

OpRA: Optimizing Resiliency Assessment for Deep Neural Networks

Original

OpRA: Optimizing Resiliency Assessment for Deep Neural Networks / Bellarmino, N., Barone, S., Pappalardo, S., Bosio, A., Cantoro, R.. - In: IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS. - ISSN 0278-0070. - 45:4(2026), pp. 1921-1934. [10.1109/TCAD.2025.3610062]

Availability:

This version is available at: 11583/3016145 since: 2026-09-30T09:59:44Z

Publisher:

Institute of Electrical and Electronics Engineers

Published

DOI:10.1109/TCAD.2025.3610062

Terms of use:

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

Publisher copyright

IEEE postprint/Author's Accepted Manuscript

©2026 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collecting works, for resale or lists, or reuse of any copyrighted component of this work in other works.

(Article begins on next page)

OpRA: Optimizing Resiliency Assessment for Deep Neural Networks

Nicolò Bellarmino, Salvatore Barone, Salvatore Pappalardo, Alberto Bosio and Riccardo Cantoro

Abstract—Deep Neural Networks (DNNs) and especially Convolutional Neural Networks (CNNs) are increasingly deployed in safety-critical domains, where rigorous reliability assessment is essential. Fault Injection (FI) remains a widely adopted technique to evaluate DNN robustness under hardware faults. However, the high computational cost of FI campaigns limits their scalability and practical adoption. These campaigns traditionally rely on three components: the DNN model, a fault list, and an input stimulus set. While fault selection strategies have been extensively studied, the impact of input stimulus selection remains largely underexplored.

This work addresses this gap by investigating the role of input stimuli in activating critical faults during FI-based reliability evaluation. We propose a strategy to prioritize inputs that are more likely to expose fault-induced vulnerabilities, guided by uncertainty-based metrics. Our methodology is validated across two fault models: memory-level bit flips and stuck-at faults in a systolic array accelerator. Results show that uncertainty-ranked inputs significantly increase fault activation and detection efficiency, enabling more focused and cost-effective reliability analysis.

Index Terms—Automated Design Methodology, Deep Neural Networks, Fault Emulation, Reliability, Resiliency Assessment

I. INTRODUCTION

Deep Neural Networks (DNNs), particularly Convolutional Neural Networks (CNNs), have become the state-of-the-art models for computer vision tasks in modern computational environments.

Their deployment in safety-critical embedded systems demands a comprehensive understanding of their resilience to random hardware faults. Hardware malfunctions can result from latent defects, aging, and external perturbations, leading to hardware faults that may degrade system performance, potentially causing incorrect inference results [1] at the edge.

Fault Injection (FI) campaigns are the de facto standard methodology for evaluating the resiliency of DNNs in the presence of hardware faults [2]. Unfortunately, the increasing complexity of DNNs models and the datasets imposes significant computational challenges for comprehensive reliability assessments: exhaustive FI across entire datasets and all possible fault locations remains simply computationally prohibitive [3]. To address this complexity, researchers have

explored FI optimization techniques, such as the concept of Statistical Fault Injection (SFI), which aims to drastically reduce the number of faults considered during the FI campaign [4]. However, these techniques act only on the faults that are injected into the system. An additional dimension in FIs campaign still not so widely considered is the input workload: the set of input (in this case, *images*) that should be used to evaluate the resiliency of CNNs.

This paper explores the role of input workload for FI campaigns, and propose some methodologies and optimizations. These are based on strategically selecting a subset of inputs, focusing on those that are *challenging* for the DNNs and the task under evaluation. By prioritizing these “*difficult*” samples, the proposed methodology aims to identify network vulnerabilities more efficiently. The difficulty of each input is quantified using metrics that can be computed from the output probability vectors. We empirically show that a small subset of challenging inputs is sufficient to capture critical insights into the network’s resilience, thereby minimizing the reliance on the entire dataset. Using only challenging samples, we can estimate the number of critical faults for at least one input in the dataset, achieving a reliable upper bound on network resilience while dramatically reducing computational demands.

The main contributions of this study are summarized as follows:

- Demonstrate that metrics computed from the probability vector can reliably identify samples that are particularly suitable for FI campaigns.
- Empirically demonstrate that selecting challenging samples enhances the detection of critical faults in neural networks without requiring the entire dataset.
- Establish that challenging samples facilitate an efficient assessment of network criticality by providing an overestimation of critical faults, yielding a practical upper bound on the network’s resilience.
- Validate that specific thresholds within these metrics can categorize inputs as challenging, enabling the systematic selection of the FI workload.

We conduct our experiments across two scenarios: (1) Hardware-Agnostic FI, simulating memory faults through bit-flips in DNN weights, and (2) Hardware-Aware FI, simulating faults in a systolic array used to deploy DNN. These confirm the findings in both contexts.

The rest of the paper is organized as follows: section II introduces core concepts used in the development of this work; section III shows relevant works related to the matter in hand; section IV explain the main idea behind the ranking strategy;

Salvatore Barone is with Department of Computer Science and Information Technologies, Pegaso University, Piazza Trieste e Trento 48 – 80132, Napoli, Italy (e-mail: [name.surname]@unipegaso.it)

Nicolò Bellarmino and Riccardo Cantoro are with Department of Automation and Computer Science, Politecnico di Torino, Torino, Italy (e-mail: [name.surname]@polito.it)

Alberto Bosio and Salvatore Pappalardo are with Univ Lyon, ECL, INSA Lyon, CNRS, UCBL, CPE Lyon, INL, UMR5270, 69130 Ecully, France. (e-mail: [firstname.lastname]@ec-lyon.fr)

section V shows the experiments we performed to validate the raking strategy and finally, section VI concludes the article giving perspective on the outcomes of the present work.

II. TECHNICAL BACKGROUND

A. Fault models

Faults in electronic devices can be broadly categorized into two main types: permanent and transient faults, each with distinct causes, characteristics, and implications.

Permanent faults arise from irreversible physical damage to electronic components. These faults typically result from manufacturing defects, material degradation over time, or external physical damage. Once they occur, they persist until the faulty component is repaired or replaced. The most commonly used abstraction for modeling permanent faults is the stuck-at-fault model, where a component or node in a circuit is permanently “stuck” at either a high (1) or low (0) voltage state [5].

Transient faults, on the other hand, are temporary disruptions caused by external environmental factors. High-energy particle strikes, electrical noise, and electromagnetic interference are common sources of transient faults [5], [6]. Unlike permanent faults, transient faults do not result in lasting damage to the hardware; instead, they momentarily alter the device’s behavior. A typical manifestation of transient faults is a random bit-flip in memory or register values, which can lead to erroneous computations if not detected and corrected promptly. The different nature of permanent and transient faults necessitates tailored approaches for their management [7].

B. Fault injection

FI is one of the most widely used techniques for evaluating the impact of faults on systems, particularly in DNNs. FI involves deliberately introducing faults into a system to observe its behavior under error conditions, thereby enabling the assessment and improvement of fault tolerance. Based on the abstraction level at which faults are injected, we can distinguish three types of FI techniques [8]: simulation-based, platform-based, and radiation-based.

Simulation-based FI is the most popular method in the research community [8] due to its flexibility and cost-effectiveness. This method operates on models of the device, described using simulation languages. Faults can be injected either at the architectural representation of the hardware model or directly at the software level of the application. In the latter case, the technique is also known as *software-based FI*, and it allows researchers to explore various fault scenarios without the need for extensive hardware resources. However, the primary challenge of simulation-based FI lies in the time required for exhaustive campaigns, especially for large DNNs.

Simulation-based FI can be:

- **Hardware-agnostic FI** is carried out by injecting faults directly at the DNN model level without considering the underlying hardware on which the model is running [6]–[8]. This typically involves simulating bit-flips in network weights or activations, which are often represented in formats like 32-bit floating-point (FP32).

- **Hardware-Aware FI** is more accurate than hardware-agnostic FI, since it incorporates the abstract models of the Deep Neural Network (DNN) hardware accelerator into the fault simulation process. This approach accounts for the specific architecture and behavior of the target hardware (Systolic Arrays, GPUs), enabling more precise and realistic fault impact analysis. Unlike purely software-level techniques, Hardware-Aware FI integrates the DNN with key hardware components like memory hierarchies, computation pipelines, or data transfer patterns, providing a detailed view of how faults propagate through the system. For instance, in [9], the authors utilized a Register-Transfer Level (RTL) model of a Systolic Array (SA) accelerator to conduct their FI experiments, simulating hardware-specific behaviors.

C. Statistical Fault Injection

Given the computational overhead of exhaustive FI campaigns, researchers have developed statistical sampling techniques [2]. SFI involves selecting a statistically significant subset of possible faults to inject, based on predefined confidence intervals and error margins.

The number of injections to be randomly performed can be computed using Equation (1), from [4], where, N is the fault population size, p is the estimated probability of faults resulting in a failure; e is the desired margin of and t is the cut-off point (or critical value) corresponding to the desired confidence level.

$$n = \frac{N}{1 + e^2 \cdot \frac{N-1}{t^2 \cdot p \cdot (1-p)}} \quad (1)$$

While SFI has been effective in limiting the number of faults to analyze, there is further opportunity to streamline resiliency assessments by reducing the input workload during FI campaign.

D. Systolic Array

The concept of a SA architecture comes from the 70s [10]. A SA is basically a grid of Processing Elements (PEs) that exchange data locally to perform computations. A SA could be implemented to solve any given algorithm that could be described as a Regular Iterative Algorithm [11].

SA have gained popularity recently due to their ability to enhance the throughput of DNN-based systems. By efficiently processing large amounts of data, SA help reduce the time required for training and inference [12], ultimately improving the performance of both the models and the overall system.

In contemporary implementations of SAs, the objective is to address the matrix multiplication problem. Consequently, the term “matrix multiplier SA” is employed, or simply SA within the context of this study. A SA can compute the $R = A \times B + C$ equation, where A , B and C are known matrices. To compute R , the rows of A are multiplied by the columns of B and then an element of C is summed. In this way, each PE performs a single Multiply And Accumulate (MAC) operation per clock cycle. Figure 1 shows the structure of a single PE. It is a

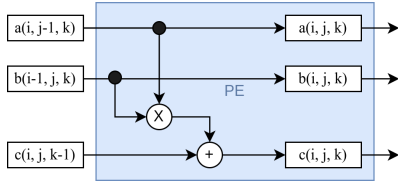


Fig. 1: General model of PE for matrix multiplication.

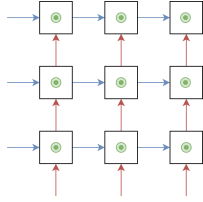


Fig. 2: Output Stationary Systolic Array. The arrows indicate the interconnection between inter-cell registers: in blue (left to right) the register a , in red (bottom to top) registers b and in green registers c .

synchronous circuit which contains three registers, one for each matrix. Every clock cycle, a PE performs the computation $c(i, j, k) = a(i, j-1, k) \cdot b(i-1, j, k) + c(i, j, k-1)$ and forwards the inputs as $a(i, j, k)$ and $b(i, j, k)$ and the partial sum $c(i, j, k)$. Depending on the way the registers a, b, c are interconnected among the PEs, different data flows emerge. In summary, three distinct configurations are possible: Output Stationary (OS), Row Stationary (RS), and Weight Stationary (WS) SAs. Consider reference [13] for further details.

III. RELATED WORKS

Testing refers to any activity aimed at assessing a system's correctness [8]. Machine learning (ML) models, including DNN, operate as statistical machines [14], meaning they generalize patterns from data rather than following predefined rules. Despite their deterministic execution—i.e. the same input and model parameters yield the same output—neural networks remain inherently difficult to test due to their statistical nature and high number of parameters.

Testing in the ML community differs fundamentally from traditional hardware testing: in ML, this primarily means evaluating a model's robustness and generalization capabilities. In this context, fault detection aims to identify test cases with a higher likelihood of misclassification—often referred to as *corner cases*—where the network is more prone to errors due to statistical uncertainties or limitations in training data [15].

The primary strategy for identifying these cases involves using a subset of the available data, commonly known as the *test set*. In [16], it is shown that *faults* in DNN are more easily detectable while using inputs lying on the decision boundary of the DNN. In their study, the authors propose metrics to select samples that can correctly identify erroneous behavior (mainly misclassification) in a trained model. The metrics can then be used to prioritize error-reviling inputs in order to run tests more efficiently.

In [17], authors show that uncertainty-based metrics correlate strongly with the accuracy of the DNN, indicating that such metrics can be used to identify misclassifications.

Assessing DNN performances becomes more important in the presence of *hardware faults*, where the networks behave differently concerning fault-free conditions. Hardware faults can induce errors during the inference process, potentially leading to incorrect predictions and reducing DNNs reliability. A fault in the underlying hardware can be, to some extent, modeled as a noise introduced in the parameters of the networks or in the intermediate representation that is computed during the forward pass (i.e., the *embeddings* in the different layers). These perturbations may affect the probability vector associated to each input. As a result of this, a perturbed probability vector may result in a wrong inference (e.g., image misclassification, wrong object detection, etc.).

In the context of *hardware faults*, only a few studies have explored strategies for ranking and selecting test set samples for FI experiments [18], [19]. Some of these have focused on crafting ad-hoc images specifically designed to detect critical faults in neural networks. For example, gradient-based adversarial techniques [20] have been extensively explored alongside. [21], [22] target image classification as task, and they proposed two methodologies for selecting test images from the dataset: a model-agnostic approach and a model-aided approach. The model-agnostic method uses distance-based metrics between images, while the model-aided test selection leverages confidence prediction metrics from the network's probability vector. These methods were validated using simulated hardware faults on a TPU, but were limited to Multi-Layer Perceptron (MLP) architectures, resulting in only 35% accuracy on CIFAR-10 dataset. Additionally, their approach excludes misclassified images from the test set, potentially overlooking cases critical to fault resilience assessment in state-of-the-art DNNs used in modern computer vision tasks.

Another set of studies [23], [24] proposed techniques for test image selection aimed at fault detection. These methods share similar objectives with previous works but differ in several significant ways. First, their proposed approach is designed to create a test set for continuous online fault detection, rather than offline fault assessment. Second, the fault models they use introduce errors at a larger scale than those traditionally used in CNNs reliability assessments. Specifically, they inject faults at rates from 0.05% to 1%, affecting multiple weights simultaneously. For example, in their "*worst-case scenario*" they select a predefined percentage of weights and flip their most significant bits (MSB), while in the "*random case*" bits are selected and flipped across the weights according to a set fault rate. According to this, applying a 1% fault rate to a VGG16 model [25], which contains 15,253,578 parameters, involves flipping the MSB of approximately 152,536 weights in the worst-case scenario. In the random case, for an 8-bit quantized network, around 1,220,286 bits are flipped. This completely disrupts the CNNs behavior. By contrast, in our research a fault affect a single bit, making detection significantly more challenging. Thus, our work considers the very worst-case scenario in terms of fault detectability. Results from previous studies are not directly comparable with our

work.

A recent contemporaneous study by [26] offers an innovative approach to reduce the computational load of reliability assessments, specifically targeting Approximate Computing hardware. They employ the Gini Index Computation on the probability vector to select samples expected to be more sensitive to approximation, allowing accurate loss estimation based solely on this subset. This technique models hardware-induced perturbations in neural networks through changes in the output probability vector, an approach closely related to fault modeling in CNNs, where perturbations can result from faults in either weights or intermediate activations during forward passes.

IV. REDUCING THE COMPUTATIONAL EFFORT OF FAULT-INJECTION

This section presents the proposed approach to efficiently computing the resilience assessment of a given DNN and its dataset. The proposed work targets image classification tasks.

A. Preliminaries

In the context of neural network-based classification tasks, the process typically involves two main phases: training and testing [14]. These rely on a labeled dataset $\mathbf{D} = \{(\mathbf{x}_i, y_i) : i = 0, 1, \dots, N-1\}$, where $\mathbf{x}_i$ represents the input features tensor and y_i denotes the corresponding class labels. Typically, $\mathbf{D}$ is divided into two subsets: the *training set* $\mathbf{D}_{\text{train}}$ and the *testing set* $\mathbf{D}_{\text{test}}$.

The testing phase involves evaluating the trained model f_θ on unseen samples from $\mathbf{D}_{\text{test}}$. This step provides an estimate of the model's generalization performance by calculating metrics like accuracy, precision, recall, or other relevant measures on $\mathbf{D}_{\text{test}}$. Let x_t be a sample belonging to the test set $\mathbf{D}_{\text{test}} = \{x_0, x_1, x_2, \dots\}$, and y_t be the corresponding class. Let f_θ be a generic model trained to infer N different classes with an inference function. When a sample x_t is fed into the model, a forward pass generates the likelihood vector $\mathbf{p} = f_\theta(x_t)$. This vector contains N elements corresponding to all possible classification outcomes, such that: $\mathbf{p} = (p_0, p_1, \dots, p_{N-1})$ where each element p_j in the vector represents the model's predicted probability that the sample x_t belongs to the class j . The following condition must hold for the likelihood vector: $0 \leq p_j \leq 1 \forall j, \sum_{j=0}^{N-1} p_j = 1$.

To label the sample x_t with the inferred class $\hat{y}_t$, we select the class $\hat{y}_t$ associated with the highest score, which is given by $\hat{y}_t = j = \arg \max(\mathbf{p})$.

The likelihood that x_t belongs to the inferred class $\hat{y}_t$ is given by $p_j = \max(\mathbf{p})$. If the inferred class $\hat{y}_t$ matches the true label y_t (i.e., $\hat{y}_t \equiv y_t$), the model is said to have correctly classified the input.

The accuracy of a model is assessed by repeating this comparison across the entire test set. In addition to classification accuracy, it is important to consider the confidence of the model in its predictions.

Measuring prediction confidence is a complex task, and various methods and metrics have been proposed in the scientific literature. Some of these approaches estimate confidence based

on the likelihood vector $\mathbf{p}$. Metrics such as the normalized Gini-Simpson index (2), minimum entropy (3), Error L_2 Norm (EL2N) (4), and cross-entropy (5) have been shown to be effective for quantifying prediction confidence [26]–[29].

$$I_G(\mathbf{p}) = \frac{N}{N-1} \left(1 - \sum_{j=0}^{N-1} p_j^2 \right) \quad (2)$$

$$H_{\min} = -\log \max(\mathbf{p}) \quad (3)$$

$$\text{EL2N} = \|\mathbf{y} - \mathbf{p}\| \quad (4)$$

$$C(\mathbf{y}, \mathbf{p}) = - \sum_{i=0}^{N-1} y_i \log(p_i) \quad (5)$$

In Equations (2-5), $\mathbf{p} = f_\theta(x_t)$ represents the likelihood vector obtained during the testing phase for the input sample x_t , and $\mathbf{y}$ is the one-hot encoding of the true class y_t corresponding to that sample.

The confidence of a model can be viewed in two ways: confidence in the predefined true label and disagreement among labels [15], [18]. In the first, the measure reflects the model's certainty about its predicted class relative to the true label in a supervised learning scenario. High-confidence predictions indicate that the model is confident in correctly classifying the sample, with a large difference between the predicted probability for the chosen class and the others. Low-confidence predictions, on the other hand, suggest that the model is uncertain or close to making a misclassification.

This concept extends naturally to the multi-class setting, where the model assigns a probability distribution over all possible classes. In this context, confidence can be interpreted through a *one-versus-other* lens, in which the prediction for a given class is contrasted against the aggregated predictions of all other classes. Formally, let $\hat{y} = \arg \max\{\mathbf{p}\}$ be the predicted class for an input sample, where $\mathbf{p}$ is the output of the softmax, and p_i , the i -th element of $\mathbf{p}$, is the probability assigned to the class i . A high confidence score corresponds to a large margin between $p_{\hat{y}}$ and $p_j = \max_{j \neq \hat{y}}\{\mathbf{p}\}$, indicating that the model strongly prefers one class, $\hat{y}$ in this specific case, over all alternatives. Conversely, when this margin is small, the model expresses ambiguity among classes, reflecting low confidence. This interpretation is central to many multi-class classification reliability analyses, as it enables quantification of uncertainty based on the separability of the predicted class from the rest of the distribution.

Confidence can also be related to the degree of disagreement among different predicted labels. This is particularly suited whether the ground truth is not available, such as in an unsupervised-learning scenario. This disagreement, captured by metrics such as entropy (Equation (5)) or Gini-Simpson index (Equation (2)), indicates the level of uncertainty the model has in distinguishing between classes. Higher disagreement typically signifies that the model is unsure about which class the sample belongs to, potentially leading to more vulnerable or unreliable predictions.

We evaluated metrics that account for both the model's confidence in the true label (e.g., EL2N) and its internal

disagreements among predicted labels (e.g., Gini Index). By analyzing these metrics, we can better understand which samples expose the model's vulnerabilities and improve the efficiency of fault detection processes.

B. Error resilience in DNNs

Several contributions to the scientific literature have extensively demonstrated that classification systems are generally resilient to errors [30], [31]. This resilience is primarily due to the training algorithms that leverage datasets with a sufficiently high number of well-distributed labeled samples. This property also supports some degree of fault tolerance when considering hardware accelerators that perform the inference phase [2], [32].

Suppose a fault, or Single Event Transient (SET) occurs during the processing of a sample x_t from the test set, and let f'_θ represent the faulty model. The perturbed likelihood vector is then denoted as $\mathbf{p}' = f'_\theta(x_t)$. The predicted class corresponding to the perturbed model is $\hat{y}'_t = \arg \max(\mathbf{p}')$. This difference can lead to one of the following three outcomes:

- (a) *Masked Fault* – The fault has no effect on the final prediction, and the likelihood vector remains unchanged, i.e., $\mathbf{p}' \equiv \mathbf{p}$.
- (b) *Non-Critical Fault* – The fault alters the likelihood vector, but the prediction remains correct, i.e., $\mathbf{p}' \neq \mathbf{p}$, $\hat{y}'_t = \hat{y}_t$.
- (c) *Critical Fault* – The fault causes a significant alteration of the likelihood vector, leading to a misclassification, i.e., $\mathbf{p}' \neq \mathbf{p}$, $\hat{y}'_t \neq \hat{y}_t$.

Cases (a) and (b) are typically expected due to the inherent resiliency of neural networks. We refer to these as *masked* and *non-critical* faults, respectively. In contrast, case (c) refers to *critical* faults, which are more concerning as they lead to significant deviations in the model's behavior, potentially resulting in unacceptable outcomes, especially in safety-critical applications.

The complexity of both trained models and hardware accelerators makes it challenging to characterize the element-wise difference ϵ between the fault-free and faulty likelihood vectors, even from a statistical perspective. Characterizing these differences would require testing the model across a wide range of inputs and fault scenarios, which is computationally infeasible.

Now, consider that $\mathbf{p} = f_\theta(x_t)$ and let $\hat{y}_t = \arg \max(\mathbf{p})$ represent the predicted class. The model's confidence in its prediction can be categorized into the following cases:

- (i) *Reliable Inference*: The model is highly confident in its prediction, with the highest probability concentrated on the predicted class, i.e., $p_j \approx 1$ and $p_i \approx 0, \forall i \neq j$.
- (ii) *Unreliable Inference*: The model assigns similar probabilities to two classes, indicating a lack of confidence in the predicted classes, i.e., $p_j \approx p_l \approx \frac{1}{2}$, with $p_j > p_l$, and $p_i \approx 0, \forall i \neq j, l$.
- (iii) *Degenerate Case*: The model assigns nearly equal probabilities to all classes, i.e., $p_i \approx \frac{1}{N}, \forall i$.

From these scenarios, it can be inferred that, to cause a misclassification in the model, the error introduced by the

fault must be large when the model is confident (case (i)). On the other hand, a smaller error may suffice in cases (ii) and (iii). This suggests that, for robustness analysis, using samples where the model is confident (case (i)) would likely result in the fault being undetected. However, samples falling into cases (ii) and (iii) are more likely to trigger detectable faults, causing critical or non-critical failures. Thus, focusing on *challenging* samples – those in cases (ii) and (iii) – is more effective for evaluating the reliability of neural networks. This targeted approach can lead to a more efficient and comprehensive analysis of network vulnerabilities, particularly for fault detection tasks.

C. Metric-based Test Samples Ranking Method

As mentioned, earlier, conducting a full FI campaign on a DNN is often impractical due to the immense dimensionality of the fault space. To address this, SFI is typically employed: this significantly reduces the number of faults to be injected while maintaining statistical significance of results. Specifically, the number of injections can be computed by resorting to Equation (1), that allows the required number of faults to inject to decrease exponentially as the acceptable error margin increases. For instance, an accepted error margin of 1% with a confidence interval of 99% gives at least 16,640 faults injected for a highly parametrized network [6], [8], [33].

A significant challenge in traditional SFI lies in its choice of workload. The entire test set is often used, which proves to be suboptimal due to the following reasons:

- The effect of a fault is often uniform across a large portion of the test set, with critical faults impacting multiple images simultaneously.
- Many faults are masked, producing no observable effect during inference.

Given these observations, a reduced subset of images can often capture the majority of critical faults. Specifically, there are two main cases to consider:

- **Average Case**: Evaluates the reliability of the DNN under normal operational conditions, without introducing any bias in image selection. This typically involves random sampling from the test set.
- **Worst Case**: Focuses on the NN's performance under the most challenging conditions, necessitating a biased selection of images that are likely to stress the network beyond typical scenarios.

To address the *worst-case* scenario, we propose a metric-based approach to prioritize *challenging* samples, i.e., samples for which the *prediction confidence* is low, as indicated in Sec. IV-B. The scientific literature suggests, though formal proof is lacking, that these challenging samples, which activate the highest number of faults, are often associated with complex or borderline conditions [20], [22], [24]. In Section V we will empirically prove that such samples are crucial for uncovering latent vulnerabilities in the network.

The proposed framework for metric-based sample selection consists of the following steps:

- 1) **Metric Selection**: Identify a suitable metric to measure the complexity or challenging nature of the samples.

Common metrics include confidence scores, Gini Index, or distance-based measures in the feature space.

- 2) **Forward Pass:** Perform a forward pass on the entire test set to compute embeddings or probability vectors for each sample, as required by the chosen metric.
- 3) **Ranking:** Rank the test samples based on the computed metric values.
- 4) **Subset Selection:** Select the best n samples from the ranked list for inclusion in the FI workload.

An important strength of the proposed workload analysis and test selection method lies in its computational efficiency, especially when compared to fault injection-based reliability evaluation. The metric used for selecting informative test samples requires only a single forward pass per image, making it extremely lightweight. For most neural networks, computing this metric over the entire test set takes only a few seconds on a standard GPU.

In contrast, FIs campaigns incur a significant computational cost. Each fault from a predefined list must be injected individually, and inference must be repeated for each fault-image pair. For example, evaluating 16641 faults on a single image with ResNet-18 requires 16641 forward passes. Extending this to the full test set of 10000 images results in 166.41 million inferences. However, imagine that by leveraging our proposed metric, we can identify a reduced and more informative subset of, say, 1000 images. This lowers the required inferences for FI from 166.41 million to just 16.641 million, plus the 10000 forward passes needed to compute the golden outputs and the metric—achieving an order-of-magnitude reduction.

Crucially, our framework introduces no additional overhead. Since fault injection inherently requires computing the fault-free (golden) output for each input as a reference, the metric can be calculated during this initial forward pass. This integration makes the approach in practice cost-free. In our experiments, computing the metric for 10,000 test images took under 10 seconds, while performing fault injection on a single image with 16641 faults took over 40 minutes.

This stark contrast highlights the efficiency of our method, which enables scalable and targeted reliability evaluations without the need to exhaustively analyze the full test set. By focusing on critical samples and bypassing redundant ones, the proposed approach offers a practical and computationally viable alternative for assessing the robustness of AI systems. Consequently, the selected samples can be integrated into any FI framework—whether for CPUs, GPUs, or TPUs—with minimal modifications to the codebase.

A high-level pseudocode description of the aforementioned approach is provided in Algorithm 1: it requires the fault list F , the network model N , the test set T , the confidence metric m , and the number of test samples to be considered as workload n . Lines 2 to 7 construct the ranked test data set R : for each sample belonging to the test set, the forward pass is performed (line 4) and the sample, and the likelihood vector resulting from the model are stored in the R set for the subsequent analysis. Subsequently, at line 7, the R set undergoes sorting based on the confidence metric, which serves to prioritize samples with lower degrees of prediction confidence. A fault f from the fault list is injected into the

network at line 10; then at line 11, a sample is picked from R , and the forward pass is performed at line 12. Line 13 compares the output of the faulty network against the fault-free network to ascertain whether the injected fault can be detected. In the latter case, the fault is categorized as critical or non-critical, based on the predicted class (line 14). The counter of critical and non-critical faults is incremented accordingly (lines 15 and 17, respectively).

Algorithm 1 Pseudocode of the OpRA approach.

```

1: procedure OPRA( $F, N, T, m, n$ )
2:    $R \leftarrow \emptyset$ 
3:   for  $(t, l) \in T$  do
4:      $p \leftarrow N(t)$ 
5:      $R \leftarrow R \cup \{(t, p)\}$ 
6:   end for
7:    $R \leftarrow \text{sort}(R, m(\cdot))$ 
8:    $R \leftarrow R[:n]$ 
9:   for  $f \in F$  do
10:     $N' \leftarrow \text{injectFault}(f, N)$ 
11:    for  $(t, p) \in R$  do
12:       $p' \leftarrow N'(t)$ 
13:      if  $p' \neq p$  then
14:        if  $\arg \max(p') \neq \arg \max(p)$  then
15:           $\text{critical} \leftarrow \text{critical} + 1$ 
16:        else
17:           $\text{noncritical} \leftarrow \text{noncritical} + 1$ 
18:        end if
19:      end if
20:    end for
21:  end for
22:  return  $\text{noncritical}, \text{critical}$ 
23: end procedure

```

V. EXPERIMENTAL SETUP

TABLE I: HW-Agnostic FI campaign summary grouped by dataset. Accuracy refers to the fault-free scenario.

Network	Accuracy	Applied Faults	Critical Faults
ImageNet			
ConvNeXt (1)	82.52%	16 639	967
ConvNeXt (2)	82.47%	16 639	723
ConvNeXt (3)	81.80%	16 639	854
ImageNette			
VGG11	95.94%	16 641	115
CIFAR-100			
HCGNetA2	82.40%	16 623	2756
CIFAR-10			
ResNet18	93.52%	16 641	826
ResNet9	92.17%	16 639	1508
LeNet	79.36%	16 638	1158
VGG11	90.10%	16 641	819
VGG16	91.61%	16 641	894
MobileNetV2	91.72%	16 640	1340
MNIST			
ResNet18	99.63%	16 641	581
ResNet9	99.65%	16 639	527
LeNet	99.25%	16 638	484
VGG11	99.68%	16 641	661
VGG16	99.58%	16 641	555

In order to show the effectiveness of metric-based raking for test selection, we performed two sets of experiments: *Hardware agnostic FI* and *Hardware aware FI*. We addressed a classification task considering a relatively lightweight network. In the former case, a bit-flip in one of the weights of the DNN, mimics a fault in the system memory. These errors are relatively easy to simulate, allowing the analysis of a large amount of data. However, the single bit flip scenario represents the worst case in terms of online detectability, as the

resulting perturbation is often minimal and may not produce significant deviations in the network's behavior. In contrast, hardware-aware FI considers details of the DNN hardware accelerator. These simulations necessitate a greater computational effort compared to Hardware-agnostic FI. Therefore, an optimized test set is crucial for conducting FI. The hardware-aware experiments are not faster, if anything the opposite: for every convolution, our framework [34] needs to compute the Toeplitz's matrix of the inputs and process all the tiles separately, resulting in substantial simulation times.

The experimental setup included a selection of representative and diverse CNNs commonly used in computer vision tasks for embedded systems. The focus was on lightweight networks, as their reduced memory footprint makes them particularly well-suited for deployment in resource-constrained embedded environments [35]. We considered five distinct datasets. Specifically, we considered a LeNet-5 [36], two ResNet (with 9 and 18 layers, respectively) [37], two VGGs (with 11 and 16 layers, respectively) [25] trained on both MNIST [38] and CIFAR10 [39] dataset, a MobileNetV2 [40] trained on CIFAR10, an HCGNet [41] trained on CIFAR100 [39], a VGG11 trained on Imagenette, a curated subset of ImageNet consisting of 10 classes [42], and a ConvNeXt network [43], trained on ImageNet [44]. The FI on this final network, constrained by the available computational resources, was conducted on a subset of ImageNet consisting of 4,000 randomly sampled images, repeated three times using different seeds.

An overview of the networks being considered, and their test accuracy, is given in Tab. I.

Before proceeding to the discussion of the experiment, the following section will present some preliminary experiments conducted to shed useful light on the most appropriate metric to be adopted.

A. Selecting adequate metrics

In order to ascertain the existence of any discrepancy between the metrics examined in Section IV-B and to determine which metric more effectively conveys prediction confidence, two preliminary experiments were conducted.

The first experiment is designed to investigate potential discrepancies among confidence metrics, specifically those outlined in Equations (2-5). To achieve this, the respective test sets were randomly sampled, and the metrics were examined for any correlations between them, employing the Spearman correlation index [45]. The Spearman correlation coefficient is employed for the evaluation of the relationship between two variables, measuring the strength of monotonic relationships, whether linear or not, between two variables. In the absence of repeated data values, a perfect Spearman correlation of +1 or -1 is indicative of a perfect monotone function between variables. The results for LeNet-5, ResNet-18 and HCGNet CNNs are presented in Table II, Table III, and Table IV, respectively. As the index is close to 1 in each of the case, we can safely state that the considered metrics are equally effective for measuring the confidence during the inference phase. While different metrics analyze *challenging* samples in distinct

TABLE II: Spearman correlation index between (2-5) metrics, for the LeNet-5 CNN on MNIST.

H_{min}	0.99		
EL2N	0.99	0.99	
$C(\mathbf{y}, \mathbf{p})$	0.99	0.99	0.99
	$I_G(\mathbf{p})$	H_{min}	EL2N

TABLE III: Spearman correlation index between (2-5) metrics, for the ResNet-18 CNN on CIFAR-10

H_{min}	0.86		
EL2N	0.85	0.99	
$C(\mathbf{y}, \mathbf{p})$	0.98	0.85	0.85
	$I_G(\mathbf{p})$	H_{min}	EL2N

ways, the strong correlations among metrics imply flexibility in their selection. Depending on the specific requirements of an application or dataset, any of these metrics can be confidently employed for evaluating challenging samples.

The objective of the second experiment is to provide empirical evidence that a lower prediction confidence on a sample point, x_t , is more likely to be beneficial in identifying both non-critical and critical faults. To achieve this, we performed a fault injection campaign. Following the approach from [4], we exploited Equation (1) to compute the number of faults to be injected to achieve 1% error and 99% confidence in the estimation. According to this, we injected about 16640 faults in each network (the precise number depends on the equation (1) and is shown in Tab. I).

Figures 3-6 plot the number of critical faults detected for each of the images in the test dataset, varying the GINI-index (Equation (2)), the Cross-entropy (Equation (5)) or the EL2N (Equation (4)), respectively. Specifically, the x-axis represents the metric value associated with each image, while the y-axis denotes the number of critical faults activated by that specific image. The clear result is that although the number of detected critical faults is uniformly distributed across images, certain images clearly allow more faults to be detected, suggesting that certain samples may be better suited for fault detection, or are inherently more sensitive to fault activations. These images often correspond to specific confidence metric values, e.g., ≈ 0.5 for the Gini-Index, ≈ 1 for the Cross-Entropy, and ≈ 0.7 for the EL2N.

The experimental results obtained substantiate our hypothesis, which posits that as the model's prediction confidence diminishes, there is an increased propensity for misclassification, thereby leading to an elevated fault detection rate. To better illustrate this, we sorted the images based on their respective metric values and grouped them into subsets of 500 samples each. For each subset, Figure 7a and Figure 7b report the number of critical faults detected for each image in that subset. The x-axis reports the subsets in order, with the leftmost subset comprising test samples with the larger EL2N (4) value.

TABLE IV: Spearman correlation index between (2-5) metrics, for the HCGNet CNN on CIFAR-100

H_{min}	0.99		
EL2N	0.93	0.91	
$C(\mathbf{y}, \mathbf{p})$	0.93	0.92	0.99
	$I_G(\mathbf{p})$	H_{min}	EL2N

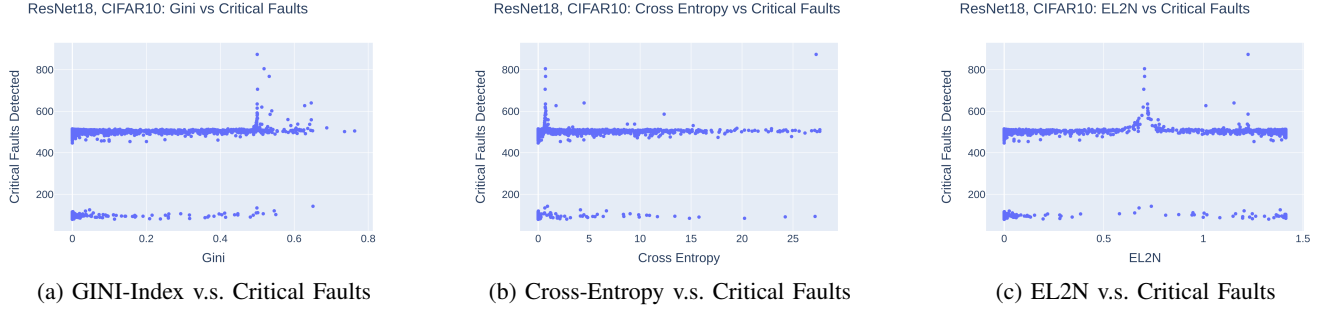


Fig. 3: Confidence v.s. fault detected for ResNet-18 on CIFAR-10

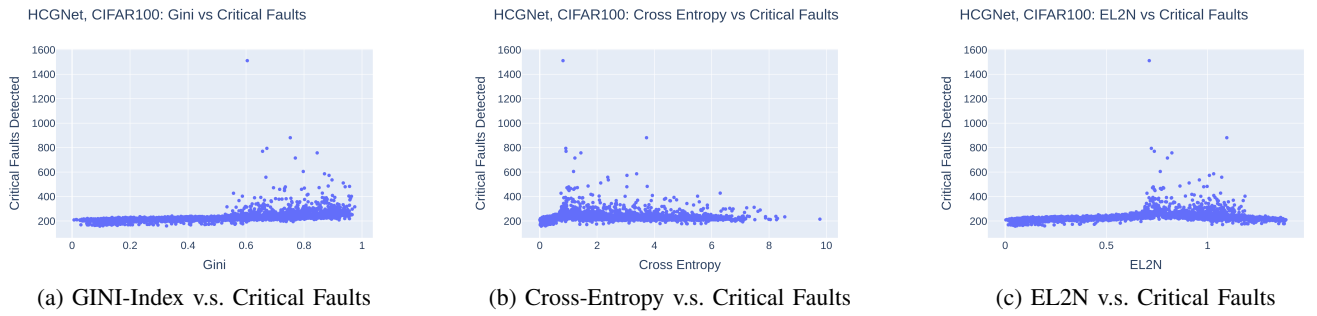


Fig. 4: Confidence v.s. fault detected for HGCNet on CIFAR-100

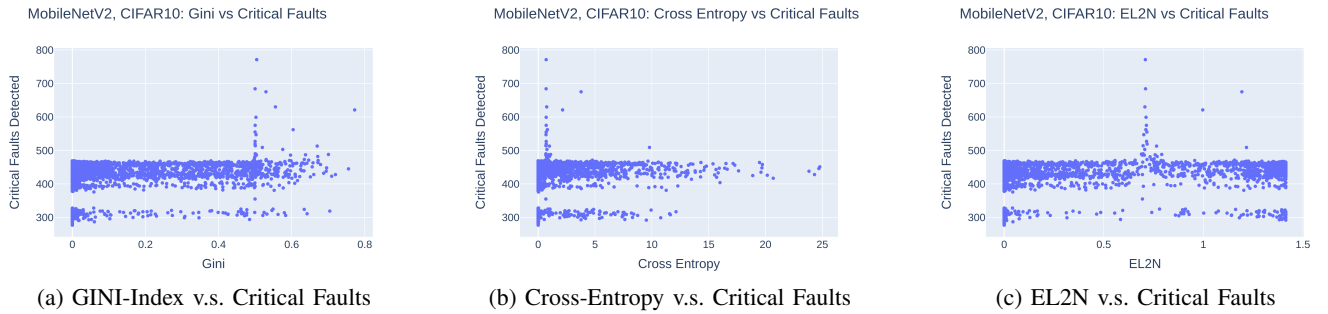


Fig. 5: Confidence v.s. fault detected for MobileNet on CIFAR10

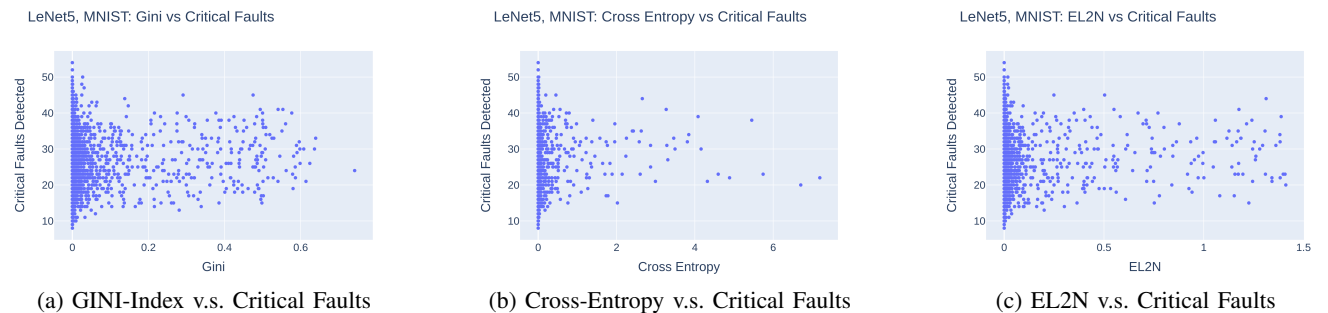


Fig. 6: Confidence v.s. fault detected for LeNet on MNIST

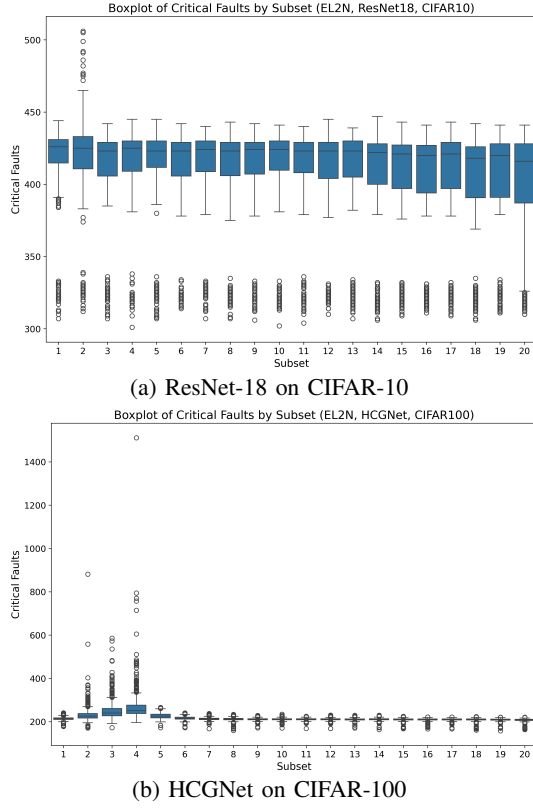


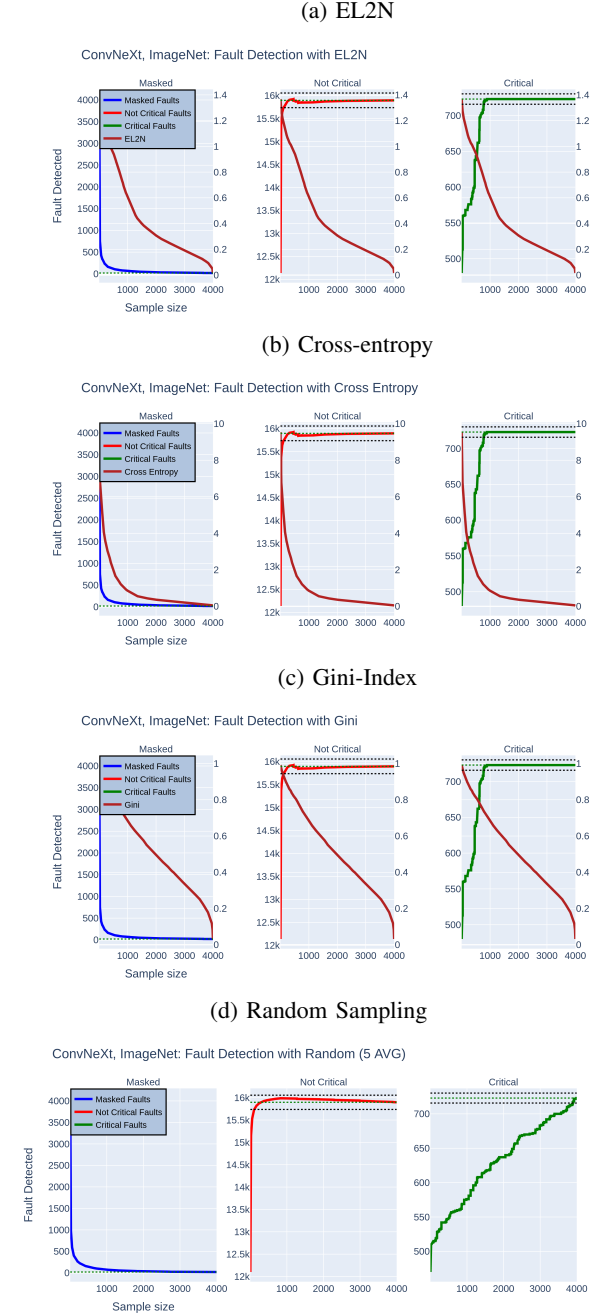
Fig. 7: BoxPlot of fault captured vs. metric values, for a subset of 500 images sorted by confidence metric value.

Despite the evident differences among the considered CNNs and datasets, a comparable tendency can be observed. The detection rate across subsets is nearly identical for non-challenging images. However, subsets that collect images with values close to the threshold (≈ 0.7 for EL2N) exhibit outliers, i.e., images that activate a significantly higher number of faults compared to the typical distribution. This confirms that the prediction confidence can be exploited to identify critical samples regardless of the dataset or the CNN. It is noteworthy that the outcomes for ResNet18 may be indicative of a more optimal correspondence between EL2N values and critical faults, a phenomenon that can be attributed to the relatively uncomplicated nature of CIFAR-10 and the potential for a more uniform distribution of critical samples. In contrast, the more diffuse distribution observed in the HCGNet plot might imply greater variability in the distribution of critical images, a consequence of the augmented complexity inherent in CIFAR-100. Notwithstanding the aforementioned observations, it is noteworthy that for both models and datasets, the approximate EL2N value of 0.7 appears to be the most critical for images.

B. Hardware-Agnostic Experiments

The proposed approach makes use of *fault detection curves*, which plot the cumulative number of detected faults against the number of images used for detection. Notably, the classification of each fault is cumulative and unidirectional, meaning a fault cannot be reassigned to a less severe category once classified into a more severe one. These curves facilitate the

Fig. 8: Metric values vs. fault identified by each image for ConvNext on ImageNet.

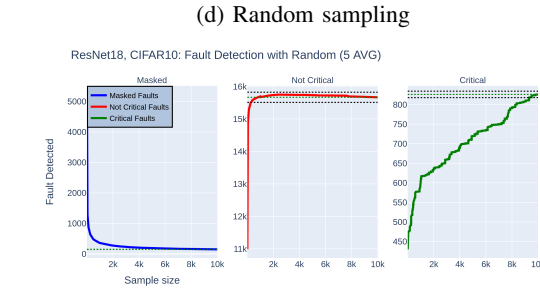
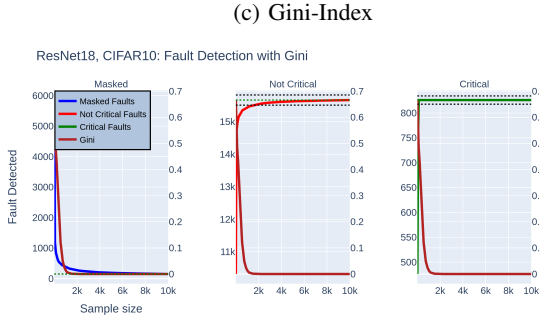
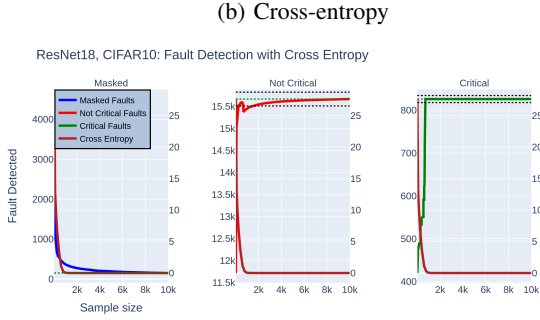
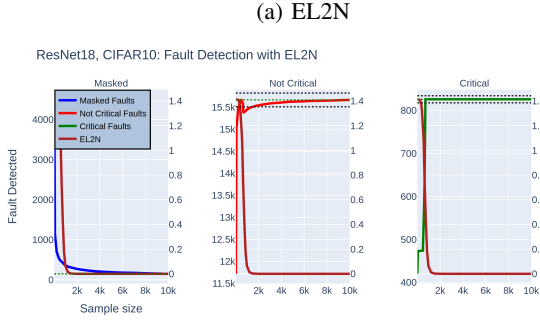


comparative analysis of different metrics and their influence on fault activation and detection.

Experiments show how sorting samples based on metrics could be beneficial for this task. In our graphs, blue curves/-points represent masked faults, red curves/points indicate non-critical faults, and green curves/points correspond to critical faults. Red-brick curves show the metric values. Dotted horizontal green lines represent the final values, the dotted black lines are placed at $\pm 1\%$ of the final values.

Using a random approach, nearly all images are required to detect all the critical faults. This random analysis was repeated with five different random image sorting, with the curves representing the average of the five runs.

Fig. 9: Metric values vs. fault identified by each image for ResNet-18 on CIFAR-10 (Gini index, Cross Entropy and EL2N).



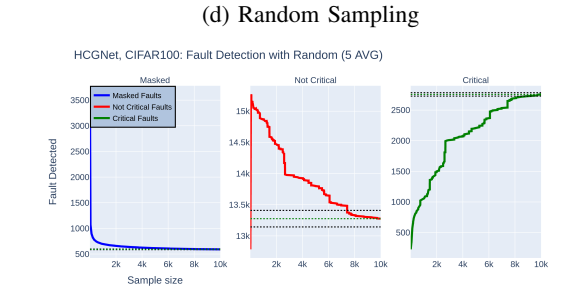
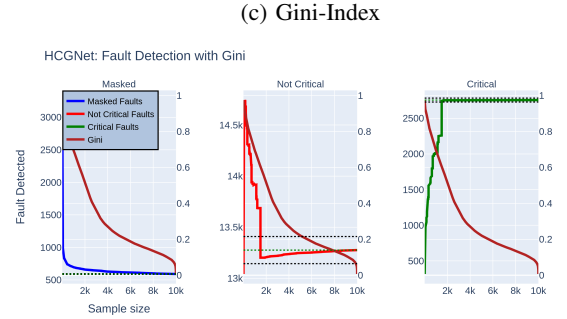
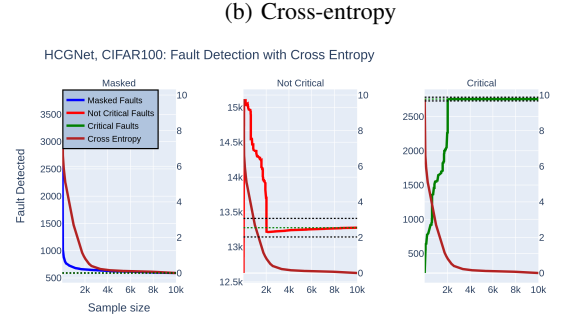
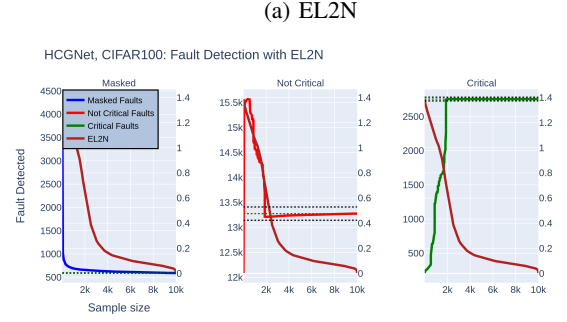
When images are sorted and the detection algorithm is applied, we observe that after an initial transitory phase, the sorted metric values quickly approach zero and then stabilize at a *plateau*.

It is evident that all images activating critical faults fall within the transitory phase of the curve. Once the critical faults stabilize (i.e., all critical faults are detected), the metric values approach zero or quickly decrease, and the plateau is reached. This behavior is consistent across all metrics computed at the probability vector level;

Not all samples are required for detecting critical faults; rather, only the *challenging* samples seem sufficient to find all the critical faults.

Regarding the number of *challenging* samples to deploy, it

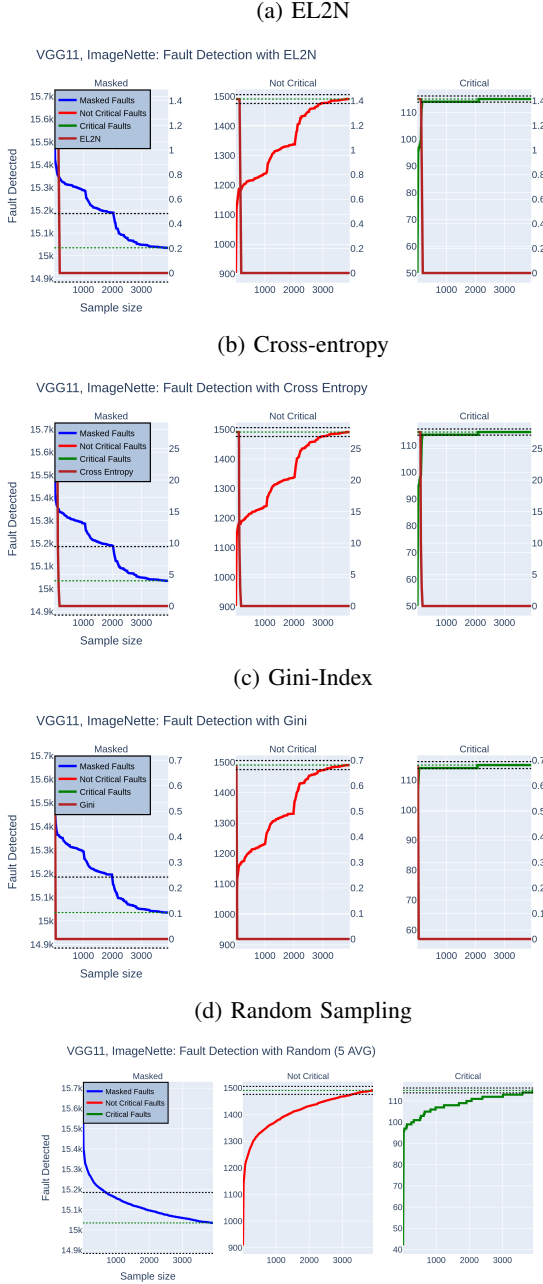
Fig. 10: Metric values vs. fault identified by each image for HGCNet on CIFAR-100.



heavily depends on the dataset and network: for the ResNet18 on CIFAR10, less than 100 test samples are enough to fully discover all critical faults. For the LeNet architecture, instead, about 8,000 samples over 10,000 are necessary to discover the 95% of the critical faults. However, this network is heavily resilient, and there are no considerable outliers in the distribution of images that discover critical faults. Table V provides an overview of the number of samples required to detect all critical faults for at least one image. The table clearly demonstrates the advantage of sorting images by different metrics compared to a non-sorted approach.

Knowing this, to estimate the required number of samples for the FI, we may employ several strategies, detailed in Tables V VI. The first rely on looking only at the metric values

Fig. 11: Metric values vs. fault identified by each image for VGG11 on Imagenette.



prior to the FI.

As a first approximation, we define a sample as challenging if it lies outside the so-called plateau region. The plateau is characterized by regions where the absolute value of the first derivative of a given metric function is close to zero or remains consistently below a defined threshold (e.g., $\epsilon = 5 \times 10^{-5}$). This threshold should be adapted based on the specific behavior of the metric under consideration, such as the red curve in our analysis (Fig.8-10). Samples preceding the plateau are deemed of interest for FI purposes. Selecting only those samples for which the metric deviates from zero provides an upper bound on the number of inputs required for effective FI. Alternatively, one may use the hotspot regions (see Fig. 3-6) to filter inputs based on specific metric values.

For instance, in the case of the EL2N metric, we may select samples within a tolerance window of $\epsilon = 0.3$ around the distribution's central mode. This value, while illustrative of the method's potential, should be tuned based on the specific dataset and metric distribution.

Using this selection strategy enables the identification of nearly all faults deemed critical for at least one input in the dataset, while significantly reducing the size of the test set. A summary of these results is presented in Table VI, where the random baseline is computed as the average over five randomly selected subsets of the same size as the EL2N-based selection.

A second strategy involves performing FI in an online fashion. In this case, a fault detection counter is maintained to monitor the number of unique faults uncovered during the campaign.

Detecting the knee point of a curve is quite tricky, but the *Kneedle* approach [46] provides a valuable method for addressing this challenge. It is based on the notion of curvature, and on the observation that the points of maximum curvature in a discrete dataset are approximately the set of points in a curve that are local maxima (or minima) for the curve when the curve is rotated clockwise (or counterclockwise) through the line formed by the absolute minimum and maximum points. Nevertheless, to select the proper number of samples, the Kneedle algorithm from [46] has to be adapted to work online, iteratively refining the knee point as soon as new samples and detected faults pairs are available while performing the FI campaign. This stopping criterion ensures that the FI experiment continues only as long as new faults are being detected, optimizing computational resources while maintaining statistical relevance. This approach is guaranteed to stabilize due to the nature of the fault classification algorithm: once a fault is identified as critical, its status remains unchanged. As a result, the critical fault curve is, by definition, monotonic. For both the approaches, we analyzed the number of simulations to be performed. An overview is given in the last columns of Tables VI, where we evaluated the time saving by using challenging samples only to estimate the number of critical faults/performing the FI campaign. To simplify, we considered a time of 1 millisecond per inference, independent of the network being used. This is not strictly true in practice, since inference time depends on the specific network, dataset and hardware used, but we simplify here. The total time required to test all samples (100%) is given by: $\text{Total Time} = 1 \text{ ms} \times \# \text{TestImages} \times 16,640 \text{ faults}$. This lead to a total time of about 1.93 days for CIFAR10, CIFAR100 and MNIST (with 10,000 test images), 0.76 days for Imagenette (with 3,925 test images) and 0.77 days for ImageNet (considering a subset of 4000 test images). The time needed to perform the reliability assessments scales with the sample size. Specifically, for any given sample size, the time is calculated as $\text{Time} = \text{Total Time} \times \left(\frac{\% \text{ of Samples}}{100} \right)$.

C. Hardware-aware Experiments

To evaluate whether our methodology remains effective on real hardware, we performed hardware-aware fault injection using the SAFFIRA framework [34]. SAFFIRA provides a

TABLE V: Percentage of test images required to detect all critical faults, per network and metric. Metrics rank samples in descending order.

Dataset	Network	Cross Entropy	Gini	EL2N	Random (5 avg)
ImageNet	ConvNext (1)	22.05%	24.07%	20.00%	99.33%
	ConvNext (2)	22.22%	20.06%	20.06%	99.72%
	ConvNext (3)	23.15%	20.95%	21.18%	99.65%
ImageNette	VGG11	53.4%	52.8%	53.8%	99.5%
CIFAR-100	HCGNet	20.92%	18.99%	19.14%	99.12%
CIFAR-10	MobileNetV2	9.30%	2.08%	9.23%	98.54%
	ResNet18	6.79%	0.90%	6.73%	99.30%
	ResNet9	7.96%	0.91%	7.94%	99.54%
	VGG16	8.48%	6.43%	8.46%	97.59%
	VGG11	10.58%	2.24%	10.56%	97.99%
	LeNet5	52.19%	50.01%	53.10%	99.36%
MNIST	VGG16	0.52%	2.17%	0.52%	90.50%
	VGG11	0.36%	1.54%	0.34%	99.14%
	ResNet18	0.40%	0.09%	0.40%	90.35%
	ResNet9	0.36%	0.12%	0.35%	97.65%
	LeNet5	97.39%	97.39%	97.60%	99.78%

logic-level simulation of a systolic array, enabling realistic modeling of faults. During matrix multiplications, it considers faults that may affect the registers within each processing element. Specifically, every time a forward pass is executed through a convolutional layer, the computation is redirected to SAFFIRA, which simulates the convolution while accurately accounting for the impact of any injected faults.

Due to the high computational cost of simulation, fault injection was limited to transient faults occurring in the first layer of the neural network. Additionally, to further reduce simulation time, only a subset of input stimuli was used. In our experiments, we used 100 input images. This choice is linked to simulating a real environment in which to run *fault-detection tests*; when employing this technique on field to perform fault-detection, only a small subset of the test set can be used for two main reasons: (i) memory footprint, the dataset must be small enough to fit on the dedicated hardware and (ii) offline timing, the time spent doing fault-detection must be small enough not to undermine the functionality of the system. As shown later, in table VI, often it is possible to detect most faults with a small portion of the test set. We find that 100 images can be a good compromise between test latency and fault detection. A fault affecting a PE causes an error on the register, which is then propagated to other PEs. Furthermore, for the experiments, we used a OS systolic array. Each of the registers on the PE has an equal probability of being affected by a fault. We injected 384 faults, a number determined according to the statistical guidelines proposed in [4] and in Equation 1. For the experiments, we used a subset of the networks illustrated above. More specifically, we used three networks: a ResNet20, a VGG11, and a MobileNetV2. Each fault injection was carried out 11 times: one time with the metric-based ranking and the other 10 times with different random subsets of images. This choice was made to properly demonstrate the effectiveness of the ranking system.

Figure 12 shows the results of the fault detection algorithm. The three subfigures 12a, 12b, 12c show the cumulative number of faults detected for each sample of the dataset

processed. Each graph shows three curves: (i) the dotted blue line is the number of faults detected by averaging the results of the 10 random rankings experiments; (ii) the orange dash-dotted line shows the *best* random: the run with the most faults detected; finally, (iii) the solid green line shows the faults detected using the metric-based ranking. It is visible that the fault discovery trends are very similar regardless of the network. The average of the random rankings shows that the maximum number of faults discovered at the end of the test is relatively small. In fact, looking at the solid line, we can see that the fault discovery with the metric-based ranked set can detect more than double the number of faults. Finally, in the “best” random sample among the 10 used, even though the performance is better than the average, the number of faults discovered at the end of the inferences is always smaller than that discovered with the metric-based ranking. It is worth noticing the presence of images with high metric values in the best random ranking. For instance, the best random ranking for ResNet20 had 4 images belonging to the metric-based one, the highest among all the random rankings. Similarly, the best random ranking for VGG11 had 2 images belonging to the best ranking. The ranking-based approach methodology allows a much faster fault detection: using the metric-based ranking, not only permits reaching a plateau in the fault detected, but this is also reached using just a handful of samples.

To give concrete figures, let us compare the experiments done with the ResNet20 (Figure 12a). On average (dotted line), only about 90 faults are detected using 100 samples. The best random ranking (dash-dotted line) shows a plateau with 150 faults detected after 40 samples. Although this ranking performed much better than average, the metric-based ranking (solid line) shows 180 faults discovered using less than 40 samples in total, implying a faster convergence and more efficient fault discovery.

VI. CONCLUSION

This study presents an empirical approach to evaluating the reliability of DNNs by prioritizing *challenging* samples identified through confidence metrics such as EL2N during FI campaigns. We demonstrate that this prioritization significantly reduces the number of samples required to achieve comprehensive fault coverage, thereby lowering computational costs. Our experiments, conducted in both hardware-agnostic and hardware-aware settings, confirm the effectiveness of the method across diverse architectures and datasets. For example, fewer than 100 samples sufficed to fully assess the criticality of ResNet-18, while fewer than 2,000 samples were adequate for more complex models like HCGNet trained on CIFAR-100. The hardware-aware evaluations further validate the approach’s applicability to realistic implementations, highlighting its robustness and versatility. This is particularly important because fault injection simulations in hardware-aware scenarios are computationally intensive and time-consuming; thus, reducing the sample set directly translates into substantial savings in simulation time and resources. Moreover, the introduced *fault detection curves* offer valuable visual insights into fault discovery dynamics, illustrating the rapid convergence of

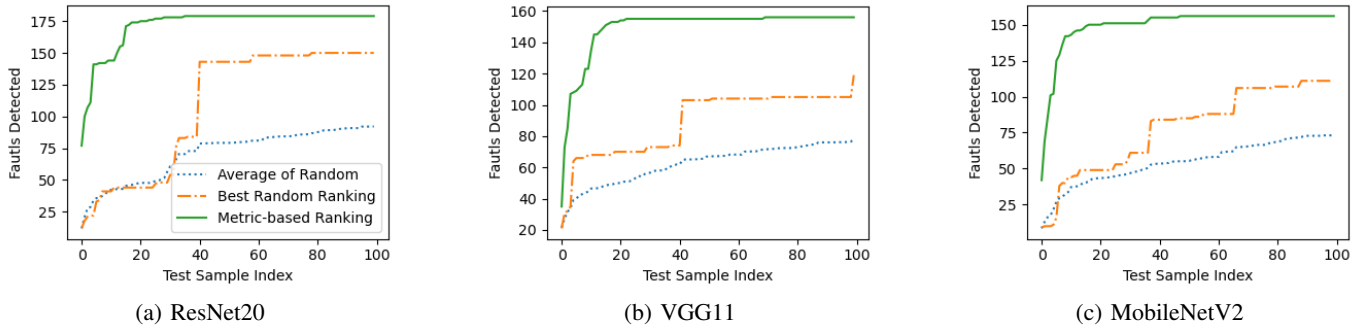


Fig. 12: Hardware-aware fault detection experiments results. The faults are injected in the first layer of the neural networks.

critical fault detection when employing metrics-based sample prioritization. This not only accelerates FI campaigns but also enhances understanding of the interplay between test sample characteristics and fault activation.

The proposed methodology constitutes a meaningful advancement in DNN reliability assessment by enabling efficient and targeted FI campaigns. Future work will focus on extending the framework to additional fault models and more complex DNNs, thereby broadening its scope and impact. Additionally, we intend to explore applications beyond classification, including object detection, image segmentation, and natural language processing tasks. This will require identifying suitable confidence metrics tailored to each task, with the potential to generalize the approach across a wide range of deep learning domains.

ACKNOWLEDGMENTS

This work has been carried out within the ANR France 2030 AdaptING project “ANR-23-PEIA-0009”.

REFERENCES

- [1] A. Ruospo *et al.*, “A survey on deep learning resilience assessment methodologies,” *Computer*, vol. 56, no. 2, pp. 57–66, 2023.
- [2] A. Ruospo *et al.*, “Assessing Convolutional Neural Networks Reliability through Statistical Fault Injections,” in *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2023, pp. 1–6.
- [3] G. Yu *et al.*, “A Survey on Failure Analysis and Fault Injection in AI Systems,” 2024.
- [4] R. Leveugle *et al.*, “Statistical fault injection: Quantified error and confidence,” in *Automation & Test in Europe Conference & Exhibition 2009 Design*, 2009, pp. 502–506.
- [5] G. Di Natale *et al.*, *Cross-Layer Reliability of Computing Systems*. USA, 2020.
- [6] S. Pappalardo *et al.*, “A Fault Injection Framework for AI Hardware Accelerators,” in *2023 IEEE 24th Latin American Test Symposium (LATS)*, Veracruz, Mexico, 2023.
- [7] N. Bellarmino *et al.*, “Investigating on Gradient Regularization for Testing Neural Networks,” in *LOD 2024 : 10th International Conference on machine Learning, Optimization and Data science*, Riva de Sole, Italy, Sep. 2024.
- [8] A. Ruospo *et al.*, “A Survey on Deep Learning Resilience Assessment Methodologies,” *Computer*, vol. 56, no. 2, pp. 57–66, Feb. 2023.
- [9] 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)*, May 2023, pp. 181–186.
- [10] H. T. Kung *et al.*, “Systolic arrays (for vlsi),” in *Sparse Matrix Proceedings 1978*, vol. 1. Society for industrial and applied mathematics Philadelphia, PA, USA, 1979, pp. 256–282.
- [11] S. K. Rao *et al.*, “Regular iterative algorithms and their implementation on processor arrays,” *Proceedings of the IEEE*, vol. 76, no. 3, pp. 259–269, 1988.
- [12] J. Kaplan *et al.*, “Scaling laws for neural language models,” *arXiv preprint arXiv:2001.08361*, 2020.
- [13] V. Sze *et al.*, “Efficient Processing of Deep Neural Networks: A Tutorial and Survey,” *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.
- [14] W. Sarle, “Neural networks and statistical models,” 1994.
- [15] Q. Hu *et al.*, “Evaluating the Robustness of Test Selection Methods for Deep Neural Networks,” 2023.
- [16] T. Byun *et al.*, “Input Prioritization for Testing Neural Networks,” in *2019 IEEE International Conference On Artificial Intelligence Testing (AITest)*, 2019, pp. 63–70.
- [17] W. Ma *et al.*, “Test Selection for Deep Learning Systems,” *ACM Transactions on Software Engineering and Methodology*, vol. 30, no. 2, pp. 1–22, 2021.
- [18] C. Zhao *et al.*, “Can test input selection methods for deep neural network guarantee test diversity? A large-scale empirical study,” *Information and Software Technology*, vol. 150, p. 106982, 2022.
- [19] T. Borovicka *et al.*, “Selecting Representative Data Sets,” in *Advances in Data Mining Knowledge Discovery and Applications*. IntechOpen, 2012.
- [20] W. Li *et al.*, “Adversarial Testing: A Novel On-Line Testing Method for Deep Learning Processors,” in *2023 IEEE 32nd Asian Test Symposium (ATS)*, 2023, pp. 1–6.
- [21] S. Kundu *et al.*, “Special Session: Effective In-field Testing of Deep Neural Network Hardware Accelerators,” in *2022 IEEE 40th VLSI Test Symposium (VTS)*, 2022, pp. 1–4.
- [22] S. Kundu *et al.*, “Toward Functional Safety of Systolic Array-Based Deep Learning Hardware Accelerators,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 29, no. 3, pp. 485–498, 2021.
- [23] F. Meng *et al.*, “A Self-Test Framework for Detecting Fault-induced Accuracy Drop in Neural Network Accelerators,” in *Proceedings of the 26th Asia and South Pacific Design Automation Conference*. Tokyo Japan: ACM, 2021, pp. 722–727.
- [24] F. Meng *et al.*, “Exploring image selection for self-testing in neural network accelerators,” in *2022 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2022, pp. 345–350.
- [25] K. Simonyan *et al.*, “Very Deep Convolutional Networks for Large-Scale Image Recognition,” 2015.
- [26] M. Barbareschi *et al.*, “Investigating the Resilience Source of Classification Systems for Approximate Computing Techniques,” *IEEE Transactions on Emerging Topics in Computing*, p. 12, 2024.
- [27] R. König *et al.*, “The Operational Meaning of Min- and Max-Entropy,” *IEEE Transactions on Information Theory*, vol. 55, no. 9, pp. 4337–4347, 2009.
- [28] M. Paul *et al.*, “Deep Learning on a Data Diet: Finding Important Examples Early in Training,” in *Advances in Neural Information Processing Systems*, vol. 34. Curran Associates, Inc., 2021, pp. 20 596–20 607.
- [29] A. Mao *et al.*, “Cross-Entropy Loss Functions: Theoretical Analysis and Applications,” in *Proceedings of the 40th International Conference on Machine Learning*. PMLR, 2023, pp. 23 803–23 828.
- [30] V. K. Chippa *et al.*, “Analysis and characterization of inherent application resilience for approximate computing,” in *Proceedings of the 50th Annual Design Automation Conference on - DAC '13*. Austin, Texas: ACM Press, 2013, p. 1.
- [31] I. J. Goodfellow *et al.*, “Explaining and Harnessing Adversarial Examples,” 2015.

TABLE VI: Random vs EL2N-based Detection.

Network	Method	% of Samples	Faults Detected	Time (days)	% of Critical Faults Detected
CIFAR100					
HCGNet	Central Mode	11.94%	2606	0.23	94.56%
	Random Detection	11.94%	1452	0.23	52.69%
	Pre-Plateau Detection	29.11%	2756	0.56	100.00%
	Full Dataset	100%	2756	1.93	100.00%
ImageNet					
ConvNext	Central Mode	30.10%	929	0.23	96.07%
	Random Detection	30.10%	738	0.23	76.32%
	Pre-Plateau Detection	53.87%	967	0.41	100.00%
	Full Dataset	100%	967	0.77	100.00%
ConvNext	Central Mode	29.07%	698	0.22	96.54%
	Random Detection	29.07%	595	0.22	82.30%
	Pre-Plateau Detection	54.07%	723	0.42	100.00%
	Full Dataset	100%	723	0.77	100.00%
ConvNext	Central Mode	30.23%	732	0.23	85.71%
	Random Detection	30.23%	702	0.23	82.20%
	Pre-Plateau Detection	91.25%	854	0.70	100.00%
	Full Dataset	100%	854	0.77	100.00%
CIFAR10					
LeNet5	Central Mode	21.18%	1132	0.41	97.75%
	Random Detection	21.18%	774	0.41	66.79%
	Pre-Plateau Detection	79.45%	1158	1.53	100.00%
	Full Dataset	100%	1158	1.93	100.00%
ResNet18	Central Mode	1.81%	822	0.04	99.52%
	Random Detection	1.81%	503	0.04	60.90%
	Pre-Plateau Detection	14.47%	826	0.28	100.00%
	Full Dataset	100%	826	1.93	100.00%
VGG11	Central Mode	1.40%	793	0.03	96.83%
	Random Detection	1.40%	518	0.03	63.25%
	Pre-Plateau Detection	16.41%	819	0.32	100.00%
	Full Dataset	100%	819	1.93	100.00%
VGG16	Central Mode	0.98%	891	0.02	99.66%
	Random Detection	0.98%	512	0.02	57.27%
	Pre-Plateau Detection	13.33%	894	0.26	100.00%
	Full Dataset	100%	894	1.93	100.00%
MobileNetV2	Central Mode	3.37%	1306	0.07	97.46%
	Random Detection	3.37%	640	0.07	47.76%
	Pre-Plateau Detection	23.28%	1340	0.45	100.00%
	Full Dataset	100%	1340	1.93	100.00%
ResNet9	Central Mode	1.77%	1327	0.03	88.02%
	Random Detection	1.77%	621	0.03	41.19%
	Pre-Plateau Detection	16.27%	1508	0.31	100.00%
	Full Dataset	100%	1508	1.93	100.00%
MNIST					
ResNet18	Central Mode	0.26%	581	0.01	100.00%
	Random Detection	0.26%	483	0.01	83.13%
	Pre-Plateau Detection	3.64%	581	0.07	100.00%
	Full Dataset	100%	581	1.93	100.00%
VGG11	Central Mode	0.15%	645	0.00	97.58%
	Random Detection	0.15%	511	0.00	77.31%
	Pre-Plateau Detection	3.61%	661	0.07	100.00%
	Full Dataset	100%	661	1.93	100.00%
VGG16	Central Mode	0.26%	551	0.01	99.28%
	Random Detection	0.26%	515	0.01	92.79%
	Pre-Plateau Detection	3.28%	555	0.06	100.00%
	Full Dataset	100%	555	1.93	100.00%
ResNet9	Central Mode	0.33%	527	0.01	100.00%
	Random Detection	0.33%	518	0.01	98.29%
	Pre-Plateau Detection	3.11%	527	0.06	100.00%
	Full Dataset	100%	527	1.93	100.00%
LeNet5	Central Mode	0.79%	342	0.02	70.66%
	Random Detection	0.79%	291	0.02	60.12%
	Pre-Plateau Detection	9.01%	436	0.17	90.08%
	Full Dataset	100%	484	1.93	100.00%
ImageNet					
VGG11	Central Mode	0.23%	97	0.00	84.35%
	Random Detection	0.23%	87	0.00	75.65%
	Pre-Plateau Detection	4.38%	114	0.03	99.13%
	Full Dataset	100%	115	0.76	100.00%

- [32] M. H. Ahmadilivani *et al.*, “Special Session: Approximation and Fault Resiliency of DNN Accelerators,” in *2023 IEEE 41st VLSI Test Symposium (VTS)*. IEEE, 2023, pp. 1–10.
- [33] A. Ruospo *et al.*, “Image Test Libraries for the on-line self-test of functional units in GPUs running CNNs,” in *2023 IEEE European Test Symposium (ETS)*, 2023, pp. 1–6.
- [34] M. Taheri *et al.*, “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.
- [35] I. Miro-Panades *et al.*, “A 772uj/frame imagenet feature extractor accelerator on hd images at 30fps,” in *2024 IEEE Asia Pacific Conference on Circuits and Systems (APCCAS)*, 2024, pp. 25–29.
- [36] Y. Lecun *et al.*, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.
- [37] K. He *et al.*, “Deep Residual Learning for Image Recognition,” in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Las Vegas, NV, USA: IEEE, 2016, pp. 770–778.

- [38] Y. LeCun *et al.*, “MNIST Handwritten digit database,” <http://yann.lecun.com/exdb/mnist/>, 1998.
- [39] A. Krizhevsky, “Learning Multiple Layers of Features from Tiny Images,” Tech. Rep., 2009.
- [40] M. Sandler *et al.*, “Mobilenetv2: Inverted residuals and linear bottlenecks,” 2019.
- [41] C. Yang *et al.*, “Gated Convolutional Networks with Hybrid Connectivity for Image Classification,” 2019.
- [42] J. Howard, “Imagenette: A smaller subset of 10 easily classified classes from the imagenet dataset,” <https://github.com/fastai/imagenette>, 2019, accessed: 2025-06-09.
- [43] Z. Liu *et al.*, “A convnet for the 2020s,” 2022.
- [44] J. Deng *et al.*, “Imagenet: A large-scale hierarchical image database,” in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255.
- [45] C. Spearman, “The Proof and Measurement of Association between Two Things,” *The American Journal of Psychology*, vol. 15, no. 1, pp. 72–101, 1904.
- [46] V. Satopaa *et al.*, “Finding a “Kneedle” in a Haystack: Detecting Knee Points in System Behavior,” in *2011 31st International Conference on Distributed Computing Systems Workshops*, 2011.



Nicolò Bellarmino received the PhD cum Laude in Computer and Control Engineering at Politecnico di Torino in 2024. His main research interests are Machine Learning, Data Analysis, and AI systems and their application to real-world cases. He worked in Machine Learning applied to device testing and reliability since 2020. He is part of IEEE-HKN honor society.



Salvatore Barone is a Tenured Assistant Professor at the Department of Computer Science and Technologies, Pegaso University, Naples, Italy. He received the PhD in Information Technologies and Electrical Engineering in 2022 from the University of Naples Federico II, Italy. His main research fields are high-performance computing systems, artificial intelligence, safety Critical systems, railway systems, and embedded systems based on the FPGA technology.



Salvatore Pappalardo is currently a PhD student at Ecole Centrale de Lyon. He received his master degree in Computer Engineering (with a focus on control theory) from Politecnico di Torino in October 2022. His main research interests are reliable hardware design, approximate computing and neural network in embedded systems.



Alberto Bosio received the PhD in Computer Engineering from Politecnico di Torino in Italy in 2006. Currently he is a full professor at the Ecole Centrale de Lyon, Institute of Nanotechnology in France. He published articles spanning diverse disciplines, including testing, verification, reliability, approximate computing and emerging technologies. He is an IEEE member and the Chair of the European Test Technical Technology Council (eTTTC).



Riccardo Cantoro received the MS degree and the PhD in computer engineering from Politecnico di Torino, Italy, in 2013 and 2017, respectively. He is currently an Associate Professor with the Department of Computer Engineering of the same university. His research interests include software-based functional testing of SoCs and memories, and machine learning applied to test and diagnosis. He is a member of the IEEE.