution $C = A * B$, it can be implemented as a transformation lif of the matrix multiplication of reshaped matrices, $low_a(A) \times low_b(B)$. In formulas:

$$
C = lif(low_a(A) \times low_b(B)), \tag{3}
$$

where Lif, Low_a and Low_b are corresponding transformations, as shown in the example of Fig. 3

3.1.3. *Simulation and Injections*

To perform the simulation, it is sufficient to solve the system described above (Eq. 2). However, this method also enables the injection of faults into the values in a hardware-aware manner. Fault injection can be achieved by altering the values $a(p)$, $b(p)$, or $c(p)$ at specific points p. The resulting faulty values are then propagated to the subsequent PEs, simulating the impact of the faults on the system. Given that each point p is projected to the physical space $r = (x,y)$ using the physical space-projection matrix, $r = Pp$, it can be inferred that how the injected values are propagated through the different PEs. Specifically, for some dependence vector d for the different labeled variables a, b, c. Looking at the system above in Eq. 2, the following can be observed: $d_a = (0,1,0)$, $d_b = (1,0,0)$ and $d_c = (0,0,1)$. Afterward, the propagation direction can be found using the same relationship shown before: $\delta x_i = Pd_i, i = \{a,b,c\}$. This means that the value of a in some PE in position s will be propagated to the PE in position $s + \delta x_a$. The same reasoning can be done for the time, supposing that a fault is propagated not only in space but also in time. We can compute $\delta t_i = \pi d_i$. For simplicity, $\pi = (1,1,1)$ is fixed to reduce the exploration space. In this case, the time dependency δt_i will always be 1: $\delta t_i = 1$.

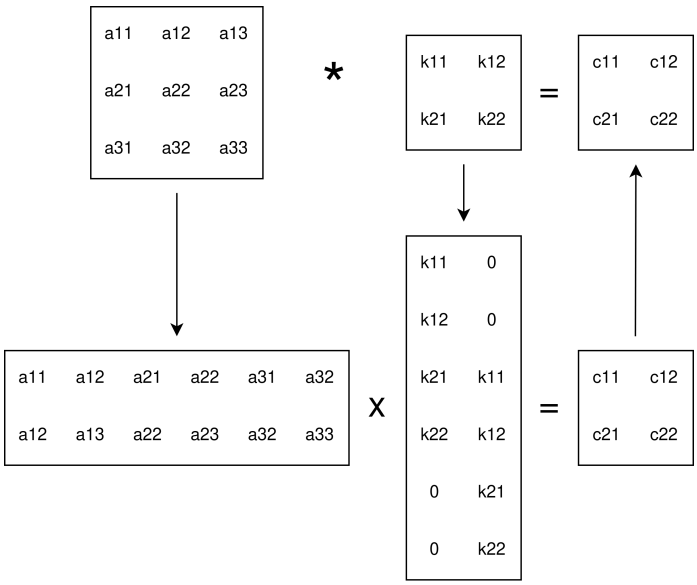


Fig. 3: LoLif example. Applied transformations are similar to im2col and im2row.

Figure 4 shows an example. 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 space, to the element cascading from s. Note that the value propagation only happens after each clock cycle. This means that the next injected element will be displaced also 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 set of points belonging to the injection can be transposed back into the iteration space E_n using the pseudo-inverse P^{-1}. The set of points identified with this strategy will be subject to injection. Formally, injection is as a function h applied to a variable:

$$a(p) = h(a(p - w_a)) \tag{4}$$

4. Reliability Assessment

DNN's resilience The reliability is evaluated by comparing the output probability vector of the golden run (i.e., the DNN operating as expected, without faults) and the faulty run (i.e., the DNN with injected faults). This evaluation can be performed using various metrics, as highlighted by Ahmadilivani et al.[2], who provide

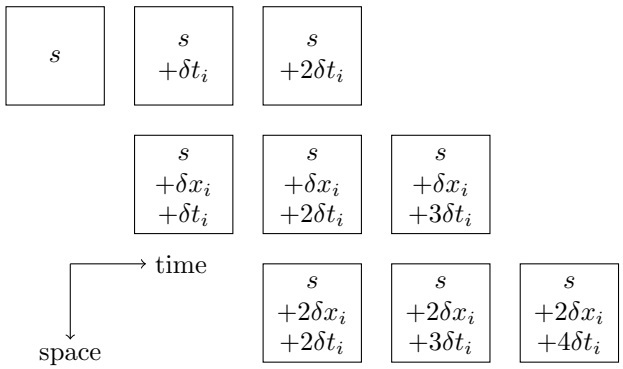


Fig. 4: When injecting element s, the fault is propagated in time (thus affecting elements $s+\delta t_i$ and $s+2\delta t_i$) and in space (forwarding the faulty value to neighboring elements $s + \delta x_i + \delta t_i$, $s + 2\delta x_i + \delta t_i$ and so on).

an overview of the most commonly used metrics for this purpose.

Three primary ways are used to estimate the reliability:

- **Accuracy Loss** - it is defined as $a_g - a_f$, where a_g is the accuracy of the golden system and a_f is the accuracy of the faulty one.
- **SDC Rate** - it measures the amount of Silent Data Corruption (SDC) that happen in the system as the proportion of faults that caused misclassification in comparison to the golden model [31].
- **Fault Classification** - it consists in assigning classes to faults depending on the measure outcome; e.g. we could label a fault "critical" if there is misclassification with respect to the golden simulation and "safe" if there is not.

These metrics have some drawbacks, as they provide a general trend of the DNN's performance without offering meaningful insights into the underlying reasons for such behavior. More specifically, **Accuracy Loss** measures the extent of DNN performance degradation in the presence of faults. While it provides an understanding of how many samples are misclassified, it lacks additional information regarding other classes and the confidence level of the predictions.

The **SDC Rate** enhances this concept by quantifying the number of misclassifications and their severity (e.g., SDC-2, SDC-10%, etc.[31]). However, this metric does not capture the magnitude of the misclassification; instead, results are grouped into bins predefined by the experimenter.

Fault Classification takes this further, albeit in a different direction, by providing detailed insights into various categories of faults. These categories can be customized based on specific constraints used to compare faulty and golden inferences, offering a more granular view of fault impact. Also, the categories in Fault Classification are somewhat arbitrary, although they provide more meaningful in-

formation. In fact, authors often set arbitrary thresholds for these categories. For instance, Pappalardo et al.[13] chose a 5% threshold, meaning that if the confidence of the faulty inference is less than 5% of the golden inference, the fault is considered more severe than others.

To address these issues, we propose a novel metrics that provide more detailed insights into specific fault patterns. The following subsections describe all the metrics used in this paper, while introducing novel concepts aimed at developing new metrics to better evaluate the performance of DNNs.

4.1. *Metrics in the literature*

For the purposes, we have chosen to use the **SDC Rate** and **Fault Classification** metrics, as defined in the following sections.

4.1.1. *SDC Rate*

The SDC Rate is defined as the probability that a fault produces an SDC[31]. This is a family of metrics, where each metric is based on a different criterion. Specifically, we define the following:

- **SDC-1**: The top-ranked element predicted by the DNN differs from the one predicted by its fault-free execution. This is considered the most critical SDC because the top-ranked element is typically used for downstream processing.
- **SDC-5**: The top-ranked element is not among the top five predicted elements in the fault-free execution of the DNN.
- **SDC-10%**: The confidence score of the top-ranked element deviates by more than $\pm 10\%$ from its fault-free execution.
- **SDC-20%**: The confidence score of the top-ranked element deviates by more than $\pm 20\%$ from its fault-free execution.

These metrics give us a good representation of the misclassification behavior of the network depending on the probability of the fault-free top-ranked class.

4.1.2. *Fault categorization*

This metric assigns a "label" to each fault depending on the relationship between the faulty and the golden inferences. More specifically, faults are labeled using the following categories:

- **masked**: A fault is considered masked when $F = G$, where F and G represent the faulty and golden inferences, respectively.
- **safe**: A fault is considered safe if there is correct classification ($\arg\max(F) = \arg\max(G)$), but the observed output is different ($F \neq G$).
- **critical**: A fault is considered critical if there is misclassification, i.e., $\arg\max(F) \neq \arg\max(G)$.

The specific conditions and labels used in this metric can vary, which may limit its informativeness. However, despite these variations, the metric still provides valuable insights for evaluating the novel proposals in this study.

4.1.3. *Failures in Time*

It is possible to compute the hardware reliability by computing the Failures In Time (FIT) rate, corresponding to the number of failures per time unit. It can be calculated by differentiating SDC rates of injected transient faults into defined classes and calculating FIT for the accelerator (*accel*) by its components (*comp*) with (5) in which FIT_{raw} is provided by the manufacturer, $Size_{comp}$ is the total number of the component bits, and SDC_{comp} is obtained by FI.

$$FIT_{accel} = \sum_{comp} FIT_{raw} \times Size_{comp} \times SDC_{comp} \tag{5}$$

4.2. *Novel Real Metrics*

As discussed earlier, the metrics used so far provide information on the proportion of failures, primarily misclassifications, caused by faults. However, they do not adequately capture the magnitude of these misclassifications. To address this gap, we propose two new metrics designed to quantify the extent of misclassification at the output of the DNN.

4.2.1. *Faulty Distance*

Assuming the golden probability vector is denoted by G, the faulty probability vector by F, and the function $ag(\cdot)$ represents the argmax function, the faulty distance (FD) is defined as:

$$FD = \left(1 - \frac{G \cdot F}{\|G\| \cdot \|F\|}\right) \cdot (ag(F) - ag(G)) \tag{6}$$

In this metric, the cosine similarity, defined as $\cos\theta = \frac{G \cdot F}{\|G\| \cdot \|F\|}$, is employed. Cosine similarity is a measure used to quantify the similarity between two non-zero vectors in an inner product space. It computes the cosine of the angle between the vectors, which effectively normalizes their dot product. In the context of this study, cosine similarity is applied to compare the probability distributions of the classes in both the faulty and golden modes.

The cosine similarity produces values between -1 and 1. A value closer to 1 indicates that the vectors are highly similar. Thus, the faulty distance metric will return 0 when the faulty output aligns with the correct classification. The larger the faulty distance, the more severe the misclassification.

Finally, the average faulty distance (AFD) is defined as the mean of all individual faulty distances:

$$AFD = \frac{1}{N}\sum_{i=1}^{N} FD_i \tag{7}$$

Where N represents the total number of points in the dataset.

4.2.2. *Euclidean Distance*

Similarly, we define the Euclidean distance as follows:

$$ED = ||G - F||_2 = \sqrt{\sum_i (G_i - F_i)^2}, \tag{8}$$

where the operator $||\cdot||_2$ correspond to the L_2-norm, also known as the Euclidean distance.

In this case, a value of 0 indicates that F and G are equal, thus the fault is masked: there is no difference in the output of the network in presence of faults. Then, the more difference the two vectors are, the higher the metric is.

Although this strategy seemed promising, it failed to give us meaningful information on the actual outcome of the faults: as presented in section 5, it is not possible to understand whether there was misclassification or not. Given this perspective on the problem, we have developed more sophisticated metrics to better capture the nuances and improve upon the existing approach. To achieve this, we sought to encode the information in the complex plane, which allows for a richer representation of the fault-induced variations in the model's behavior.

4.3. *Metrics on the Complex Plane*

For our problem, we need to capture two key pieces of information:

- Whether there was a misclassification.
- The magnitude of the difference between the golden inference and the faulty inference.

To address this, we have designed metrics on the *complex plane* where one piece of information is represented in the real part, and the other is represented in the imaginary part. By encoding these aspects in separate components, we aim to provide a more comprehensive assessment of the fault's impact on the model. We expect these metrics to assist in classifying inferences based on the severity and nature of the faults and, hopefully, gain a deeper understanding of how faults affect the DNNs. Ideally, we envision these metrics being used in the future to assess the reliability of networks, thereby minimizing the number of experiments required or, at the very least, reducing the size of the input set needed for evaluation.

Hence, we propose a transformation to apply to F and G.

First, we treat the outputs F and G as probability distributions by applying the softmax function, resulting in:

$$F^s = \text{softmax}(F), \quad G^s = \text{softmax}(G) \tag{9}$$

This transformation allows us to interpret the components of F^s and G^s as the confidence of the network's predictions. Additionally, the sum of the components for both vectors will equal 1:

$$\sum_i G_i^s = \sum_i F_i^s = 1 \tag{10}$$

Next, we define a new quantity, ρ, to represent the confidence of the network for a given label l, where $l = arg\max(G^s)$. This will provide a more detailed measure of the network's certainty regarding its predictions. Given this, we define the vectors χ and Ω as follows:

$$\begin{aligned} \chi_g &= argsort(G^s), \qquad \Omega_g = sort(G^s), \\ \chi_f &= argsort(F^s), \qquad \Omega_f = sort(F^s), \end{aligned} \tag{11}$$

where the function $argsort(x)$ returns the indices that would sort the vector x; hence, $argsort(G^s)$ give us a vector of indices such that G^s would be sorted with the highest probability first. Please note that we can interpret the index of F^s and G^s as the class corresponding to the prediction. Additionally, in this paper we use 0-indexed vectors.

Vectors χ and Ω give us the classes from the most likely to the least likely and the confidence probability respectively. This means that given G^s the predicted class will be $\chi_g[0]$ with probability $\Omega_g[0]$. The same applyst to F^s

Finally, we define three novel complex equations: the Complex Distance, the Radial Distance and the Phasic Distance.

4.3.1. *Complex Distance*

We define the complex distance as follows:

$$ID = (||\Omega_g||_2 - ||\Omega_f||_2) + i(||\chi_g[0:2]||_2 - ||\chi_f[0:2]||_2) = I_f - I_g \tag{12}$$

In this case, we encode the norm of Ω_k as the real part of I_k and the norm of the first two components of χ_k as the imaginary part of I_k (where $k \in \{f, g\}$). And then we take the difference between I_f and I_g, obtaining the metric.

In this case, when the fault is masked, the metric will have a value of 0. As the fault increases, the metric will grow based on two factors:

- The distance between the probabilities on the real axis (i.e., the confidence values of the predicted classes)

- The distance between the first two most likely classes (i.e., the classes with the highest probabilities in F^s and G^s).

Thus, the metric captures both the confidence of the network in its prediction and the degree of misclassification by measuring how far the faulty output deviates from the expected output. The further the fault-induced change in these factors, the higher the metric value, indicating a more severe misclassification.

4.3.2. *Radial Distance*

We define the radial distance as follows:

$$RD = (2 - \delta(\chi_f[0] - \chi_g[0])) \exp\left(i||\Omega_g - \Omega_f||_2\right) \tag{13}$$

where $\delta(x)$ is the Dirac-delta, such that:

$$\delta(x) = \begin{cases} 1 & \text{if } x = 0 \\ 0 & \text{otherwise} \end{cases} \tag{14}$$

In this case the points will form an arc with its center at 0 and a radius dependent on the first class: if there is misclassification the radius will be 2 otherwise it will be 1. The angle of each point depends on the phase, which in turn depends on the distance between the probability vectors Ω.

4.3.3. *Phasic Distance*

The exponential distance is, in a way, similar to the radial distance and is defined as follows:

$$PD = ||\Omega_g - \Omega_f|| \exp\left[i(2 - \delta(\chi_g[0] - \chi_f[0]) - \delta(\chi_g[1] - \chi_f[1]))\right] \tag{15}$$

In this case, the points will form three lines. Each line depends on the first two classes:

- If they are the same, the phase, and therefore the angle, will be 0,
- If only one is different, the phase will be 1,
- If both are different, the phase will be 2.

The distance from the center is influenced by the separation between the probability vectors. This metric appears promising, as it could enable us to characterize the distribution of points along a specific line, providing a way to quantify the reliability of the network.

Table 1: Base accuracy of networks under test

DNN	accuracy (%)
8-bit LeNet-5 (MNIST)	93.8
16-bit LeNet-5 (MNIST)	95.4
AlexNet (CIFAR-10)	78.0
VGG-16 (CIFAR-10)	93.4
ResNet-18 (CIFAR-10)	93.8

5. Experiments and Results

Two distinct sets of experiments are conducted using SAFFIRA. The first set involves fault injection based on a permanent-fault model applied to two quantization versions of the LeNet-5 network (8-bit and 16-bit integers). The second set focuses on fault injection using a transient fault model across three different benchmark networks: AlexNet, VGG-16, and ResNet-18. All networks are fully quantized to the INT data type, encompassing activations, weights, and biases. The base accuracies are reported in Table 1

The SA model for these experiments is *output stationary*, meaning that its physical-space projection matrix P is as follows:

$$P = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \tag{16}$$

Such a matrix corresponds to a rectangular SA with $N1 \times N2$ PEs. Please note that with this projection, the variable c (i.e. the partial sum) is a *stationary variable* since it is always available on the same PE regardless of the iteration. Whether a variable is stationary or not depends on the employed projection.

In all experiments, FI is repeated several times to reach an acceptable confidence level[32]. This work provides an equation to reach 95% confidence level and 1% error margin.

5.1. *Computation Time*

The experiments were performed on a server using *Python* with an Intel Xeon Silver 4210, with a total number of 40 cores. SAFFIRA completes 500 inferences of two convolutional layers, with the same sSA, in about 10 minutes with minimal optimization. This means a total of about 16.3 simulations per second. For comparison, by utilizing the framework presented in [33] to perform fault injection on the same networks as this work, on average, 5.8 simulations per second are executed. The mentioned framework is the state-of-the-art hybrid (software/hardware co-design) hardware-aware fault injection framework. Therefore, SAFFIRA provides about 2.8× speed up by performing the same analysis. Also, the same fault injection campaign is performed at the RT level using QuestaSim. The results show

0.007 simulation per second, which is 2100× slower than the proposed method in this work.

5.2. *Metrics*

Table 2 shows the results of the FI for permanent fault injection experiments on LeNet-5 with the different metrics. It can be seen that this network was highly susceptible to the injected permanent faults. Specifically, the SDC-1 and SDC-5 are very high: on average, about 82% of the time, the faulty inference misclassified the input; furthermore, about 93.5% of the inputs were completely missed since the correct label was below the fifth position. The SDC-10% and SDC-20% rates are very high as well: more than 95% of the inputs had the correct class with a probability much too low than expected. Average Faulty Distance (AFD) is also reported that shows the 16-bit network in this particular case, is more reliable compared to the 8-bit network in the presence of permanent faults in the systolic architecture.

These results show that the DNN used was not usable in a safety-critical environment. This result was expected since the network was not trained to withstand stuck-at faults like the ones injected.

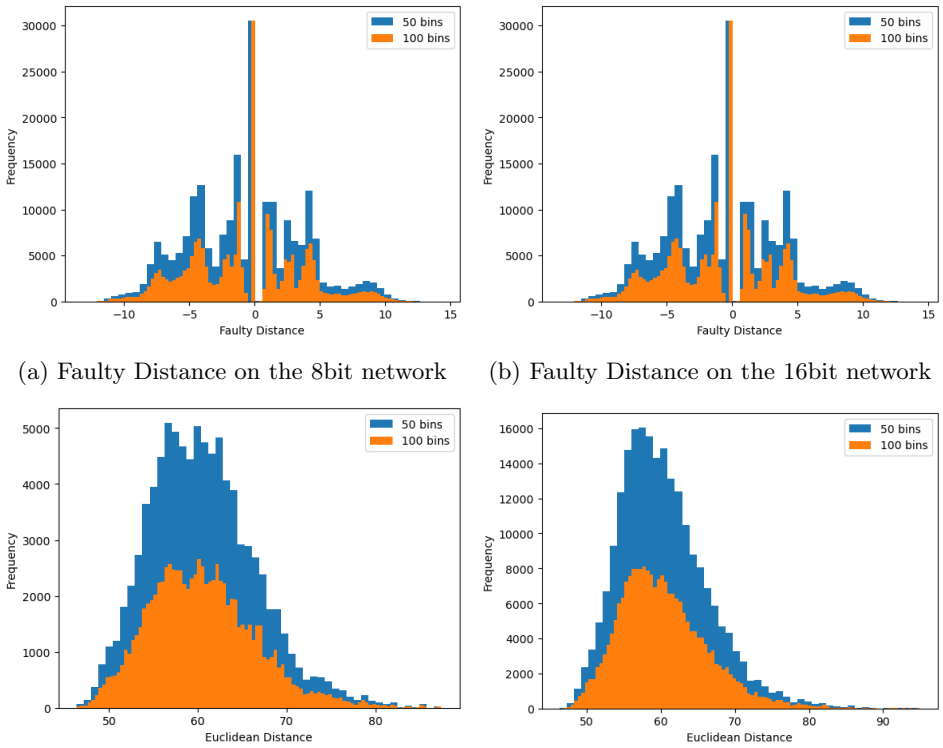
For the second experiment, only SDC and AFD are reported in table 3. They show that these DNN are much more reliable than LeNet. As observed, the AFD is very small due to the minimal amount of misclassification occurring in the presence of faults. For this reason, the following studies will focus on the two LeNets DNN.

Table 2: FI experiments results on two LeNet-5

Metric	16bit	8bit
SDC-1 (%)	77.84	87.70
SDC-5 (%)	93.05	94.49
FIT (failures/10^9 hours)	4.9e-4	5.0e-4
SDC-10% (%)	98.16	98.53
SDC-20% (%)	96.21	96.97
AFD	-0.04	-0.53

5.3. *Proposed metrics*

In the previous subsection, only the AFD was presented. However, it is also insightful to examine the distribution of the points. Figure 5 presents the histograms of the metrics for each experiment, using both 50 and 100 bins. Regarding the FD, a peak is observed at 0, indicating the correctly classified inputs. The height of this peak is exactly the complement of the SDC-1 metric in absolute terms. Additionally, two



(a) Faulty Distance on the 8bit network (b) Faulty Distance on the 16bit network

(c) Euclidean Distance on the 8bit network (d) Euclidean Distance on the 16bit network

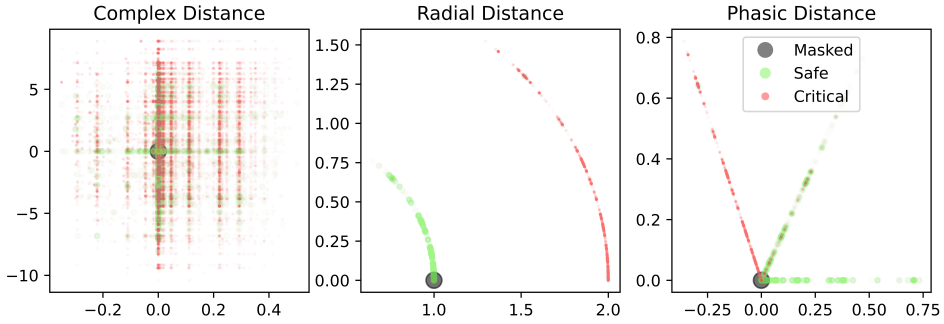
Fig. 5: Histogram plot of the Faulty distance values

Table 3: Reliability analysis of different state-of-the-art DNN benchmarks

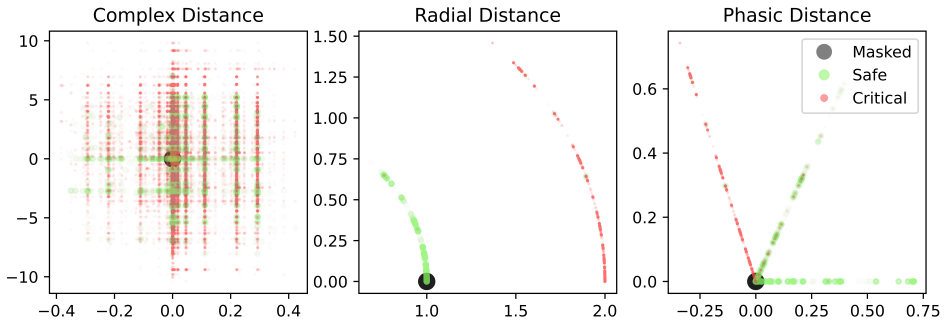
DNN	SDC-1	SDC-5	SDC-10%	SDC-20%	AFD
AlexNet (CIFAR-10)	4.3	29.1	13.1	9.7	7.1×10^{-2}
VGG-16 (CIFAR-10)	3.0	40.0	46.5	84.5	0.19×10^{-2}
ResNet-18 (CIFAR-10)	1.5	23.0	16.5	82	0.16×10^{-2}

distinct but similar trends emerge for the two networks. Figure 5a reveals three additional peaks around +1, −1, and −5. This suggests that while many inputs were misclassified, the difference between the faulty and golden vectors was generally not large. On the other hand, figure 5b shows many more peaks, this means that it is more difficult to predict how a fault will propagate in this case. Figures 5c and 5d show the distribution of the Euclidean distance. The distribution shows a peak at around 58 for both the cases: this is expected since the two networks are two different quantization of the same network.

5.3.1. *Complex Metrics*



(a) Complex metrics for 8bit network



(b) Complex metrics for 16bit network

Fig. 6: Complex metrics. Each graph shows the complex plane with the real part on the x-axis and the imaginary part on the y-axis. The points have different opacities and radii.

Figure 6 shows the complex metrics. In all the subfigures, the x-axis corresponds to the real part, while the y-axis corresponds to the imaginary part. The points are plotted, for clarity, using the same fault classification given in section 4.1.2: big black points describe masked faults, medium-sized green points describe safe faults (i.e. no misclassification) and red points describe critical faults.

Regarding the complex distance, we can see that no clear separation between the three categories is clear, nevertheless some interesting patterns emerge that deserve to be studied in deep, although such study is outside the scope of this paper. The radial and phasic distance, on the other hand, show very different and clear separation: the masked faults are all in $(0, 0)$ while the safe faults and the critical faults are distributed predictably in both cases. The radial distance shows three distinct categories: the masked faults are in $(0, 0)$, the safe faults are distribuited

around a circumference arc with radius 1 and the critical faults are distributed along an circumference arc with radius 2. Similar observation can be made about the phasic distance: it shows four distinct categories: the masked faults, the safe faults distributed along a line with phase 0, critical faults along a line with phase 2 and a mix of the two on a line with phase 1.

6. Discussion

The presented results try to quantify the error that a DNN application makes when subjected to hardware faults. They represent important first steps toward explainability of the application behavior when subjected to faults. On top of that, we report an important finding: when subjected to hardware faults, the DNN misclassifies if both the first and the second classes of the faulty inference are different from the golden inference, as visible in figure 6 (phasic distance). In facts, the majority of the critical faults are located on the line with phase 2. This finding represent an important step into the comprehension of neural network processing. In general, this results could be used to quantify the reliability of the application in a more precise manner or online to assure the system reliability by employing double modular redundancy. The authors believe such topic deserve in-depth studies and intend to follow up with them.

7. Conclusions

This paper presents a novel hierarchical fault injection strategy for systolic arrays, addressing the time efficiency issue by introducing a novel hierarchical software-based hardware-aware fault injection strategy tailored for systolic array-based DNN implementations. The approach demonstrates a reduction of the fault injection time up to threefold compared to the state-of-the-art hybrid (software/hardware) hardware-aware fault injection frameworks and more than 2000× compared to RT-level fault injection frameworks — without compromising accuracy. Additionally, we propose and evaluate new metrics through experimental assessment, putting the basis for (i) more sophisticated reliability metrics and (ii) better understand the inner-working of DNNs.

Acknowledgements

The authors sincerely thank Paul Jimenez, from INL team in Ecole Centrale de Lyon, for proposing the idea of using complex metrics and for his efforts in their conceptualization and development.

This work was supported in part by the Estonian Research Council grant PUT PRG1467 "CRASHLESS" and by Estonian-French PARROT project "EnTrustED".

References

1. M. Taheri, Dnn hardware reliability assessment and enhancement, in *27th IEEE European Test Symposium (ETS)*, 2022.
2. M. H. Ahmadilivani *et al.*, A systematic literature review on hardware reliability assessment methods for deep neural networks, *ACM Computing Surveys* **56**(6) (2024) 1–39.
3. A. Bosio, I. O'Connor, M. Traiola, J. Echavarria, J. Teich, M. A. Hanif, M. Shafique, S. Hamdioui, B. Deveautour, P. Girard *et al.*, Emerging computing devices: Challenges and opportunities for test and reliability, in *2021 IEEE European Test Symposium (ETS)*, IEEE2021, pp. 1–10.
4. 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)*, IEEE2023, pp. 124–127.
5. 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).
6. M. Taheri *et al.*, Deepaxe: A framework for exploration of approximation and reliability trade-offs in dnn accelerators, in *2023 24th International Symposium on Quality Electronic Design (ISQED)*, IEEE2023, pp. 1–8.
7. A. Ruospo *et al.*, A survey on deep learning resilience assessment methodologies, *Computer* **56**(2) (2023) 57–66.
8. M. H. Ahmadilivani *et al.*, Special session: Approximation and fault resiliency of dnn accelerators, in *2023 IEEE 41st VLSI Test Symposium (VTS)*, IEEE2023, pp. 1–10.
9. 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)*, IEEE2020, pp. 25–31.
10. N. Narayanan *et al.*, Fault injection for tensorflow applications, *IEEE Transactions on Dependable and Secure Computing* (2022).
11. Z. Chen *et al.*, Binfi: an efficient fault injector for safety-critical machine learning systems, in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2019, pp. 1–23.
12. U. K. Agarwal, A. Chan and K. Pattabiraman, Lltfi: Framework agnostic fault injection for machine learning applications (tools and artifact track), in *2022 IEEE 33rd International Symposium on Software Reliability Engineering (ISSRE)*, IEEE2022, pp. 286–296.
13. S. Pappalardo *et al.*, Resilience-performance tradeoff analysis of a deep neural network accelerator, in *2023 26th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS)*, IEEE2023, pp. 181–186.
14. A. Azizimazreah *et al.*, Tolerating soft errors in deep learning accelerators with reliable on-chip memory designs, in *2018 IEEE International Conference on Networking, Architecture and Storage (NAS)*, IEEE2018, pp. 1–10.
15. W. Li *et al.*, Soft error mitigation for deep convolution neural network on fpga accelerators, in *2020 2nd IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, IEEE2020, pp. 1–5.
16. E. Ozen and A. Orailoglu, Low-cost error detection in deep neural network accelerators with linear algorithmic checksums, *Journal of Electronic Testing* **36**(6) (2020) 703–718.
17. M. Jasemi, S. Hessabi and N. Bagherzadeh, Enhancing reliability of emerging memory technology for machine learning accelerators, *IEEE Transactions on Emerging Topics in Computing* **9**(4) (2020) 2234–2240.

18. E. Ozen and A. Orailoglu, Boosting bit-error resilience of dnn accelerators through median feature selection, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* **39**(11) (2020) 3250–3262.
19. B. F. Goldstein *et al.*, A lightweight error-resiliency mechanism for deep neural networks, in *2021 22nd International Symposium on Quality Electronic Design (ISQED)*, IEEE2021, pp. 311–316.
20. S. Burel, A. Evans and L. Anghel, Mozart+: Masking outputs with zeros for improved architectural robustness and testing of dnn accelerators, *IEEE Transactions on Device and Materials Reliability* **22**(2) (2022) 120–128.
21. "N2D2 CAD framework for DNNs" `https://github.com/cea-list/N2D2`, [Online].
22. L.-H. Hoang, M. A. Hanif and M. Shafique, Tre-map: Towards reducing the overheads of fault-aware retraining of deep neural networks by merging fault maps, in *2021 24th Euromicro Conference on Digital System Design (DSD)*, IEEE2021, pp. 434–441.
23. Y.-Y. Tsai and J.-F. Li, Evaluating the impact of fault-tolerance capability of deep neural networks caused by faults, in *2021 IEEE 34th International System-on-Chip Conference (SOCC)*, IEEE2021, pp. 272–277.
24. T.-H. Nguyen *et al.*, Low-cost and effective fault-tolerance enhancement techniques for emerging memories-based deep neural networks, in *2021 58th ACM/IEEE Design Automation Conference (DAC)*, IEEE2021, pp. 1075–1080.
25. Y. Zhao, K. Wang and A. Louri, Fsa: An efficient fault-tolerant systolic array-based dnn accelerator architecture, in *2022 IEEE 40th International Conference on Computer Design (ICCD)*, IEEE2022, pp. 545–552.
26. A. Samajdar *et al.*, Scale-sim: Systolic cnn accelerator simulator, *arXiv preprint arXiv:1811.02883* (2018).
27. 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.
28. 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, in *2024 27th International Symposium on Design Diagnostics of Electronic Circuits Systems (DDECS)*, 2024, pp. 19–24.
29. P. Quinton, Automatic synthesis of systolic arrays from uniform recurrent equations, *ACM SIGARCH Computer architecture news* **12**(3) (1984) 208–214.
30. S. Hadjis *et al.*, Caffe con troll: Shallow ideas to speed up deep learning, in *Proceedings of the Fourth Workshop on Data analytics in the Cloud*, 2015, pp. 1–4.
31. 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 *SC17*, 2017.
32. R. Leveugle, A. Calvez, P. Maistri and P. Vanhauwaert, Statistical fault injection: Quantified error and confidence, in *2009 Design, Automation & Test in Europe Conference & Exhibition*, IEEE2009, pp. 502–506.
33. S. Pappalardo *et al.*, A fault injection framework for ai hardware accelerators, in *2023 IEEE 24th Latin American Test Symposium (LATS)*, IEEE2023.