POLITECNICO DI TORINO
Repository ISTITUZIONALE

Reliability of Deep Neural Networks: Impact and Open Issues

Original
Reliability of Deep Neural Networks: Impact and Open Issues / Ruospo, A., Pappalardo, S., Turco, V., Bosio, A., Sanchez, E.. - In: IEEE DESIGN & TEST. - ISSN 2168-2356. - ELETTRONICO. - 42:3(2025), pp. 14-25. [10.1109/MDAT.2025.3544125]

Availability:
This version is available at: 11583/2999037 since: 2025-04-10T13:38:00Z

Publisher:
IEEE

Published
DOI:10.1109/MDAT.2025.3544125

Terms of use:

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

Publisher copyright
IEEE postprint/Author's Accepted Manuscript

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

(Article begins on next page)

06 November 2026

Reliability of Deep Neural Networks: Impact and Open Issues

Annachiara Ruospo[1], Salvatore Pappalardo[2], Vittorio Turco[1], Alberto Bosio[2], Ernesto Sanchez[1]
[1]*Politecnico di Torino, Dip. di Automatica e Informatica, Torino, Italy*
[2]*Univ Lyon, ECL, INSA Lyon, CNRS, UCBL, CPE Lyon, INL, UMR5270, 69130 Ecully, France*
Email: [1]{name.surname}@polito.it, [2]{name.surname}@ec-lyon.fr

***Abstract*—Nowadays, Deep Neural Networks (DNNs) are widely used in safety-critical fields such as automotive and healthcare, where their reliability is crucial due to their direct impact on human lives. Over the years, evaluating their resilience through software-level fault injection experiments has become a common research approach. The corruption of individual bits in the model's parameters has been one of the most studied fault models in the last decade. This work introduces a methodology to evaluate the impact of permanent faults on DNN weights in image classification and object detection tasks, highlighting key ideas, main contributions, and the research's impact over time.**

***Index Terms*—Deep Neural Networks, Reliability, Permanent Faults.**

I. Introduction

Deep Learning (DL) technologies are advancing rapidly, with Deep Neural Networks (DNNs) increasingly integrated into safety-critical systems due to their performance and efficiency. Current devices are manufactured using the most advanced semiconductor technologies (to achieve performance and reduce power consumption). However, new hardware technologies are more critical in terms of reliability [1]. In addition, artificial intelligence (AI) models are never 100% accurate, even in fault-free scenarios: they provide a prediction score that indicates their confidence in performing a specific task, such as classifying an object or segmenting an image. Therefore, the need to ensure the safety and reliability of AI-based systems has emerged as a crucial issue. Today, many efforts are underway in industry and academia to both seek more sophisticated and efficient approaches and to establish safety standards that specifically address the unique challenges posed by DNNs, e.g., the ISO/IEC CD TR 5469 [2].

Recent works have highlighted that permanent faults in DNN hardware accelerators have a major impact on DNN accuracy concerning transient faults such as soft errors [3]. Even single corrupted bits can change the final prediction of DNNs [4]. Therefore, it may be necessary to evaluate the reliability of the AI model before deploying it on the chosen hardware, to understand its vulnerabilities and develop appropriate mitigation solutions. The reliability evaluation is often performed relying on a Fault Injection (FI) method, a popular technique in the dependability domain in which intentional corruptions (e.g., bit-flips) are introduced in the model, and the system's behavior is checked against this deviation. In the last few years, many FI tools and methodologies have been developed at different abstraction levels to carry out this resilience characterization, e.g., [5]–[9]. This work presents a methodology to characterize the resilience of DNN applications against the impact of permanent faults affecting DNNs' parameters (synaptic weights). Under the single-fault assumption, individual bits were corrupted one at a time, and the response of the faulty DNN application was evaluated given a reference set of images. To this end, a fault injection tool was created by exploiting the darknet open-source DNN framework [10]. This research adopts a layer-wise Statistical Fault Injection (SFI) approach [11], [12] to estimate the criticality of different layers in DNNs. The statistical analysis quantifies the percentage of faults that result in incorrect predictions and those that degrade prediction quality without affecting the DNN's correct functionality. The results, provided with a 95% confidence level and a margin of error of 1%, also reveal the most critical bit positions in 32-bit floating-point (FP) data representations.

Reliability investigations were conducted on two well-popular CNN models. The first was a LeNet-like CNN [13], a pivotal architecture used for classifying handwritten digits, trained and validated using the MNIST database. The second model was YOLO [14], a deep convolutional neural network designed for real-time object detection. Overall, this approach aims to study the effect of permanent faults at the software level, so that it is not influenced by the hardware architecture running the CNN (i.e. CPU, GPU, or HW accelerator). To the best of the authors' knowledge, the original idea presented here, was introduced in [6], and it was one of the first research publications targeting permanent faults in the DNN parameters.
In addition, the research offers an overview of the literature, providing an analysis of the most popular FI tools exploited at different levels of abstraction and an overview of reliability assessments on different DL tasks.

Finally, this article explores open issues and emerging trends in the field, highlighting two primary challenges: the *complexity* of current AI models, characterized by numerous parameters, and AI hardware fabricated with advanced technology nodes, and the difficulty of establishing a *fault model* that is both accurate and easy to manage.

The rest of the article is organized as follows. In Section II a description of the approach is given and the most significant experimental results are provided. Next, Section III highlights

Manuscript submitted October 2024.

the impact of the research in the last few years. Finally, Section IV presents open issues in the field and new trends.

II. An Effective Fault Injection Method on DNNs

This research work presents a methodology to evaluate the impact of permanent faults affecting the static parameters (i.e., synaptic weights) of CNNs adopted in automotive fields. The following key contributions are provided:

- A software-layer FI-based analysis of CNN resilience, independent of any potential hardware architecture running the CNN under assessment;
- An analysis of the criticality of different layers in DNNs;
- An analysis of the criticality of different faults depending on the bit position in an IEEE 754 single-precision FP representation.

This Section first introduces the proposed methodology (Section II-A), followed by a detailed description of the experimental setup (Section II-B), and concludes with an analysis of the experimental results (Section II-C).

A. Proposed Methodology

To assess the impact of permanent faults on DNN applications, a FI tool was created by exploiting the darknet open source DNN framework [10] implemented in the C programming language. Darknet is a generic environment that serves as a general-purpose engine for running various types of neural networks. Reliability investigations were performed on two popular CNN models, i.e., LeNet and YOLO [10].

As a permanent fault, we considered the Stuck-at Fault (SaF) model at 0/1 ($SaF0$ and $SaF1$). The Fault Location (FLo) is defined by (1).

$$FLo =< Layer, Connection, Bit, Polarity > \quad (1)$$

Where $Layer$ specifies the CNN layer, $Connection$ is the edge connecting one node of the $Layer$ and Bit is one of the bits of the weight associated with the $Connection$. Finally, $Polarity$ indicates whether the fault is Stuck-at-0 ($SaF0$) or Stuck-at-1 ($SaF1$).

Table I provides the CNN's topology details exploited in the experiments. The first column lists the layer number; the second column specifies the layer type, either Convolution (*Conv*) or Fully Connected (*FC*). The third column reports the number of connections for each layer, while the last column indicates the data representation used for storing each weight. As shown, all CNN weights are represented as 32-bit FP numbers (FP32).

Algorithm 1 Algorithm for Permanent Faults Injection on FP32 synaptic weights of CNNs

```
1: RUN_CNN(CNN, golden_prediction)
2: for i = 0 to FLO.SIZE do
3:     INJECT_FAULT(FLo[i], CNN)
4:     RUN_CNN(CNN, faulty_prediction)
5:     COMPARE(faulty_prediction, golden_prediction)
6:     RELEASE_FAULT(FLo[i], CNN)
7: end for
```

TABLE I: CNN Characteristics.

Layer	Type	Synaptic Connections	Data Representation
LeNet			
0	Conv	2,400	FP32
1	Conv	51,200	FP32
2	FC	3,211,264	FP32
3	FC	10,240	FP32
Tiny YOLO			
0	Conv	432	FP32
1	Conv	4,608	FP32
2	Conv	73,728	FP32
3	Conv	294,912	FP32
4	Conv	1,179,648	FP32
5	Conv	4,718,592	FP32
6	Conv	262,144	FP32
7	Conv	1,179,648	FP32
8	Conv	130,560	FP32
9	Conv	32,768	FP32
10	FC	884,736	FP32
11	FC	65,280	FP32

To ensure independence from any hardware architecture running the CNN, the FI methodology operates entirely at the software layer. The pseudo-code is provided in Algorithm 1, which corresponds to a straightforward serial fault injector. This injector modifies the CNN topology (i.e., it corrupts individual bits) guided by the fault universe defined for the reliability investigation. Specifically, the fault locations `FLo` are determined as outlined in equation 1.

For a given pre-trained CNN, the fault injection process begins by conducting a reference run to establish a baseline set of results, known as the "golden prediction" (line 1 in Algorithm 1). Next, the actual fault injection is initiated.

The first step involves generating a list of fault injection points (also named fault location `FLo`). This list specifies the locations within the CNN where faults will be introduced. For each fault location in the list (line 2 of Algorithm 1), a prediction run is executed with the fault injected, processing the entire test-set (line 4 of Algorithm 1). The resulting output is recorded as a "faulty prediction". Subsequently, the faulty prediction is compared to the golden prediction (line 5 of Algorithm 1). For each inference (i.e., for each image), the golden output (produced without faults) is compared with the faulty output, and any discrepancies are logged for further analysis. Once all image inferences for a specific fault are completed, the fault is removed, and the FI process proceeds to the next fault (line 6 of Algorithm 1).

In detail, the function `compare` of Algorithm 1 classifies the prediction/classification of the faulty CNN with respect to the golden one. The classification is performed as follows:

- **Masked:** No difference is observed between the faulty CNN and the golden one.
- **Observed:** A difference is observed between the faulty CNN and the golden one. Depending on how much the results diverge, we further classify these as:
  - **Safe:** The confidence score of the top-ranked element varies by less than +/-5% with respect to the golden one;
  - **Unsafe:** The confidence score of the top-ranked element varies by more than +/-5% with respect to

the golden one, or the top-ranked element predicted by the faulty CNN is different from that predicted by the golden one. As discussed in [5], this is the most critical observed fault. In [5], a change in the top-1 prediction in CNNs is referred to as Silent Data Corruption SDC-1.

B. Experimental Setup

Table II lists the fault list size for each layer of the two CNN architectures. The number of possible faults is calculated by multiplying the number of synaptic connections by the weight size (32 bits) and then doubling it to account for stuck-at-0 and stuck-at-1 faults. As shown in Column 2, the total number of possible faults is very large, making a full FI campaign impractical. To reduce the execution time, we randomly select a subset of faults. To ensure statistically significant results with a 1% error margin (e=0.01%) and 95% confidence new(t=1.96), approximately 9,000 fault injections are required on average. The exact numbers are provided in the last column of Table II, and they were calculated using the following formula, described in [11].

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

In Eq. (2), N corresponds to the actual size of the population and p to the probability of success, i.e., the probability that the fault will produce a failure. Being a probability, p assumes values between 0 and 1. When the probability of success is unknown, it is common to set p equal to 0.5, meaning that the single event under investigation has the same probability of success (p=50%) or of failure (p=50%).

TABLE II: Statistical Fault Injection and Fault List Details.

Layer	Total Stuck-at Faults (N)	Statistically Injected Faults (n)
LeNet		
0	153,600	9,039
1	3,276,800	9,576
2	205,520,896	9,604
3	655,360	9,465
Tiny YOLO		
0	27,648	7,128
1	294,912	9,301
2	4,718,592	9,584
3	18,874,368	9,599
4	75,497,472	9,603
5	301,989,888	9,604
6	16,777,216	9,599
7	75,497,472	9,603
8	8,355,840	9,593
9	2,097,152	9,560
10	56,623,104	9,602
11	4,177,920	9,582

As above-mentioned, experimental analyses have been performed on two CNN architectures, i.e., a LeNet-like architecture and YOLO, performing image classification and object detection, respectively. We employed the pre-trained LeNet weights from [15]. For the FI campaign, a random selection of 37 validation images from the MNIST database was used. For YOLO, we utilized the pre-trained *yolov3-tiny.weights* available from [10]. The workloads, comprising seven distinct images, were also sourced from [10]. In both case studies, a reduced set of images was used to decrease computational costs and time overhead, enabling a larger number of fault injections to be performed as a trade-off (i.e., by reducing the margin of error). All inferences were run on a server equipped with a dual Intel Xeon CPU E5-2680 v3. A single MNIST inference required approximately 47 milliseconds. For YOLO, each inference took around 200 milliseconds per image, with the exact time varying based on the CPU's workload. For these experiments, we retained the same fault classification criteria used for LeNet, with only one exception. In the case of LeNet, the network classifies the input image as one precise digit, allowing us to consider only the highest top-rank output. However, with YOLO, multiple objects can be detected within a single image, for example, several cars may be present in the image. Therefore, we updated the Unsafe Observed faults definition as follows:

1) The number of detected objects is different from the golden and faulty prediction;
2) The number of detected objects is equal, but the labels associated with them are different (i.e., wrong detection);
3) The "location" of the top ranked element varies by more than +/-5% w.r.t. the golden one.

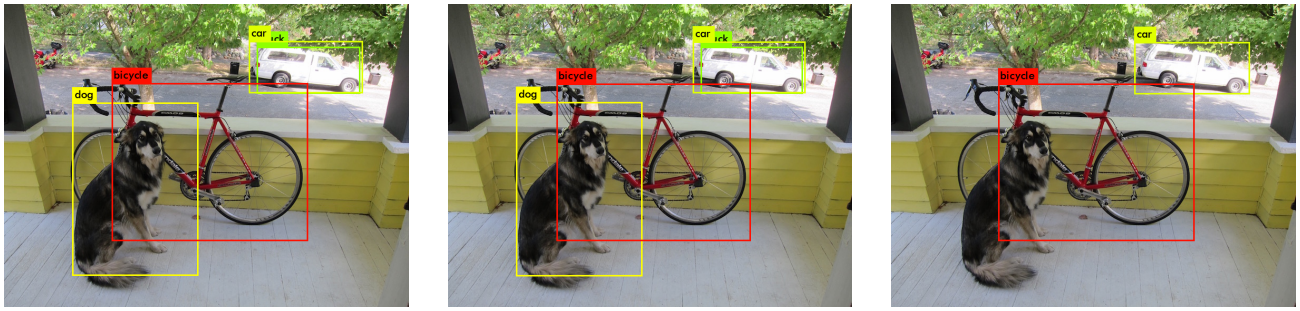
The other definitions (Masked and Safe) do not change. To better clarify this definition, let us resort to an example depicted in Fig. 1 and compare it with the golden prediction shown in Fig. 1a.

In the case of Safe Observed faults, the faulty YOLO successfully detects all four objects, as shown in Fig. 1b, but their "locations" deviate slightly (by less than 5%) from those identified by the golden reference. The term "location" means the rectangle identifying the object in the picture. Conversely, Fig. 1c illustrates an example of an Unsafe Observed fault: it is evident that the CNN only recognizes two objects (the bike and the car) resulting in a wrong prediction.

C. Experimental Results

Experimental results on the CNN performing *image classification* show that convolutional layers are less reliable in the presence of permanent faults compared to fully connected layers. Indeed, the percentage of *Unsafe* faults is about 50% higher. The obtained result is particularly interesting, as it reveals an opposite trend in comparison to the impact of soft errors (e.g., [5]). Our analysis suggests that stuck-at faults affecting the early convolutional layers negatively impact the entire feature extraction process. Fig. 2 reports the percentage of Unsafe Observed Faults across different layers.

FI results on the CNN performing *object detection* (YOLO) do not evidence a significantly higher criticality of convolutional layers compared to fully connected layers: only the first convolutional layer exhibits a greater vulnerability compared to other layers. This behavior is likely tied to the differing importance of the feature extraction process in object detection CNNs. On the other hand, YOLO's resilience presents a dependency on the workload: more contextually-complex



(a) Golden Prediction

(b) Safe Observed faults Prediction - missing recognition

(c) Unsafe Observed faults Prediction - strong corruption of prediction result

Fig. 1: Examples of YOLO predictions.

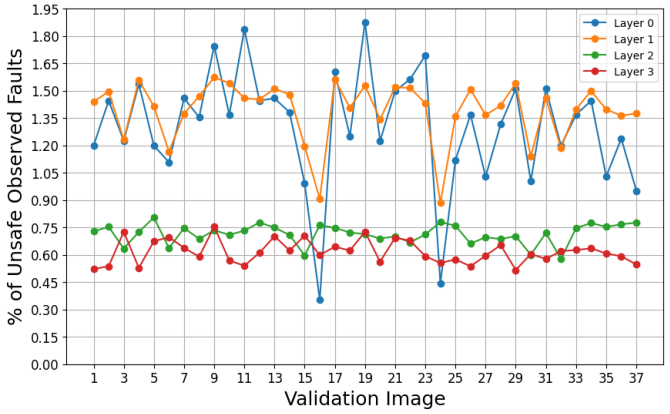


Fig. 2: LeNet: Layer-wise distribution of Unsafe Observed Faults (SDC-1).

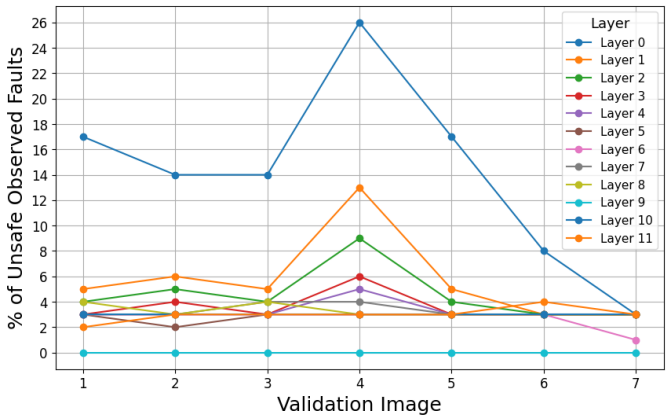


Fig. 3: YOLO: Layer-wise distribution of Unsafe Observed Faults (SDC-1).

images are more prone to mispredictions in the presence of permanent faults.

As shown in Fig. 3, workload number 4 exhibits a peak in Unsafe Observed faults across all layers, especially for the *Layer 0*. Similar to the results observed with LeNet, *Layer 0* is the most sensible layer to the input workload. However, in this case, the variation is significant, with an observed increase of 10% in Unsafe Observed Faults. Therefore, we

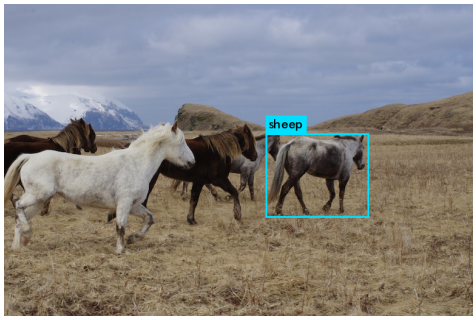


Fig. 4: Example of YOLO misprediction in a highly complex scene. The image contains multiple overlapping objects and ambiguous features, making classification challenging even for human observers.

deeply investigated the input workload number 4, illustrated in Fig. 4. The faulty CNN, instead of detecting the horses, detects one sheep. This input image is inherently "complex" due to the presence of multiple horses to detect. For this specific image, faults in *Layer 0* seem to lead to more Unsafe Observed faults. These findings suggest that, depending on the network topology, the input workload can play a significant role.

D. Most Critical Bits in FP32

Overall, all the *Unsafe* faults (SDC-1) originated from faults affecting the bits of the exponent of the 32-bit FP representation. The sign and the mantissa bits do not have significant impacts (i.e., they led either to *Masked* or *Safe* faults).

III. Impact and Survey of Relevant Works

Ensuring the safety of modern devices is currently one of the most crucial and, at the same time, important requirement. While AI remains one of the most promising technologies of the future, the potential risks are worrying and limit its adoption to strictly controlled environments.

To provide a comprehensive understanding of the impact of this research over the past few years, it would be intriguing to observe the progression in the quantity of published studies in this specific field, starting from the year 2000. Figure 5 shows the number of published works matching the keywords "Reliability of neural network" since 2000. The analysis has

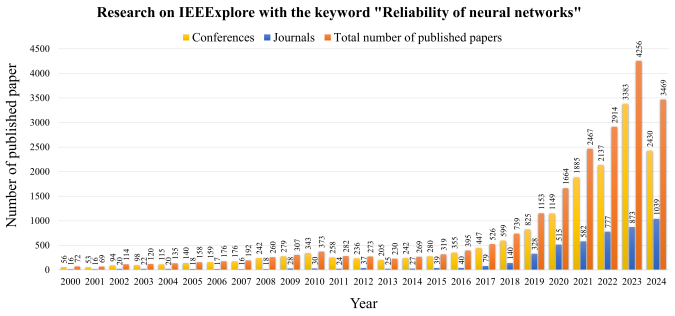


Fig. 5: Number of published works in this topic from 2000 to 2024. Data collected on IEEE *Xplore* on October, 8th, 2024.

been done on the IEEE *Xplore* library, and a distinction has been done between journals and conference papers. The graph demonstrates that over approximately 24 years, there has been a consistent level of interest from the research community in this particular topic, indicating a stable trend over the time. Indeed, DNNs were considered intrinsically resilient due to their redundancy and parallel and distributed architecture. On the contrary, from 2018 to 2024 it is possible to notice a non-negligible increase of interest, thanks also to pioneering studies conducted in 2017 and 2018 showing that even single corrupted bits in memories can change the prediction of a neural network (e.g., [**?**], [16]).

This research work [6] has been published in 2019 and have found a great interest in the research community. As above-mentioned, it is one of the first works targeting the impact of permanent faults affecting the static parameters (weights) of neural networks, independent of any hardware architecture. A novel version of the FI tool has been released as open-source and is now available on [17].

Recent studies have emphasized that permanent faults occurring in DNN accelerators significantly affect the accuracy of DNNs with respect to temporary faults, such as soft errors (e.g., [3]). Permanent faults may be caused by silicon wear out, aging effects, or induced by an external perturbation (i.e., in a harsh environment), and it has been shown that can significantly impact the inference leading to DNNs prediction failures [18]. When DNNs are deployed on hardware devices, being read-only variables, weights and static data are stored in memories and thus subjected to a cumulative bit-flip phenomena induced by external perturbation. It is true that memories can be protected with Error Correction Codes (ECC), but it also true that ECC comes at the cost of extra circuitry with power/performance overheads that may be too costly for low-cost embedded application. Additionally, since the target applications are DNNs, it is claimed that they are intrinsic resilient to faults, and thus it justifies the absence of an error correction mechanism. Moreover, as the characterization based on radiation static/dynamic test shows [19], multiple errors occur, and for that, ECC is simply not effective. This motivates the validity of the adopted fault model: permanent faults targeting synaptic weights in DNNs. Before 2019, few research works targeted the reliability of DNNs in the literature, but *mainly* targeting soft errors (i.e., bit-flip), e.g., [**?**], [20]. In [20], the authors evaluated the reliability of one CNN executed on three different GPU architectures (Kepler, Maxwell, and Pascal). The injection of soft errors has been done by exposing the GPUs running the CNN under controlled neutron beams. Other works address the reliability of DNN accelerators, performing software FIs on data-path elements (i.e., injecting transient faults, [**?**]), or an RTL description in registers storing the DNN's parameters (i.e., [16]). Next, [21] performed a reliability assessment by running FIs directly on an emulation platform, and they targeted both permanent and transient faults. Compared to [21], a very relevant aspect of the research presented in [6] is its ability to conduct reliability assessments independently of any specific hardware architecture.

In summary, the key discoveries of the research presented, which have contributed to its current impact, can be summarized as follows: (i) DNNs, although they try to mimic the human brain, they cannot be considered inherently robust. As depicted in Fig. 5, this finding is reflected in the level of interest observed in published works from 2019 onwards; (ii) as already written in [6], a very big problem remains the impossibility of generalizing: each DNN must be analyzed separately, and, worse, every little change in the model (such as updating DNN's weights) could change the reliability assessment; (iii) the exponent bits in a FP representation are the most critical ones.

A. FIs at different abstraction levels and popular FI tools

In the dependability domain, fault injection is a widely adopted technique for simulating potential failures in both hardware and software components. Tools designed for fault injection play an essential role in evaluating systems' reliability. These evaluations enable the development of effective strategies and techniques aimed at mitigating risks and enhancing tolerance to faults. In the field of deep learning, the adoption of fault injection methods may present some challenges. One of the most pressing is the time required to run the experiments, thus the need to accelerate the injection process while maintaining accuracy of results, while simulating the most realistic conditions possible. Given the diverse scenarios that can be modeled across various components and levels of abstraction, it is fundamental to inject faults that closely mimic real-world conditions. Any inaccuracies in this process could result in misleading conclusions regarding the system's resilience, highlighting the importance of precision in fault injection practices. A more detailed analysis on open issues and challenges is provided in Section IV.

State-of-the-art FI techniques, balancing FI time and accuracy of results, can be performed at very different abstraction levels: considering only the software model, enabling faster FI experiments, while others also include the hardware architecture, resulting in slower execution time but more precise fault injections that are closer to the actual hardware behaviour. Indeed, to address the needs of AI computations, new flavours of specialized hardware architectures have started to emerge. Researchers have found that co-designing the algorithm and the hardware could lead to achieving superior performance.

Both temporal and spatial architectures have been developed to deal with data-intensive workloads introduced by AI applications [22]. Among the temporal architecture, Graphical Processing Units (GPUs) found huge applicability and are considered today as the leading architecture for running AI models. They are a specialized processor originally designed to accelerate graphics rendering but turned out to be very efficient for parallel computing. Among spatial architecture, both specialized Application-Specific-Integrated-Circuits (ASIC) and Field-Programmable-Gate-Arrays (FPGA) accelerators have been proposed. Several research works have shown that, with the advance of technological nodes, new hardware architectures like AI accelerators are susceptible to random-hardware faults [23]. The main causes may be due to external factors such as radiation-induced faults such as Single Event Upsets (SEUs), aging, wearout. Hence, to safety and reliably adopt such architectures in safety-critical systems, it is needed to assess and implement resilience mechanisms.

Given these premises, FI methods and tools operate across various levels of abstraction, ranging from hardware to higher application layers. They typically target three main areas:

- *Simulation-Based FI:* This FI approach is carried out in simulation, by intentionally introducing faults to simulate their effects, as follows:
  - *Software Level*: Faults are injected into high-level DNN models without considering specific hardware, focusing on model vulnerabilities and layer sensitivity.
  - *Hardware Level*: Faults are injected into detailed HDL hardware representations (e.g., RTL, gate-level) to simulate the faults' propagation from the hardware to the DNNs output.
- *Platform-Based FI*: Faults are injected into software models, but the experiments run on physical devices like GPUs and FPGAs, allowing engineers to assess DNN behaviour in real-world conditions.
- *Radiation-Based FI*: External conditions, such as radiations, are used to evaluate DNN resilience by exposing physical platforms to radiation-induced faults, providing highly accurate results. The software is not intentionally corrupted, only the hardware is exposed to radiation-induced faults.

An increasing number of FI tools are being developed and released for DNN reliability analysis, many of which are open-source and offer a wide range of features. These tools are typically classified based on the DL framework they support, with the two most prominent being PyTorch and TensorFlow. One of the first fault injector designed specifically for TensorFlow was TensorFI [8], introduced in 2018. TensorFI enables fault injections at the level of individual operators within the TensorFlow computational graph. Its high-level mechanism allows for straightforward mapping between faults and the TensorFlow graph architecture, making it a flexible tool for dependability analyses. TensorFI supports complex neural network architectures and includes modes for simulating single bit-flip faults. The tool has since evolved, offering enhanced support for new versions of TensorFlow and the Keras APIs. The updated version, named TensorFI+, expands FI capabilities to include both activation matrices and weights, whereas the original TensorFI focused on activation matrices. Additionally, the enhanced version of the tool introduces the ability to simulate both transient and permanent faults, adding more flexibility to fault injections. Other tools have also emerged for TensorFlow. For instance, BinFI [7] implements a binary FI method that enhances the efficiency of analyzing critical bits in DNN systems. BinFI's binary search-like approach contrasts with traditional random fault injections by significantly pruning the search space of the faults' universe, making it optimal for quickly identifying critical bits.

Regarding the PyTorch-based DL frameworks, tools like PyTorchFI [24] have been released to inject faults by perturbing weights or neurons during the DNN execution. This capability adds significant versatility in research, particularly when evaluating the resilience of models in tasks such as object classification and detection. PyTorchFI enables users to simulate faults across different network components, allowing them to observe how these faults affect model outputs and assess the robustness of the network against adversarial attacks or unexpected runtime errors. Building on this, PyTorchALFI [25] extends PyTorchFI by offering additional functionalities for fine-grained studies along with targeted fault scenario configurations, data loader wrapping, test evaluation, visualization, and experiment monitoring.

While its primary focus is not on the FI campaign itself, it excels in providing tools for monitoring and analyzing experiments, offering a comprehensive environment for FI testing and evaluation. In addition, there are fault injectors such as SFIadvancedmodels [17], designed to perform statistical fault injection analyses.

So far, the tool supports only the injection of permanent faults on weights and output neurons, but in the future it will be extended to support advanced and novel fault models, including multiple fault injections. Finally, some tools are capable of supporting both PyTorch and TensorFlow frameworks. An example is CLASSES [26], which can perform fault injection campaigns at the GPU level by leveraging NVBitFI [27]. NVBitFI is a hardware-level fault injection tool for GPU programs that dynamically instruments code without requiring source code access. It effectively analyzes error propagation in GPU systems used in high-performance computing and safety-critical environments. It is also widely used for assessing the resilience of DL workloads. CLASSES combines architectural- and software-level fault injections to extract corruption patterns, thus, injects more realistic fault models. A further FI tool for assessing the resilience of DL accelerators is FIdelity [**?**]. This FI tool, requiring a minimal amount of high-level design information, can inject a major class of hardware errors (transient errors in logic components) in software with high fidelity.

B. FIs targeting different DL tasks

In the previous section, popular FI methods and tools have been discussed considering the abstraction level and the

DL framework adopted (i.e., PyTorch or Tensorflow). It is interesting to add that FI resilience evaluations have been also performed targeting different DNN architectures, performing different tasks. Several studies have analysed the effects of permanent or transient faults on the output of DL models across various tasks. Given the variety of tasks, this subsection will be limited by reporting a few works in the context of computer vision: image segmentation (e.g. [28]), image classification (e.g., [21] [**?**]), object detection (e.g., [18], [29]), and tasks using transformers (e.g., [30]).

Research works [31] and [28] explored FI in semantic segmentation. The former provides a comprehensive analysis, highlighting critical vulnerabilities in layers and parameters when subjected to consistent FI campaigns. The latter proposes a technique for fault identification and masking to mitigate random-hardware faults.

Across other deep learning tasks, the impact of fault injection can vary significantly, depending on the nature of the task and the architecture of the model. For instance, in image classification, fault injection studies like [4] have demonstrated that faults in convolutional layers can drastically reduce accuracy, with certain layers proving more sensitive to perturbations. In object detection tasks, the authors in [18] evaluates the impact of permanent faults on network used in autonomous vehicles. They also perform accelerated neutron beam tests to assess the sensitivity of these networks to transient faults, along with a methodology for determining fault criticality.

Moving on to advanced perception tasks, such as those used by autonomous robots in resource-limited environments, the authors in [30] focus their experiments on assessing the resilience of transformer architectures against radiation-induced faults. These studies aim to assess how radiation-induced faults affect model performance, offering a comprehensive view of fault propagation and its impact on model accuracy.

Works in the literature across different DNN architectures outline the importance of evaluating individual DNN topologies as specific architectural elements, such as layer types, data types, and network depth, significantly influence the resilience of DNNs.

IV. Open Issues and New Trends

Evaluating the reliability of AI systems has become essential today; however, the task is proving to be increasingly challenging for two main reasons: the *complexity* of current AI models and AI hardware in terms of parameters and advanced technology nodes, and the challenge of defining a *fault model* that is both accurate and easy to manage.

A. Complexity of current AI systems

To address the complexity issue, which leads to huge and impractical fault list sizes (in the case of FI-based approaches), one of the current strategies involves selecting a reduced set of faults that is statistically significant. In other words, that is able to represent the characteristic of the entire population of faults with a given margin of error and a confidence level. The majority of the experimental results obtained in recent years are based on statistical fault injection methods [5], [11], [12], [18], [21]. Even though it is possible to dramatically reduce the total number of fault injections by exploiting fault sampling, as the complexity of models increases, more sophisticated techniques should be devised. Indeed, given a fixed (i) error margin, (ii) confidence level, (iii) probability of success, as the population size grows, the sample size saturates at a fixed value. It also means that the number of FI experiments that need to be performed for a population equal to 50 million elements corresponds to the same for a population of 100 million. This does not invalidate the methodology; rather, it emphasizes the importance of adhering to the statistical assumptions underlying it.

B. Fault Model Selection

The second challenge, when dealing with FI-based reliability assessment methodology, is the definition of a proper fault model. As above-mentioned, many research works targeted random-hardware faults affecting synaptic weights in neural networks. However, it is worth saying that, being a software-level fault model, its accuracy is not very high [32]. The real impact of software-level faults can be different based on the adopted machine learning tool (e.g., PyTorch or TensorFlow), due to existing numerical inaccuracies in operations used for deep learning workloads [4]. Moreover, memories can be protected via error-correcting coding techniques, which makes the injection of single or multiple faults in memory only relevant when assuming that these protections are absent. Furthermore, it is worth saying that many works in the literature adopt the single-fault assumption, either permanent or transient faults affecting the synaptic weights of neural networks. As the technology nodes are shrinking, multiple faults are arising, and the ECC is simply not effective with the detection and correction of multiple faults. To deal with multiple faults, some works in the literature propose software-level FI-based reliability assessments adopting increasing Bit Error Rate (BER) or fault patterns altering a given percentage of bits within a given percentage of weights in neural networks [33]. The problem with the latter solution is that there are practically infinite multiple fault patterns, and the selection of a subset of them is not statistically defined (in other words, it can lead to too pessimistic or optimistic results, with no correlations with the real failure rate of the population of faults).

To overcome the limitations of introducing software-level faults, recent works in the literature have introduced corruption patterns, as a combination of architectural- and software-level fault injections (e.g., [26]). However, because of their dataflow dependency, corruption patterns present additional challenges. It is impossible to generalize these patterns since they depend significantly on the workload scheduling and the particular hardware architecture being targeted.

Overall, given these premises, there is a need to develop more accurate fault models to be injected at the software level, reflecting the complexity of the hardware architecture and the impact of dataflows on fault propagation.

V. Conclusion

This work presents a characterization framework for analyzing the impact of permanent faults affecting CNNs intended to be used, for instance, in automotive applications. The characterization was done by means of a fault injection campaign on the *darknet* open source DNN framework. The experiments were performed at the software level with the aim of being independent of the HW architecture and to derive a common characterization of the behavior of these networks in the presence of random-hardware faults. We believe that this is an important outcome since the designer, starting from these results, could be able to select the most convenient HW architecture for a given CNN. The paper then outlines the impact of this research and related work in the literature. To conclude, it presents open questions, in the authors' experience, and open points for future research directions.

Acknowledgments

This study was carried out within the FAIR - Future Artificial Intelligence Research and received funding from the European Union Next-GenerationEU (PIANO NAZIONALE DI RIPRESA E RESILIENZA (PNRR) – MISSIONE 4 COMPONENTE 2, INVESTIMENTO 1.3 – D.D. 1555 11/10/2022, PE00000013) and from the RE-TRUSTING project, ANR-21-CE24-0015. This manuscript reflects only the authors' views and opinions, neither the European Union nor the European Commission can be considered responsible for them.

References

[1] C. Liu, Y. Sun, P. Ren, D. Gao, W. Luo, Z. Chen, and Y. Xia, "New Challenges of Design for Reliability in Advanced Technology Node (Invited)," in *2020 4th IEEE Electron Devices Technology Manufacturing Conference (EDTM)*, pp. 1–4, 2020.

[2] ISO/IEC JTC 1/SC 42, "ISO/IEC TR 5469:2024: Artificial intelligence, Functional safety and AI systems." https://www.iso.org/standard/81283.html, 2024. Accessed: December 9th.

[3] J. J. Zhang, T. Gu, K. Basu, and S. Garg, "Analyzing and mitigating the impact of permanent faults on a systolic array based neural network accelerator," in *2018 IEEE 36th VLSI Test Symposium (VTS)*, 2018.

[4] E. Ozen and A. Orailoglu, "Low-Cost Error Detection in Deep Neural Network Accelerators with Linear Algorithmic Checksums," *Journal of Electronic Testing*, vol. 36, no. 6, p. 703–718, 2020.

[5] G. Li *et al.*, "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*.

[6] A. Bosio, P. Bernardi, A. Ruospo, and E. Sanchez, "A Reliability Analysis of a Deep Neural Network," in *2019 IEEE Latin American Test Symposium (LATS)*.

[7] Z. Chen, G. Li, K. Pattabiraman, and N. DeBardeleben, "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*, pp. 1–23, 2019.

[8] Z. Chen, N. Narayanan, B. Fang, G. Li, K. Pattabiraman, and N. DeBardeleben, "TensorFI: A flexible fault injection framework for tensorflow applications," in *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*, pp. 426–435, IEEE, 2020.

[9] Y. He, P. Balaprakash, and Y. Li, "FIdelity: Efficient Resilience Analysis Framework for Deep Learning Accelerators," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 270–281, 2020.

[10] J. Redmon, "Darknet: Open Source Neural Networks in C," 2013–2016.

[11] 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*, pp. 502–506, April 2009.

[12] A. Ruospo, G. Gavarini, C. de Sio, J. Guerrero, L. Sterpone, M. S. Reorda, E. Sanchez, R. Mariani, J. Aribido, and J. Athavale, "Assessing Convolutional Neural Networks Reliability through Statistical Fault Injections," in *2023 Design, Automation Test in Europe Confe rence Exhibition (DATE)*, pp. 1–6, 2023.

[13] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[14] P. Adarsh, P. Rathi, and M. Kumar, "YOLO v3-tiny: Object detection and recognition using one stage improved model," in *2020 6th international conference on advanced computing and communication systems (ICACCS)*, pp. 687–694, IEEE, 2020.

[15] Ashitani, "Darknet Mnist." https://github.com/ashitani/darknet_mnist, 2018. Accessed: 2024-12-10.

[16] B. Salami, O. S. Unsal, and A. C. Kestelman, "On the resilience of RTL NN accelerators: Fault characterization and mitigation," in *2018 30th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, pp. 322–329, IEEE, 2018.

[17] CAD-Polito-IT, "SFIadvancedmodels." https://github.com/cad-polito-it/SFIadvancedmodels. Accessed: 2024-10-10.

[18] A. Lotfi *et al.*, "Resiliency of automotive object detection networks on GPU architectures," in *2019 IEEE International Test Conference (ITC)*, pp. 1–9, IEEE, 2019.

[19] L. Matanaluza, A. Ruospo, D. Soderstrom, C. Cazzaniga, M. Kastriotou, E. Sanchez, A. Bosio, and L. Dilillo, "Emulating the Effects of Radiation-Induced Soft-Errors for the Reliability Assessment of Neural Networks," *IEEE Transactions on Emerging Topics in Computing*, pp. 1–1, 2021.

[20] F. Fernandes dos Santos *et al.*, "Evaluation and Mitigation of Soft-Errors in Neural Network-Based Object Detection in Three GPU Architectures," in *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, 2017.

[21] B. Reagen *et al.*, "Ares: A framework for quantifying the resilience of deep neural networks," in *Proceedings of the 55th Annual Design Automation Conference*.

[22] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient Processing of Deep Neural Networks: A Tutorial and Survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.

[23] J. J. Zhang, K. Basu, and S. Garg, "Fault-Tolerant Systolic Array Based Accelerators for Deep Neural Network Execution," *IEEE Design Test*, vol. 36, no. 5, pp. 44–53, 2019.

[24] A. Mahmoud, N. Aggarwal, A. Nobbe, J. R. S. Vicarte, S. V. Adve, C. W. Fletcher, I. Frosio, and S. K. S. Hari, "PyTorchFI: A runtime perturbation tool for dnns," in *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, pp. 25–31, 2020.

[25] R. Gräfe, Q. S. Sha, F. Geissler, and M. Paulitsch, "Large-scale application of fault injection into PyTorch models-an extension to pytorchFI for validation efficiency," in *2023 53rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks-Supplemental Volume (DSN-S)*, pp. 56–62, IEEE, 2023.

[26] C. Bolchini, L. Cassano, A. Miele, and A. Toschi, "Fast and accurate error simulation for CNNs against soft errors," *IEEE Transactions on Computers*, vol. 72, no. 4, pp. 984–997, 2022.

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

[28] S. Burel, A. Evans, and L. Anghel, "Techniques for detecting and masking faults in semantic segmentation applications," *Microelectronics Reliability*, vol. 157, p. 115397, 2024.

[29] F. F. dos Santos, P. Navaux, L. Carro, and P. Rech, "Impact of Reduced Precision in the Reliability of Deep Neural Networks for Object Detection," in *2019 IEEE European Test Symposium (ETS)*, pp. 1–6, 2019.

[30] P. R. Bodmann, P. Rech, and M. Saveriano, "Evaluating the Reliability of Vision Transformers for Space Robotics Applications," in *2024 International Conference on Space Robotics (iSpaRo)*, pp. 278–283, 2024.

[31] G. Govarini, A. Ruospo, and E. Sanchez, "A Fast Reliability Analysis of Image Segmentation Neural Networks Exploiting Statistical Fault Injections," in *2023 IEEE 24th Latin American Test Symposium (LATS)*, pp. 1–6, 2023.

[32] A. Ruospo, E. Sanchez, L. M. Luza, L. Dilillo, M. Traiola, and A. Bosio, "A Survey on Deep Learning Resilience Assessment Methodologies," *Computer*, vol. 56, no. 2, pp. 57–66, 2023.

[33] D. A. Moussa, M. Hefenbrock, and M. Tahoori, "Testing for Multiple Faults in Deep Neural Networks," *IEEE Design Test*, vol. 41, no. 3, pp. 47–53, 2024.

Annachiara Ruospo is an Assistant Professor at Politecnico di Torino, Italy. She received an MsC (2018) and a PhD degree cum laude (2022) in Computer Science Engineering from the same institution. Her research interests encompass AI safety and security, neural networks reliability, and testing and verification of modern dependable systems.

Salvatore Pappalardo received his Master degree in Computer Engineering from Politecnico di Torino (Italy) in 2022. He is currently a Ph.D. student at Ecole Centrale de Lyon, Institute of Nanotechnology. His main research interests are trustworthy hardware for AI, approximate computing and AI system's online testing.

Vittorio Turco received his Master's degree in Mechatronic engineering from Politecnico di Torino in 2023. Currently, he is a Ph.D. student and researcher at the DAUIN Department at Politecnico di Torino. His research interests include safety and reliability of AI system.

Alberto Bosio is a Full Professor at Ecole Centrale de Lyon, Institue of Nanotechnology (France). His research activities are related to the design and test of advanced digital circuits and systems. He is a member of the IEEE and the Vice-Chair of the European Test Technical Technology Council.

Ernesto Sanchez is an Associate professor at Politecnico di Torino, Italy. His research interests include digital circuits and systems reliability, evolutionary computation and ANN reliability. He received his degree in Electronic Engineering from Universidad Javeriana, Colombia in 2000, and his Ph.D. degree in computing engineer from Politecnico di Torino in 2006. He is an IEEE senior member.