A Systematic Literature Review on Hardware Reliability Assessment Methods for Deep Neural Networks

MOHAMMAD HASAN AHMADILIVANI, MAHDI TAHERI, and JAAN RAIK, Tallinn University of Technology, Estonia
MASOUD DANESHTALAB, Mälardalen University, Sweden and Tallinn University of Technology, Estonia
MAKSIM JENIHHIN, Tallinn University of Technology, Estonia

Artificial Intelligence (AI) and, in particular, Machine Learning (ML), have emerged to be utilized in various applications due to their capability to learn how to solve complex problems. Over the past decade, rapid advances in ML have presented Deep Neural Networks (DNNs) consisting of a large number of neurons and layers. DNN Hardware Accelerators (DHAs) are leveraged to deploy DNNs in the target applications. Safety-critical applications, where hardware faults/errors would result in catastrophic consequences, also benefit from DHAs. Therefore, the reliability of DNNs is an essential subject of research.

In recent years, several studies have been published accordingly to assess the reliability of DNNs. In this regard, various reliability assessment methods have been proposed on a variety of platforms and applications. Hence, there is a need to summarize the state-of-the-art to identify the gaps in the study of the reliability of DNNs. In this work, we conduct a Systematic Literature Review (SLR) on the reliability assessment methods of DNNs to collect relevant research works as much as possible, present a categorization of them, and address the open challenges.

Through this SLR, three kinds of methods for reliability assessment of DNNs are identified, including Fault Injection (FI), Analytical, and Hybrid methods. Since the majority of works assess the DNN reliability by FI, we characterize different approaches and platforms of the FI method comprehensively. Moreover, Analytical and Hybrid methods are propounded. Thus, different reliability assessment methods for DNNs have been elaborated on their conducted DNN platforms and reliability evaluation metrics. Finally, we highlight the advantages and disadvantages of the identified methods and address the open challenges in the research area. We have concluded that Analytical and Hybrid methods are light-weight yet sufficiently accurate and have the potential to be extended in future research and to be utilized in establishing novel DNN reliability assessment frameworks.

CCS Concepts: • **General and reference** → **Surveys and overviews**; • **Hardware** → **Hardware reliability**; • **Computer systems organization** → **Neural networks**; **Reliability;**

This work was supported in part by the European Union through the European Social Fund in the frames of the "Information and Communication Technologies (ICT) programme" ("ITA-IoIT" topic), the Estonian Research Council grant PRG1467 "CRASHLESS," the Estonian-French science and technology cooperation programme PARROT project "EnTrustED," and by the Swedish Innovation Agency VINNOVA project SafeDeep.
Authors' addresses: M. H. Ahmadilivani, M. Taheri, J. Raik, and M. Jenihhin, Tallinn University of Technology, Tallinn, Estonia; e-mails: {mohammad.ahmadilivani, mahdi.taheri, jaan.raik, maksim.jenihhin}@taltech.ee; M. Daneshtalab, Mälardalen University, Västerås, Sweden and Tallinn University of Technology, Tallinn, Estonia; e-mail: masoud.daneshtalab@taltech.ee.

0360-0300/2024/01-ART141
https://doi.org/10.1145/3638242

Additional Key Words and Phrases: Reliability assessment, deep neural networks, DNN hardware accelerator, fault injection

ACM Reference format:
Mohammad Hasan Ahmadilivani, Mahdi Taheri, Jaan Raik, Masoud Daneshtalab, and Maksim Jenihhin. 2024. A Systematic Literature Review on Hardware Reliability Assessment Methods for Deep Neural Networks. *ACM Comput. Surv.* 56, 6, Article 141 (January 2024), 39 pages.
https://doi.org/10.1145/3638242

1 INTRODUCTION

Deep Neural Networks (DNNs) are nowadays extensively applied to a wide variety of applications due to their impressive ability to approximate complex functions (e.g., classification and regression tasks) via learning. Since powerful processing systems have evolved in the recent decade, DNNs have emerged to be deeper and more efficient as well as employed in an ever broader extent of domains. Meanwhile, using **DNN Hardware Accelerators (DHAs)** in safety-critical applications, including autonomous driving, raises reliability concerns [1, 2]. In compliance with ISO 26262 functional safety standard for road vehicles, the evaluated **FIT (Failures In Time)** rates of hardware components must be less than 10 (meaning 10 failures in 1 billion hours) to pass the highest reliability level [3], which requires diligent design.

DNNs are deployed in their target application by different DHA platforms, including **Field-Programmable Gate Arrays (FPGAs)**, **Application-Specific Integrated Circuits (ASICs)**, and **Graphics Processing Units (GPUs)** [4]. Depending on the DHA and the application's environment, different fault types may present a threat to the reliability of the component [5]. Figure 1 illustrates the reliability threats (described in Section 2.3) in an example DHA. In this figure, different fault types originating from several reasons could occur in any of the DHA's components that may lead to a disastrous misclassification, e.g., once a red light is detected as a green light. Faults are originated from hardware, however, they can also be modeled at software platforms for the ease of study. Accordingly, the reliability of DNNs is tightly coupled with the reliability of DHAs as faults are coming from hardware. It is worth highlighting that the reliability in this article does not relate to the reliability in software engineering or security issues, e.g., adversarial attacks.

It has been shown in several studies that the functionality of DNNs in terms of accuracy is remarkably degraded in the presence of faults [6–10]. Recently, numerous research works have been published on the assessment and enhancement of DNNs' reliability. However, due to the extent of the DNNs domain, these works approach the problem of the reliability of DNNs from various perspectives. We are faced with several applications of DNNs as well as a variety of DNN algorithms for different tasks. Therefore, it will lead to distinct platforms and reliability threats, which hinders unifying and generalizing the methods of reliability assessment and enhancement of DNNs.

Throughout the literature, various methods of DNN reliability assessment and enhancement are presented. Some review papers have been published on the topic of DNNs reliability enhancement methods [4, 5, 11–14]. These works aim to formulate the reliability problem in DNNs, categorize available reliability improvement methods in this domain, and overview the fault injection methods for reliability assessment. The analysis in Reference [14] is the first review on the subject of fault tolerance in DNNs and describes different fault models and reliability improvement methods in DNNs. However, the topic was still not as mature as it is today, and numerous works have been published afterwards. Subsequent works such as References [4, 5, 11] provide extensive reviews on the reliability improvement methods for DNNs and characterize taxonomies of different methods. Nevertheless, they do not consider the assessment and evaluation methods of the reliability for

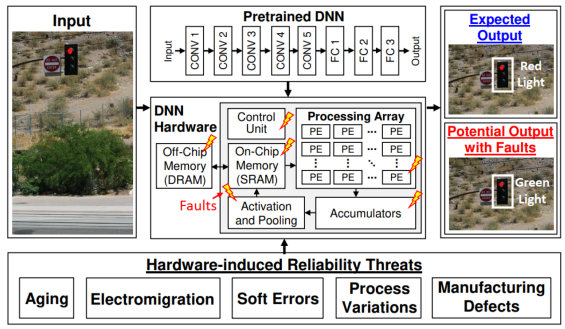

Fig. 1. Hardware-induced reliability threats in an example DHA and their possible impact on the output [1].

DNNs. Other surveys [12, 13] have reviewed fault injection methods for DNNs reliability assessment, with the former work focused merely on fault criticality assessment and the latter including only a few papers in the survey. In this article, we present the first **Systematic Literature Review (SLR)** dedicated to all methods of reliability assessment of DNNs.

Reliability assessment of DNNs is a process for evaluating the reliability of a DNN that is being executed either as a software model or by a hardware platform. However, the assessment method for reliability may vary, depending on the platform. In this regard, it is necessary to comprehend and distinguish the different methods used to assess the reliability of DNNs across platforms. This article establishes a thorough picture of the reliability assessment methods for DNNs and systematically reviews the relevant literature. To achieve this, we carry out the SLR methodology [15, 16] to present this survey. The primary focus of this review is to investigate the methods of reliability assessment for DNNs, generalize and characterize the methods, and identify the open challenges in the domain.

To the best of our knowledge, this survey represents the first comprehensive literature review on reliability assessment methods for DNNs. We cover all published papers from 2017 to 2022 that could be found through a systematic search. The main contributions of this article are:

— Reviewing the literature of the reliability assessment methods of DNNs, systematically;
— Analyzing the trends of published papers over different years and methods;
— Characterizing and categorizing the reliability assessment methods for DNNs;
— Identifying fault injection methods based on the DNN platforms;
— Introducing analytical and hybrid reliability assessment methods along with fault injection;
— Addressing the open challenges in the research area and recommendations for future research directions.

The structure of the article is as follows: Section 2 presents the background on DNNs and reliability concepts; Section 3 explains the methodology of this survey and addresses the research questions; Section 4 reviews the study briefly, presents the statistics of the publications, and depicts the top-level taxonomy of reliability assessment methods for DNNs. In Section 5, the details of the reliability assessment methods are explained. Section 6 includes pros and cons of methods and open challenges of the study domain. Section 7 provides the conclusions of this survey.

2 PRELIMINARIES

2.1 Deep Neural Networks

Deep Learning (DL) is a sub-domain of **Machine Learning (ML)**, which is the study of making computers learn to solve problems without being directly programmed [17]. Regarding the

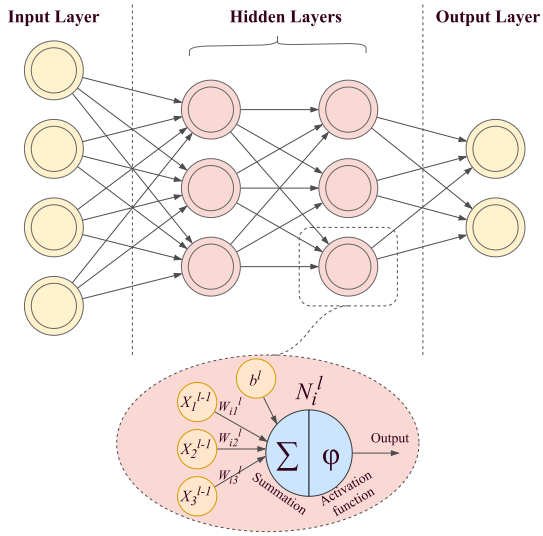

Fig. 2. Abstract view of a simple neural network with the detail of a neuron.

impressive ability of DNNs in learning, they are applicable in a vast variety of domains, such as image and video processing, data mining, robotics, autonomous cars, gaming, and so on.

DNNs are inspired by the human brain, and they have two major phases: training and inference. In the training phase, which is an iterative process and performed once, the hyper-parameters (e.g., weights and biases) of the neural network are updated on a determined dataset. A loss function is adopted in the training phase that measures the difference between the expected and the estimated output of DNN to achieve higher accuracy. Accuracy expresses the proportion of the DNN outputs coinciding with the expected output. However, in the inference phase, representing the DNN deployment, the network is run several times with the parameters obtained during the training phase [17].

DNNs are constructed of the units of neurons. Each neuron receives some activation inputs and multiplies them by the corresponding weights. Then, it conveys the summation of the weighted activations to its output. A set of neurons builds up a layer that may have other additional functions, e.g., activation function (ReLu, sigmoid, etc.), batch normalization, (max or average) pooling, and so on [17]. Equation (1) represents the function of the ith neuron in layer l (denoted as N_i^l) with input activations from the previous layer *l-1* with n outputs (denoted as X^{l-1}), where W and b represent weights and bias, respectively.

$$N_i^l = \phi\left(\sum_{j=0}^{n} X_j^{l-1} \times W_{ij}^l + b^l\right) \tag{1}$$

An abstract view of a neuron and a neural network is depicted in Figure 2. As shown, inputs are fed into the network through the input layer. The middle layers, called hidden layers, determine the depth of the network and conduct the function of the DNN. The output layer is where the network decides. It produces some probabilities of the possible outputs, i.e., output confidence score, and the class with the highest value is the top-ranked output.

DNNs have various architectures each suitable for specific applications. Nevertheless, it is worth mentioning some terms that are used in this article. **Convolutional Neural Networks (CNNs)** are extensively used in classification, object detection, and semantic segmentation tasks and consist of multiple **convolutional (CONV)** and **fully connected (FC)** layers. CONV layers have

a set of **two-dimensional (2D)** weights, called filters, that extract a specific feature from the input of the layer. A channel is a set of **input feature maps (ifmap)** that is convolved with filters resulting in the **output feature maps (ofmap)** [17].

In the research area of CNNs, there are some models of networks that are most frequently used. For instance, LeNet-5 [18], AlexNet [19], GoogLeNet [20], VGG [21], and ResNet [22] are introduced for image classification, and YOLO [23] is designed for object detection. In addition, prominent datasets that are mostly used for training networks on image classification tasks are MNIST [24], CIFAR [25], and ImageNet [26]; and on object detection are KITTI [27] and PASCAL VOC [28].

In addition, due to the large number of parameters and calculations in DNNs, **Quantized Neural Networks (QNNs)** [29] and **Binarized Neural Networks (BNNs)** [30] are introduced to reduce the complexity, memory usage, and energy consumption of DNNs. These DNNs are the quantized versions of existing DNNs that reduce the bit-width of their parameters and calculations with an acceptable accuracy loss.

2.2 DNN Platforms

2.2.1 Software Frameworks. DNN software frameworks and libraries in high-level programming languages have been developed to ease the process of designing, training, and testing DNNs. These frameworks are widely used due to their high abstraction level of modeling and short design time. Some of well-known software frameworks that are being used for training the DNNs are: TensorFlow [31], Keras [32], PyTorch [33], DarkNet [34], and Tiny-DNN [35]. All these frameworks are capable of using both CPU and GPU to accelerate the training process.

2.2.2 DNN Hardware Accelerators (DHAs). DHAs are used for the training as well as the inference phase of DNNs. They are called accelerators due to their dedicated design employing parallelism for reducing the execution time of the DNN, either in training or inference. DHAs can be generally categorized into four classes: FPGAs, ASICs, GPUs, and multi-core processors [36, 37].

According to the literature review of DHAs in Reference [37], FPGAs are used more frequently than other DHA platforms in terms of implementing DNNs, due to their availability and design flexibility for different applications [38]. FPGAs are programmed via their configuration bits that determine the functionality of the FPGA. The system of FPGA-based DNN accelerators usually consists of a host CPU and an FPGA part with corresponding interconnections between them. In this design model, the DNN is implemented on the FPGA part and the CPU controls the accelerator with software, while each part is integrated with memories [38]. A typical structure of FPGA-based DNN accelerator is depicted in Figure 3, which is based on HW/SW co-design, which means separating the implementation of DNNs on the integrated CPU (the software) and FPGA (the hardware) that are communicating with one another [39]. **High-Level Synthesis (HLS)** tools, which can synthesize high-level programming languages to RTL, are also used for developing FPGA-based DNN accelerators [38].

ASIC-based DNN accelerators are more efficient than FPGAs in terms of performance and power consumption but less flexible in terms of applications and require a long design time [40]. There are two general types of architectures for ASIC-based DHA platforms: spatial and temporal [17]. Figure 4 depicts an example of a spatial architecture model that is constructed of 2D arrays of **Processing Elements (PEs)** flowing data horizontally and vertically from input/weight buffers to output buffers. PEs perform **Multiply-Accumulate (MAC)** operations on inputs and weights representing a neuron operation in the DNN. Off-chip memories are required to store the parameters of DNNs and save the intermediate results from PEs. **Tensor Processing Unit (TPU)**, produced by Google, one of the most applicable ASIC-based DNN accelerators, is based on this type of architecture [41].

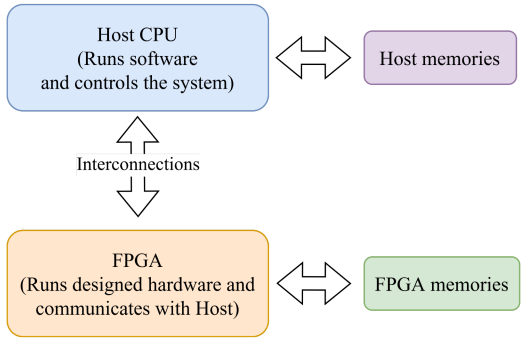

Fig. 3. Typical structure of an FPGA-based DNN accelerator [38].

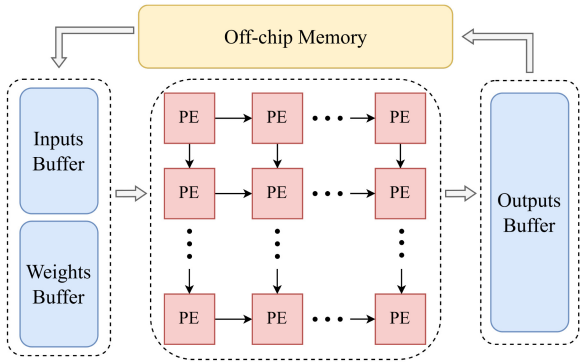

Fig. 4. An example of spatial architecture for ASIC-based DNN accelerators [42].

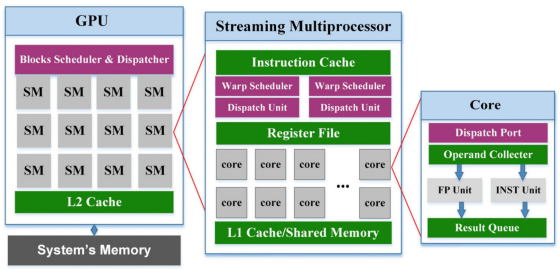

Fig. 5. General architecture of CUDA-based GPUs [44].

GPUs are a powerful platform for training and inferring deep networks and are vastly used in safety-critical applications [43]. GPUs include up to thousands of parallel cores, which make them efficient for DNN algorithms, especially in the training phase [40]. GPUs are designed to run several threads of a program and are also exploited to accelerate running DNNs [37]. The general architecture of GPUs is depicted in Figure 5. There are numerous **Streaming Multiprocessors (SMs)** in the GPU, each having several cores with a shared register file and caches, while a scheduler and dispatchers control the tasks among and within SMs and cores [44].

Multi-core processors, e.g., ARM processors, deploy DNNs mostly for edge processing and **Internet of Things (IoT)** applications [45–47]. They facilitate DNNs with parallel computing and low power consumption and provide a wider range of applications for DNNs.

2.3 Reliability, Threats, Fault Models, and Evaluation

Terms of robustness, reliability, and resilience are mostly used in the research pertaining to the reliability of DNNs. These terms are often used interchangeably and ambiguously. In the following, we present the definitions of these three terms as applied in the current literature review:

— **Reliability** concerns DNN accelerators' ability to perform correctly in the presence of faults, which may occur during the deployment caused by physical effects either from the environment (e.g., soft errors, electromagnetic effects) or from within the device (e.g., manufacturing defects, aging effects, process variations).
— **Robustness** refers to the property of DNNs expressing that the network is able to continue functioning with high integrity despite the alteration of inputs or parameters due to noise or malicious intent.
— **Resilience** is the feature of DNN to tolerate faults in terms of output accuracy.

In this work, we are concerned about the reliability of DNNs, which refers to the ability of accelerators to continue functioning correctly in a specified period of time with the presence of faults. Reliability in this article does not relate to the reliability and test in software engineering or security issues, e.g., adversarial attacks in which an attacker perturbs the inputs or parameters.

Faults are the sources of threatening the reliability of DNN accelerators (see Figure 1) that can be caused by several reasons, e.g., soft errors, aging, process variation, and so on [1]. Soft errors are transient faults induced by radiation that are caused by striking charged particles to transistors [48]. Aging is the time-dependent effect of the increasing threshold voltage of transistors due to physical phenomena that will lead to timing errors and permanent faults [49]. Process variations are alterations of transistor's attributes in the process of chip fabrication. As a consequence, voltage scaling may result in faults at the outputs of transistors during their operation [50].

Faults as reliability threats are generally modeled as *permanent* and *transient* faults [5, 11, 14]. Permanent faults result from process variations, manufacturing defects, aging, and so on, and they stay constant and stable during the runtime. However, transient faults are caused by soft errors, electromagnetic effects, voltage and temperature variations, and so on, and they show up for a short period of time. Nevertheless, once a faulty value from a component is read by another component and the propagated value does not coincide with the expected one, an *error* happens. Therefore, a fault is an erroneous state of hardware or software, and an error is a manifestation of it at the output. *Failure* or system malfunction is the corruption or abnormal operation of the system, which is caused by errors [14, 51, 52].

Faults may have different impacts on the output of DNNs and can be classified based on their effects. A fault may be masked or corrected if detected or result in different outputs compared to the fault-free execution (golden model), in which case, the fault is propagated and observed at the output. Faults observed at the output of the system can be classified in two categories: **Silent Data Corruption (SDC)** and **Detected Unrecoverable Errors (DUE)**, depending on whether a fault is undetected (SDC) or detected (DUE) [11, 53]. Figure 6 illustrates this general fault classification scheme regarding the output of systems adopted from Reference [51].

Reliability assessment is the process in which the target system or platform is modeled or presented, and by means of simulations, experiments, or analysis, the reliability is measured and evaluated. Reliability assessment is a challenging process, and several methods can be adopted for modeling and evaluating reliability. In general, evaluating the reliability of a system can be performed by three approaches: **Fault Injection (FI)** methods, analytical methods, and hybrid methods [54]. FI methods are exploited to inject a model of faults into the system implemented either in software or hardware, while the system is in simulation or being executed. Analytical

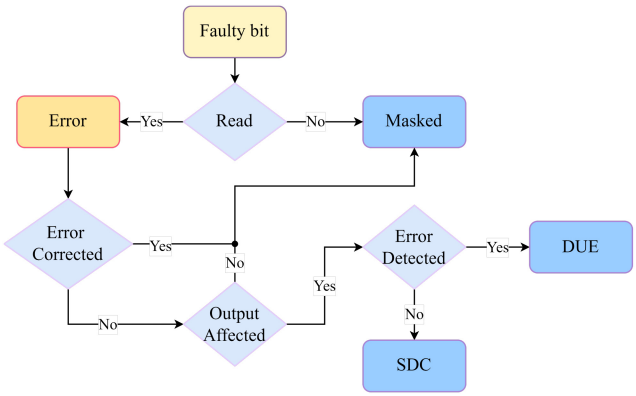

Fig. 6. The adopted fault classification based on the output point of view, as in Reference [51].

methods attempt to model the function of the system and its reliability with mathematical equations, depending on the target architecture. In hybrid methods, an analytical model is adopted alongside an FI to evaluate the reliability. Generally, FI methods are more realistic than analytical and hybrid methods; however, FI is a time-consuming process with a high computational complexity [55].

In the reliability assessment using FI, it is necessary to determine the target platform, potential fault locations (logic or memory), and the fault type (transient or permanent). Transient faults in logic show up in one clock cycle, while in the memory, they flip a bit that will remain until the end of the execution. Permanent faults are modeled as stuck-at-0 (sa-0), or stuck-at-1 (sa-1), and they exist during the whole execution. According to the selected fault model, perturbation of the model is performed, the system is run, and the outputs are gathered. The output of faulty execution should be compared with the one of the golden-model to measure the impact of faults on the system.

FI allows calculating reliability metrics, e.g., **Failures-In-Time (FIT), Architectural Vulnerability Factor (AVF)**, SDC rate, **Soft Error Rate (SER)**, cross-section, and so on. FIT is the number of failures in 10^9 hours, AVF is the probability of fault propagation from a component to other components in a design, SDC rate refers to the ratio of the outputs affected by faults, SER refers to the ratio of soft error occurrence, and cross-section is the proportion of observed errors over all collided particles. These quantitative evaluation metrics are usually tightly coupled to each other, yet follow a different purpose to express the reliability of a system.

Exhaustive fault injection into all bits of a platform at every clock cycle requires an extensive simulation. Therefore, to determine how many faults could be injected into the system to be representative statistically, a confidence level with an error margin is presented [56]. It provides a fault rate or **Bit Error Rate (BER)** for an FI experiment. The number of FI experiments' repetitions regarding the number of possible bit and clock cycle combinations to support the number of injected faults determines the execution space for the FI task.

3 REVIEW METHODOLOGY

Systematic Literature Review (SLR) is a standard methodology for reviewing the literature in a recursive process and minimizing bias in the study [15, 16, 37]. Hence, the SLR methodology is adopted in this survey. The methodology determines:

— Specifying the **Research Questions (RQs)**,
— Specifying the search method for finding and filtering the related papers,

— Extracting corresponding data from the found papers based on the RQs,
— Synthesizing and analyzing the extracted data.

Therefore, based on the aforementioned steps of SLR, the RQs that we attempt to answer are:

— **RQ1:** What is the distribution of the research works in the domain of reliability assessment? (To obtain the trend of publications in this domain).
— **RQ2:** What are the existing methods of reliability assessment for DNNs? (To comprehend the entire variety of methods in this domain).
— **RQ3:** How could the existing methods be characterized and categorized in terms of reliability assessment methods? (To categorize existing works and provide the taxonomy, a systematic instruction for finding the suitable method for potential applications in this domain).
— **RQ4:** What are the open challenges in the domain of reliability assessment methods for DNNs? (To specify the remaining areas for future research).

The motivation for this survey is the numerous recent papers published on the reliability of DNNs emphasizing the need for such a literature review. We have searched for the papers systematically through scientific search servers. The main databases and publishers we have used are: Google Scholar, IEEE Explore, ACM Digital Library, Science Direct, and Elsevier. The initial set of papers is provided by searching some keywords in the mentioned servers, including "reliability of DNNs", "hardware reliability of DNN accelerators", "resilient DNNs", "robust DNNs", "the vulnerability of DNNs", "soft errors in DNNs", "fault injection in DNNs" ("DNN" also replaced with "CNN").

Subsequently, based on the title and abstract of each paper, we select them. This selection is based on the criterion of whether the paper may be concerned with the reliability of DNNs or not. In addition, the references and citations of the papers have been checked for the chosen papers to find more related papers. In this process, we selected 242 papers based on their titles and abstracts.

In the next step, we study the introduction, conclusion, and methodology sections of each paper to decide whether we include the paper in the review or not. The inclusion criteria of the papers are:

— The paper is published by one of the scientific publishers and has passed through a peer-review process,
— The focus of the work is DNN, neither generic reliability assessment methods using DNNs as one of the examples nor employing DNNs for assessing the reliability of a platform.
— The work includes a reliability assessment method for DNNs,
— The method of reliability assessment is clear and well-defined,
— Terms including reliability, robustness, resilience, or vulnerability **must** refer clearly to reliability issues, as defined in Section 2.3.

Papers that have included similar keywords but have not matched the above conditions are excluded. As a result, we have included 139 papers published from 2017 to the end of 2022 in this literature review to build up the taxonomy of the literature review and methods' categorization.

In the following, we have designed a **Data Extraction Form (DEF)** based on the RQs. In this form, we have taken note of reviewing the papers to find some specific data such as:

— General method of reliability modeling (FI, analytical, or hybrid),
— The platform where DNNs are implemented,
— The fault model and fault locations in case of FI,
— Details of reliability assessment method,
— Reliability evaluation metrics.

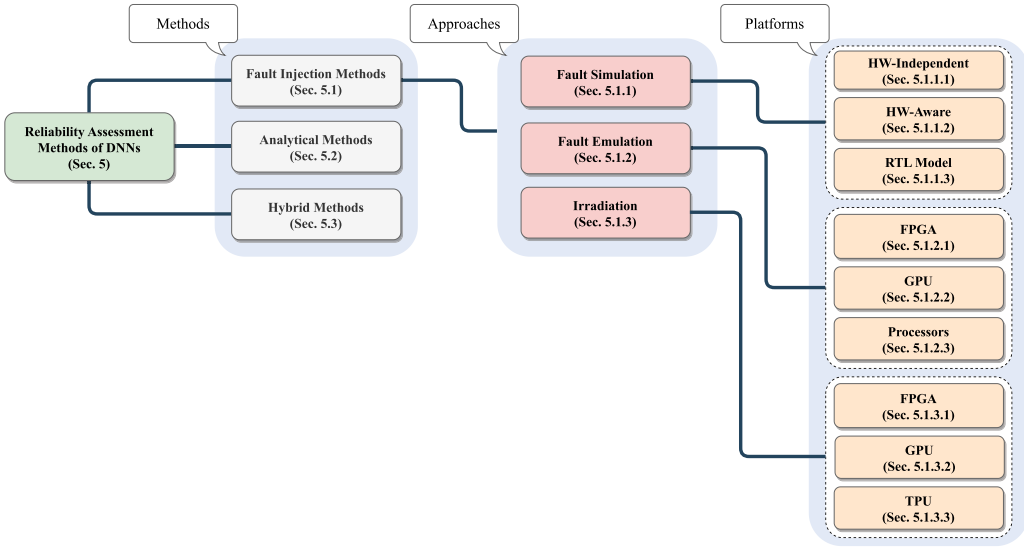

Fig. 7. Top-level overview of the reliability assessment methods in this work.

In the final step, after reviewing all the selected papers and filling in the DEF, we synthesized and analyzed the obtained data from the papers. Thereafter, we have provided the categorization taxonomy of the reliability assessment methods for DNNs, have characterized them in this article, and analyzed them to find the open challenges.

4 STUDY OVERVIEW

This section presents an overview of the study and the analyzed statistics of the included works in different categories. As mentioned, we have included 139 papers from 2017 to 2022 for categorizing the reliability assessment methods for DNNs.

4.1 Taxonomy

Figure 7 represents the top-level categorization overview of the study to address RQ2 and RQ3. Reliability assessment of DNNs is categorized into three main methods: Fault Injection, Analytical, and Hybrid.

4.1.1 ***Fault Injection (FI)*** *Methods.* The works based on this method evaluate the reliability of DNNs by fault injection campaign. There exist several taxonomies for the fault injection approaches in the hardware reliability domain [12, 54, 55, 57, 58]. Therefore, we adapt them for categorizing the related works on DNNs into three approaches addressed in Figure 7 and Table 1. FI methods are categorized into three approaches of fault injection as follows:

— **Fault Simulation:** DNNs are implemented either in software by high-level programming languages or **Hardware Description Languages (HDL)** and faults are injected into the model of the DNN. In the former case, some works consider a DHA model in their software implementations while others do not. We divide works on this approach into hardware-independent, hardware-aware, and RTL model platforms. RTL models represent ASIC-based DHAs.
— **Emulation in Hardware:** Research works on this approach implement and run DNNs on a DHA (i.e., FPGA, GPU, or processor) and inject the faults into the components of the accelerator by a software function, FI framework, and so on.

Table 1. Fault Injection Categorization with the Corresponding References

FI Method	Fault Simulation			Fault Emulation			Irradiation		
DNN Platform	**HW-independent**	**HW-aware**	**RTL Model**	**FPGA**	**GPU**	**Processors**	**FPGA**	**GPU**	**TPU**
Fault Type	Transient [59–61] [62–67] [68–73] [74–78] [79–82]	Transient [9, 83] [84–87] [88–91]	Transient [36, 92–94]	Transient [8] [95–99] [100–102] [103–105] [106–108]	Transient [10] [44, 109–111] [112–115] [116? –119] [120–122]	Transient [36] [92, 123, 124] [125–127] [128–130]	Transient [95, 96, 103, 106] [108, 131, 132] [133–135]	Transient [10, 115, 117] [121, 122] [136, 137]	Transient [138, 139]
	Permanent [72] [140–143]	Permanent [144] [6, 145–147] [148–150]	Permanent [7, 58, 151] [152–154]	Permanent [105, 106] [155, 156]	Permanent [137, 157, 158] [159, 160]				
Fault Location	Weights [61, 62] [63–65, 67, 69] [68, 70, 71, 73, 74] [75–79] [80–82, 140] [141–143, 161?]	Weights [9, 85, 86] [88–90] [91, 146, 148]	PEs, MACs [7, 151, 152] [153, 154]	Configuration Bits [8, 95] [96–99] [100–102] [103, 104, 107] [108, 162, 163]	Registers, Instructions [10, 44, 109, 110] [111–114] [115, 116, 118] [119–122] [157–160]	Register File [36, 92, 123, 124] [125–127] [128–130]	Entire FPGA Package [95] [96, 103, 132] [108, 131, 133] [135]	Entire GPU [10, 115, 117] [121, 122] [136? , 137]	Entire Chip refs [138, 139]
	Activations [59] [60, 64, 66, 72] [76, 78]	Activations [6] [9, 83? , 84] [87–90] [91, 144, 145] [147–149]	Registers, Buffers, LUTs [36, 92–94]	On-Chip Memories [8] [98, 100, 101] [102, 105, 106] [155, 162, 163]	Weights, Activations [117, 137]	Instructions [130]	HyperRAM [106, 134]		
Evaluation	Accuracy Loss [59] [60–63, 68] [69, 71–74] [75–79] [80–82, 141, 143]	Accuracy Loss [6, 9, 84, 87, 88] [89–91, 144] [145–147] [148–150]	Accuracy Loss [7, 93, 151] [152–154]	Accuracy Loss [8, 97, 99? , 100] [101, 102, 105] [106, 107, 156] [108, 155, 162, 163]	Accuracy Loss [113, 120, 158] [159, 160]	Fault Classification [36, 92, 123, 124] [125–127] [128–130]	Fault Classification [103, 133, 135]	Fault Classification [10, 115, 121] [122, 137]	Fault Classification [138, 139]
	Fault Classification [64, 65, 67] [140, 142, 143]	Fault Classification [87]	Fault Classification [36, 58, 92] [93, 94]	Fault Classification [8, 97–99] [101–103] [104, 107, 163]	Fault Classification [10, 44, 109–111] [114–116, 118] [119, 121, 122, 137] [157–160]	Reliability Equations [36, 124, 126] [129, 130]	Reliability Equations [95, 96, 103] [106, 108] [131, 133, 134]	FIT Rate [10, 115] [121, 122]	FIT Rate [138, 139]
	SDC Rate [66, 70]	SDC Rate [83, 85]		Reliability Equations [95, 96, 103] [104, 105]	Vulnerability Factors [10, 44] [109, 110, 112, 113] [114, 116, 118] [119, 121, 122]	Vulnerability Factors [129, 130]			

— **Irradiation:** DNN is implemented on a DHA (i.e., FPGA, GPU, or TPU) placed under an irradiating facility to inject beams onto it.

Most of the works on DNNs' reliability assessment use FI methods. Therefore, we characterize three approaches of FI methods in Table 1. In each approach of FI methods, the works are distinguished based on DNN platforms. Furthermore, in each category, we elaborate on how the works determine the fault types and locations and evaluate the reliability by metrics. The details will be discussed in Section 5.1.

4.1.2 Analytical Methods. Works relying on an analytical method for estimating DNNs' reliability attempt to determine how parameters and neurons of a DNN affect the output based on the connections of neurons and layers. Therefore, they analyze the structure of DNNs and provide a model for the impact of faults on the outputs to find more critical and sensitive components in the DNN. Hence, they can evaluate the reliability of DNNs by means of vulnerability analysis derived by analyses and eliminate the complexity of simulating/emulating the faults in reliability assessment.

4.1.3 Hybrid Methods. Both fault injection and analytical methods are used in this category of works to take advantage of both. In this regard, analytical methods can provide some mathematical models in addition to a straightforward fault injection into the system for reliability evaluation, so metrics of reliability evaluation can be obtained with less complexity than extensive FI experiments and more realistic than analytical methods.

4.2 Research Trends

To address RQ1, we present the main statistics on the papers included in this study. Figure 8 shows the distribution of the 139 included papers published over the years 2017–2022. Regarding the chart of Figure 8, it can be seen that research on the topic of DNNs' reliability started in 2017 and in the following years it drew increasingly more attention and turned into an active topic of study.

Figure 9 illustrates the number of papers based on different reliability assessment methods among all identified works in this literature review. It can be observed that the majority of works use fault injection to assess the reliability of DNNs while only 10% of the works consider analytical (11 works) and hybrid analytical/FI (3 works) methods. In this regard, we present Figure 10 to illustrate the distribution of works using FI over different approaches and DNN platforms. It shows that most of the works belong to the hardware-independent platform of simulation in the software approach. Moreover, in the emulation in hardware approach, most of the works are done on the GPU platform. Hence, the figures present the trend of the research domain and the distribution of works over different methods and approaches, leading to areas where there is still room for future research.

5 CHARACTERIZATION

In this section, details of reliability assessment methods for DNNs are presented based on the categorizations in Figure 7 and Table 1. We start from FI methods, which include the majority of works. Then, analytical and hybrid methods will be discussed.

5.1 Fault Injection Methods

In FI methods of reliability assessment, once the DNN platform and fault model are determined, perturbation and system execution are performed, and the reliability is evaluated. Regarding the categorization in Table 1, the identified approaches of FI methods on DNN reliability assessment are presented in this subsection, separately. Since FI is the most frequently used method in the

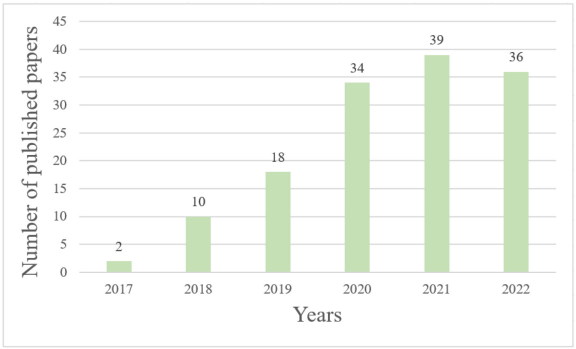

Fig. 8. Number of included papers over years.

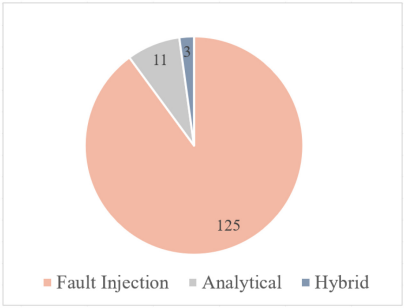

Fig. 9. Proportion of each method in the reliability assessment of DNNs among included works.

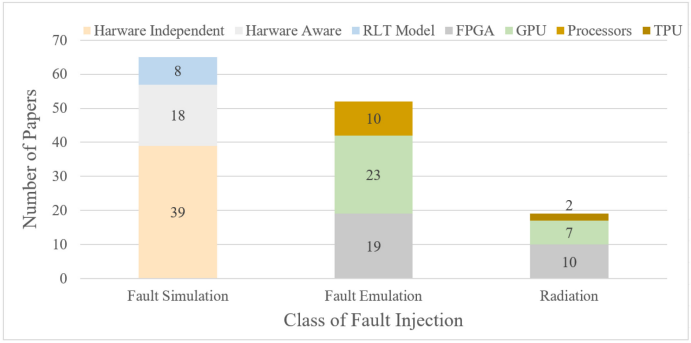

Fig. 10. Distribution of included papers over different FI approaches and platforms.

reliability assessment of DNNs, there are various presented evaluation metrics. To elaborate and distinguish different evaluation metrics, we have presented them for different approaches and platforms, separately.

5.1.1 Fault Simulation. In this subsection, the works assessing the reliability of DNNs by FI with a fault simulation approach are described. There are three platforms in this approach, i.e., hardware-independent, hardware-aware, and RTL models that are explained in the corresponding subsections.

5.1.1.1 Hardware-independent Platform. In this platform, DNNs are implemented in software DNN frameworks. Therefore, fault injection is performed on top of the frameworks, i.e., PyTorch (used in References [59, 68, 70–72, 76, 78]), Keras (used in References [61, 62, 80, 81]), TensorFlow (used in References [66, 79]), Caffe (used in Reference [77]), DarkNet (used in References [73, 140, 142]). Implementing the DNN in software provides a flexible environment for studying the effect of various fault models. As shown in the corresponding branch of Table 1, both transient and permanent faults are studied in this platform. However, most of the works studied transient faults (soft errors, SEU, MBU, etc.).

To model faults at the software level, the fault model is determined differently regarding the fault type and general aspect of DHAs. In this regard, modeling and injecting permanent faults are straightforward. They are active throughout the entire execution and set the value of a bit or variable (in weights or activations) to 0 or 1, as experimented in References [72, 140, 142]. To model transient faults, the following assumptions are considered for injecting faults into parameters, i.e.:

— DNN's parameters (e.g., weights) are stored in the memory of the accelerator. Hence, random transient faults are injected into random bits of weights as a bitflip in different executions, as experimented in References [61–65, 67–71, 73–82, 141, 143, 161].
— Faults in inputs/outputs of DNN's layers (i.e., activations) lead to the study of their impacts on both memory and logic. Activation memory faults are studied in References [72, 76], and faults in logic or datapath are investigated in References [59, 60, 64, 66, 78].

Therefore, to experiment the impact of faults on memory elements of DHAs at the software level, faults are injected into random weights and activations, and to model fault effects on logic, faults are injected into random activations. Most of the relevant works on Hardware-independent platform inject transient faults into the bits of randomly selected weights. Nearly all works in this class inject faults based on BER, determining the ratio of faulty bits throughout the values. In addition, to reach the 95% confidence level with 1% error margin, they repeat the tests several times with different random faults as in References [77, 80, 140, 142].

***Evaluation*:** For evaluating the reliability, different metrics are considered. References [59–63, 68, 69, 71–82, 141, 143] report accuracy loss under fault campaign experiments. They compare the accuracy of the faulty network with the accuracy of the fault-free network on the same test set. Some works classify the injected faults regarding the outputs of the faulty network compared with the golden model output. References [140, 142, 143] inject one permanent fault per experiment and classify them into three classes:

— **Masked:** No difference between the outputs of the faulty network and the golden model.
— **Observed-safe:** Different output of the faulty network with the golden model, while the confidence score of the top-ranked element is reduced by less than 5% with respect to the one of the golden one.
— **Observed-unsafe:** Different output of faulty network with the golden model, while the confidence score of the top-ranked element is reduced by more than 5% with respect to the one of the golden one.

Moreover, in References [65, 67] transient faults are injected into the encrypted weights of a network, and they are classified based on the effect of faults on execution of the program and results, as:

— **Silent or safe:** Similar to "masked," mentioned above in References [140, 142].
— **SDC:** Only affects the output results of the network.

— **Detected as a software exception:** Affects the execution of the program and stops it.
— **Detected by padding check action:** Corrupts the ciphertext.

Burel et al. [64] have adopted the fault classification scheme for semantic segmentation applications in which DNNs label each pixel of an input image according to a set of known classes. The corresponding classes are:

— **Masked:** Similar to "masked," mentioned above.
— **No Impact SDC:** No labels of pixels are modified.
— **Tolerable SDC:** Labels of less than 1% of pixels are modified, and no class is removed/added due to the fault.
— **Critical SDC:** Labels of more than 1% of pixels are modified or any class is removed/added due to the fault.

A specific way of fault evaluation based on fault classification is only considering the faults that affect the output as SDC, since they are critical. References [66, 70] evaluate the network based on the proportion of faults that affect the output classification results as SDC rate. Therefore, the reliability of a network can be evaluated by fault classification based on their effect on the outputs, whether by changing the output results, or by a threshold of accuracy loss, or system exceptions. This way of evaluation assists in understanding how faults would be propagated and affect the network.

***Software FI Tools*:** Some fault injectors are presented as tools that are able to support the reliability study of DNNs with different fault models in software frameworks of DNNs. PyTorchFI [164], TensorFI [165–167] and its extension TensorFI+ [168, 169], and Ares [170] inject faults into DNNs, which are implemented in PyTorch, Tensorflow, and Keras, respectively. All of these open-source frameworks can inject both permanent and transient faults into weights as well as activations with specified error rates, hence, the accuracy loss can be evaluated. TensorFI also benefits from providing the SDC rate. These frameworks are used in the reliability studies of DNNs, e.g., PyTorchFI in References [60, 70], TensorFI in Reference [66], and Ares in Reference [80].

Moreover, to enhance the efficiency of the aforementioned tools, additional fault injectors have been introduced. One such injector, known as BinFI [171], is an extension of TensorFI that aims to identify critical bits in DNNs. Another fault injector, namely, LLTFI [172], is proposed to inject transient faults into specific instructions of DNN models in either PyTorch or TensorFlow and has been found to be faster than TensorFI. Additionally, a checkpoint-based fault injector is proposed in Reference [173] that enables studying the impact of SDCs independently of the DNN implementation framework.

*5.1.1.2 **Hardware-aware Platform**.* This platform includes works that consider an abstract model of the accelerator in their implementation of DNNs in software. They implement the network in DNN software frameworks as well as high-level programming languages. Therefore, they take advantages of simulation in software fault injection while they also apply the reliability assessment to the abstract model of the accelerator.

References [83, 87] implement a DNN in Tiny-DNN and map it to the RTL implementation of the accelerator. They study the effect of transient faults in memory and datapath accurately. In these studies, FI is performed in software while all of its parameters are integrated with the corresponding hardware components. Authors in Reference [88] implement the DNN and the fault injector in software inspired by an FPGA-based DNN accelerator. Moreover, in References [9, 91], DNN and FI are implemented in Keras, and the architecture of a systolic array accelerator is considered for a fault-tolerant design. Similarly, authors in Reference [85] and [86] evaluate their proposed reliability improvement technique on memories in TensorFlow while injecting transient faults into

the weights. PyTorch is used in References [89, 90] to implement the DNN, and transient faults are injected into activations (datapath or MAC units) and weights (memory) regarding the systolic array accelerator model. Reference [84] also uses PyTorch and injects faults by a custom framework called TorchFI to inject faults into the outputs of CONV and FC layers of the network.

The effect of permanent faults at PEs' outputs is studied in References [6, 144] where the model of the accelerator is adopted from implementing the DNN in an N2D2 framework [174]. Furthermore, authors in References [145, 149] use PyTorch and study permanent faults in MAC units of an accelerator while training to improve the reliability at inference. Authors in Reference [148] have developed a Keras-based accelerator simulator to study the effect of permanent faults on the on-chip memory of accelerators by injecting permanent faults into fmaps and weights. Weight remapping strategy in memory to decrease the effect of permanent faults is evaluated in Reference [146] using Ares. SCALE-Sim [175], a systolic CNN accelerator simulator, is adopted in Reference [150] to study permanent faults in PEs and computing arrays in systolic array-based accelerators.

Similar to the Hardware-independent platform, faults are injected based on BER, or fault rate, and experiments are repeated to reach 95% confidence level and 1% error margin [9, 87, 91].

***Evaluation*:** Nearly all works in this class evaluate the DNN by accuracy loss after fault injection [6, 9, 84, 86, 88–91, 144, 146–150]. References [83] and [85] evaluate the reliability by SDC rate as the proportion of faults that caused misclassification in comparison with the golden model. In addition, authors in Reference [87] differentiate SDCs of injected transient faults into defined classes and calculate FIT for the accelerator (*accel*) by its components (*comp*) with Equation (2) in which FIT_{raw} is provided by the manufacturer, $Size_{comp}$ is the total number of the component bits, and SDC_{comp} is obtained by FI.

$$FIT_{accel} = \sum_{comp} FIT_{raw} \times Size_{comp} \times SDC_{comp} \tag{2}$$

In addition, in this work, SDCs are classified by comparing the faulty and golden model outputs as:

— **SDC-1:** Fault caused a misclassification in the top-ranked output class.
— **SDC-5:** Fault caused the top-ranked element not to exist in the top-5 predicted output classes.
— **SDC-10%:** Fault caused a variation in the output confidence score of the top-ranked output class more than 10% compared to the golden model.
— **SDC-20%:** Fault caused a variation in the output confidence score of the top-ranked output class more than 20% compared to the golden model.

5.1.1.3 RTL Model Platform. Research works that leverage the RTL model of ASIC-based DHAs and simulate fault injections are described in the following. We identify three groups of FI experiments in this platform, divided based on the architecture of DHAs:

— 2D systolic array accelerators [7, 93, 151–154],
— RTL implementation of DNNs [94]
— **Multi-Processor System-on-Chips (MPSoCs)** for DNNs, [58].

In the first group, a configuration of TPU is utilized in References [7, 93, 153, 154], and a model of a 2D systolic array is implemented in References [151, 152]. Reference [7] also uses Eyeriss [176] architecture for the accelerator. In this group, FI is performed at RTL, and all works inject random permanent faults into PEs/MACs of the arrays, except Reference [93], which injects random transient faults into buffers, control and data registers.

The second group, which includes Reference [94], implements DNNs in RTL to enable a fault simulation study in approximated DNNs. In this work, SEU injected into Look-Up Tables are simulated and studied.

In the third group, which exploits MPSoCs, faults are emulated in the components of the target multicore processor. Authors in Reference [58] propose a three-level pipeline FI framework that simulates permanent faults in the hardware model of an MPSoC and evaluate the reliability at the software level. In their framework, the RTL model of the platform is provided as well as the fault injector unit at the lowest level. The software implementation of the DNN exists in the middle level of the framework that performs a pipelined inference and runs each layer of the network on a separate core. In the top-level of the framework, synchronization of layers and reliability evaluation is fulfilled.

***Evaluation*:** Most works in this class evaluate the reliability by accuracy loss. Nonetheless, fault classification is performed in References [58, 93, 94]. Authors in Reference [58] adopted the classification of Reference [87], which was discussed in Hardware-aware platform (Section 5.1.1) previously. Furthermore, they added two more classes for the faults that cause Hang (the HDL simulation never finishes) and Crash (the HDL simulation immediately stops). Authors in Reference [94] classify the faults similar to the general fault classification scheme (Masked, SDC, crash) with different terminology.

In addition, Reference [93] classifies SDCs on how they impact classification outputs compared with the golden model:

— **Tolerable Misclassification:** The input is misclassified the same as the golden model with different output confidence scores,
— **No Impact Misclassification:** The input is misclassified in both golden and faulty models but into different classes,
— **Critical Misclassification:** The input is correctly classified in the golden model but misclassified in the faulty model,
— **Tolerable Correct Classification:** The input is correctly classified in both golden and faulty models with different output confidence scores,
— **Beneficial Correct Classification:** The input is misclassified in the golden model but correctly classified in the faulty model.

5.1.2 Fault Emulation. In this subsection, research works that assess the reliability of DNNs by emulating FI in hardware accelerators are explored. FPGA and GPU platforms are described, respectively.

5.1.2.1 FPGA Platform. DNNs are implemented fully or partially (e.g., one layer) on FPGAs to perform the inference phase as described in Section 2.2, and faults are being emulated on different locations of the accelerator. In most of the works on the FPGA platform, the fault injector unit is implemented in software that is run on a processor, and faults are injected into the FPGA running the DNN under analysis. This HW/SW co-design process benefits from the high-performance execution of DNNs and fast fault injection. It is worth mentioning that some works implement only a part of the DNN (e.g., one specific layer) on the FPGA [97, 98, 108].

In this group of works, Zynq-based architecture **System-on-Chips (SoCs)** [177], which take advantage of an ARM processor co-existing with the FPGA, are deployed. We categorize this group of studies into three classes:

— A host computer (e.g., a PC) initializes the faults [97–99, 107, 108],
— The on-board embedded processor initializes the faults [8, 95, 100–106, 162, 163],
— Fault injection module resides inside the hardware design implementation [96, 155, 156].

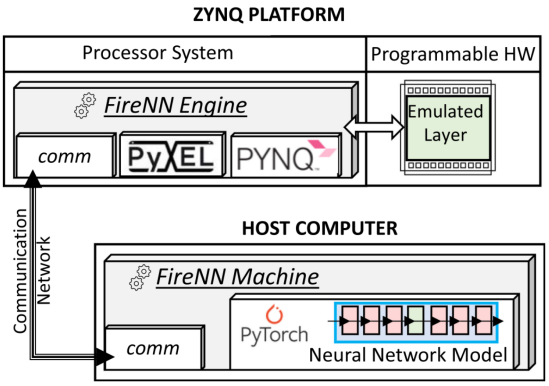

Fig. 11. An overview of the architecture of the FireNN platform [97, 98].

In the first class, faults are generated by a host computer of the accelerator design. Then, the faults, network parameters, and FPGA configuration bits will be sent to the board. The FPGA starts running, and the on-board processor collects the results. The on-board processor plays the role of a controller between FPGA and the host computer. In the end, the results would be passed back to the host computer for further processing and reliability evaluation. All works of this class emulate transient faults (SEU) in configuration bits of the FPGA and exploit the accuracy loss of the DNN for reliability evaluation. Nevertheless, authors in Reference [107] explore transient faults in **Flip Flops (FFs)** exhaustively beside random transient faults in configuration memory and classify them as tolerable, critical, and crashes.

FireNN is proposed in References [97, 98] as a platform for deploying DNNs on Zynq-based architecture SoCs along with a host computer in a way that DNN is run partially on the FPGA to perform a reliability evaluation. As shown in Figure 11, *FireNN machine* runs the neural network and communicates with the *FireNN engine* for reliability evaluation of the layer under analysis running on the FPGA. Faults are generated by the host computer and are injected to the FPGA through the engine. This platform injects SEUs in weights, layer inputs, and configuration bits.

In the second class, faults are generated and injected into the FPGA's configuration bits or on-chip memories by the embedded processor. The embedded processor or a host computer is responsible for the reliability evaluation. The proposed method in References [162, 163] provides an injection of permanent faults into the configuration bits of the FPGA as well as into the on-chip memory blocks through the interfaces between the embedded processor and FPGA on Zynq SoC. References [95, 103, 104] provide a similar design to inject transient faults into configuration bits of the FPGA. The effects of transient faults into both on-chip memories and configuration bits of an FPGA running pruned DNNs are studied in Reference [100]. Authors in Reference [95] provide random-accumulated FI and exhaustive FI approach on the configuration bits to emulate neutron and ionizing radiation. Moreover, permanent and transient faults in on-chip memory (HyperRAM) are studied in References [105, 106] with a software emulator and are validated by radiation results.

It is worth mentioning that injecting faults into the configuration memory is a repetitive process, where in each experiment of FI, the faulty configuration bits are loaded to the configuration memory. Then, the system is run and the results are collected. Thereafter, the next fault(s) are injected into the fault-free configuration bits loaded to the corresponding memory to analyze the newly injected fault(s).

A framework named Fiji-FIN is proposed in Reference [102], and the underlying method is also used in References [8, 101]. This framework is capable of injecting transient faults into both

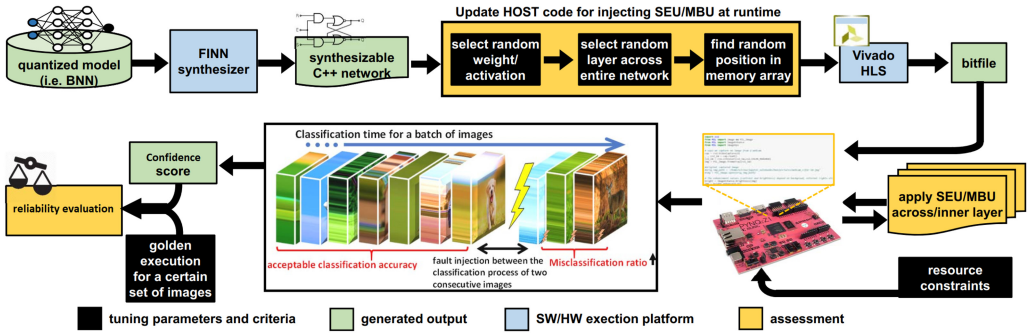

Fig. 12. Fiji-FIN framework for fault injection into FPGAs [102].

configuration bits of FPGA and on-chip memories. In this method, FINN framework [178] is used to develop and train the BNN, and the proposed framework manipulates the FINN's output to prepare it for the fault injection. The bit stream file of the FPGA is obtained by an HLS tool and imported to the FPGA. While the system is running, the faults are generated and injected by the embedded processor and the reliability is evaluated in comparison with the golden model. Figure 12 depicts in detail the steps of this FI framework.

In the third class, References [155] and [156] inject permanent faults, and the work in Reference [96] injects transient faults into the hardware implementation of the network. Authors in Reference [155] use the FINN framework to implement the QNN with 2-bit weights and activations, and a block has been added into the hardware design that is deployed for injecting stuck-at faults into the output of PEs. Reference [156] injects permanent faults into the registers of the RTL model of the network. Authors in Reference [96] explore the effect of transient faults to the configuration bits of FPGAs in which different accelerator architectures (Softcore FGPU and ZynqNet HLS) are implemented.

***Evaluation*:** For evaluating the reliability of DNNs on the FPGA platform, accuracy loss is exploited in References [8, 100–102, 106, 108, 155, 156, 162, 163]. Moreover, fault classification is also performed in References [8, 97–99, 101, 103, 104, 163]. References [103, 104] classify SEUs in configuration bits of the FPGA as critical if a fault caused misclassification with respect to the golden model; otherwise, the fault is tolerable. In addition, Benign Errors are considered in Reference [104], which are the faults that caused true classification of the inputs that were misclassified in the golden model. Another fault classification is presented in References [97, 98] that does not only consider critical and tolerable faults but also categorizes the faults that prevent the accelerator from generating the classification output. In this regard, the effect of faults on the system performance degradation is the criterion for classifying faults in Reference [99].

Reliability is evaluated by different metrics considering accuracy loss regarding the application of the target networks in References [162, 163]. These works consider top-5 and top-1 accuracy loss for image and audio classification tasks, respectively. For object detection, **mean Average Precision (mAP)**, and for image generation, **Structural Similarity Index (SSIM)** is adopted. Regarding the adopted metrics for accuracy loss in each network, the faults are classified into three classes with different ranges of accuracy loss ($\leq$1%, 1%~5%, $\geq$5%) caused by FI. In addition, they categorize the faults that are caused by a system exception that may delay or terminate processes.

To characterize the status of DNN layers' vulnerability, authors in Reference [8] classify the parameters of layers (i.e., weights and activations) separately by performing FI. In this work, parameters of layers are labeled as Low-risk, Medium-risk, and High-risk if FI process into the target layers' parameters results in less than 1%, 1%~5%, and more than 5% accuracy loss, respectively.

The metric AVF (defined in Section 2.3) is adopted in References [103, 104] and expresses the probability of fault propagating to the output. These works obtain the AVF through the FI by dividing the number of faults propagated to the output by the total number of injected faults. Furthermore, authors in Reference [104] provide a formula to estimate the cross-section (defined in Section 2.3) of the configuration memory in Equation (3), where the obtained *AVF* by FI is multiplied by the number of bits utilized by the design times the cross-section of bits of the configuration memory. This calculation can lead to further reliability metrics that authors present in Reference [104].

$$\sigma = AVF \times (\#UtilizedBits) \times \left(\frac{\sigma_{static}}{\#MemBits}\right) \tag{3}$$

In this regard, Reference [105] estimates the SER of HyperRam saving the weights similar to Equation (3) based on the extracted information from radiation experiment reports. By providing the rate of faults likely to occur in the memory, they inject faults into the weights of CNN on an FPGA accelerator.

Moreover, Reference [95] expressed the reliability of the neural network with n layers ($L_1, L_2, \ldots, L_n$) that are implemented serially as different modules on the FPGA, as an exponential distribution in Equation (4).

$$R_{NN}(t) = e^{-(\lambda_{L_1}+\lambda_{L_2}+\cdots+\lambda_{L_n})t}, \tag{4}$$

where $\lambda = \frac{1}{MTTF}$ **(MTTF (Mean Time to Failure))**.

5.1.2.2 GPU Platform. In this subsection, we explore FI in DNNs in which faults are emulated and injected into the GPU. Nearly all works on this platform have studied the effect of transient faults on GPUs. Permanent faults are studied in References [137, 157–160, 179]. To perform FI on GPUs, researchers adopt an FI framework on GPUs; except in References [117, 137], which implemented their own FI process on CUDA and TensorRT [180], respectively. FI frameworks in GPUs including FlexGripPlus [181], NVBitFI [182], and CAROL-FI [183] are used in References [113–116, 120, 157], and [122], respectively. Nonetheless, an FI framework is proposed in Reference [179] adapting and customizing NVBitFI for studying permanent faults in GPUs and is leveraged in References [158–160]. Moreover, a cross-layer fault injector framework CLASSES is presented in Reference [184] to inject SEUs at the architecture level, enabling study of the corresponding fault effects in Reference [112]. In all works, the rate of injected faults and the number of experiments in the target locations varies and depends on the confidence level and error margin, as mentioned in References [10, 44, 109, 121, 122].

SASSIFI [185] is the most frequently used framework for FI into GPUs running DNNs, which is used in References [10, 44, 109–111, 118, 119, 121]. This framework is developed by NVIDIA to conduct fault injections and is a powerful framework with different fault models covering various locations of GPUs and provides extensive reliability evaluation metrics. The studies that use SASSIFI for fault injection investigate the effect of transient faults with SASSIFI's bit-flip model into the **ISA (Instruction Set Architecture)** visible states, including general-purpose registers, memory values' predicate registers, and condition registers in a single or multiple thread.

***Evaluation*:** Reliability evaluation of DNNs in GPUs is carried out more extensively than in other platforms. Nearly all works have classified injected faults [10, 44, 109–111, 114–116, 118, 119, 121, 122, 137, 157, 159, 160]. The general model for classifying faults in the mentioned works is as follows:

— **Masked:** Fault does not affect the output,
— **SDC:** Output confidence score differs from that of the golden model,
— **DUE:** The program hangs or the system reboots (also called *Crash* in References [10, 121]).

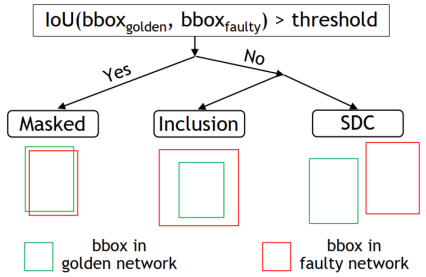

Fig. 13. Fault classification in the object detection task based on bounding boxes [137].

Furthermore, SDC is also categorized regarding the effect of faults on the accuracy of the DNN for the object recognition task in References [44, 109]. They define three categories of SDCs based on the effect of faults on the output confidence score and ranking of objects:

— **Non-critical:** Output confidence score changed, and no misclassification occurred and no objects ranking modified,
— **Light-critical:** Objects ranking modified, and no misclassification occurred,
— **Critical:** Impacted the output confidence score and caused misclassification.

However, the fault classification of SDCs proposed in Reference [122] is beyond the classic SDCs and is based on the impact of faults on the precision and recall for object detection tasks in a self-driving car, as follows:

— **Non-critical**: Precision maintains larger than 90% (a new object is detected that is not in the original classification) and recall remains 100% (all previous objects are detected).
— **Critical:** Precision is lower than 90% (many wrong objects detected) and recall is not 100% (real objects are not detected).

Furthermore, new classes of faults are presented in Reference [137], which considers the margins of the bounding box in the DNN for object detection. The authors compare the overlaps of the bounding box of the detected objects in each image for golden and faulty models and categorize the SDCs based on a threshold. Their fault classification method is depicted in Figure 13.

Vulnerability factors are also adopted to analyze the reliability of DNNs on GPU platform [10, 44, 109, 110, 114, 118, 119, 121, 122]. Vulnerability factors express the probability of propagating faults from a particular component to the output. Since faults may be injected into different locations, the vulnerability factor of the location (in different abstraction levels from architecture to program) can be measured. In this regard, **Kernel Vulnerability Factor (KVF)** [109, 118], **Layer Vulnerability Factor (LVF)** [109, 112, 118], **Instruction Vulnerability Factor (IVF)** [109, 110, 119], **Program Vulnerability Factor (PVF)** [10, 44, 109, 121], Operation Vulnerability Factor [116], and **Architecture Vulnerability Factor (AVF)** [10, 44, 109, 113, 114, 121, 122] have been presented. These metrics provide a thorough understanding of the vulnerability of each location either in DNN or in GPU.

5.1.2.3 Processors Platform. DNNs exploit processors mostly for IoT and edge applications. The research works in which faults are emulated on multi-core processors running DNNs are reviewed in this subsection. Soft errors in the register file of ARM processors running DNNs have been studied extensively in References [36, 92, 123–130]. The vulnerability of instructions is studied in Reference [130]. To emulate faults modeling soft errors in target processors, ARM-FI is developed and adopted in References [128–130] and SOFIA [92] is exploited in

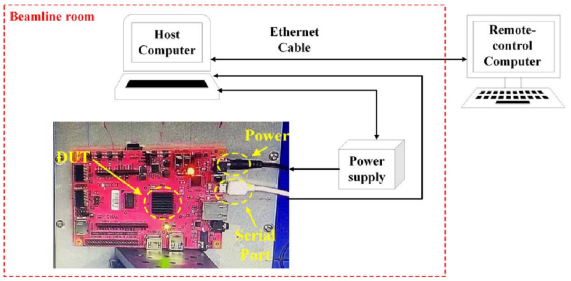

Fig. 14. Block diagram of the setup of beam experiment in Reference [108].

References [36, 92, 123–127] as fault injection frameworks. Each of the aforementioned fault injectors enables fault emulation in different components of processors.

***Evaluation*:** All works in this class have evaluated the reliability by fault classification. The classification is performed similarly to the general scheme of classifying faults in the previous platforms (Masked, tolerable SDC, critical SDC, and DUE).

Furthermore, References [36, 92] classify the faults in an object detection task for autonomous vehicles as:

— **Incorrect probability:** All objects detected correctly with different output confidence scores,
— **Wrong detection:** Misclassification or missing an object,
— **No prediction:** No object detection.

Mean Work To Failure (MWTF) is also exploited as a reliability metric to show the amount of work a neural network can perform until meeting a failure, as:

$$MWTF = \frac{1}{executiontime \times AVF_{critical-faults}}, \tag{5}$$

where $AVF_{critical-faults}$ is the probability of an erroneous classification due to faults. MWTF is adopted as a relationship between performance and reliability in References [129, 130]. AVF is obtained as the reliability metric for the register file in References [124, 129, 130]. PVF is leveraged to express the vulnerability of operations and instructions in Reference [130].

5.1.3 Irradiation. The most realistic way of fault injection is to irradiate the devices under the beam of particles, e.g., neutron or ion. In this subsection, the research works that study the reliability of DNN accelerators, i.e., FPGA and GPU under radiation, are described.

5.1.3.1 FPGA Platform. Zynq SoCs have been examined under radiation tests to assess the reliability of DNNs in References [95, 96, 103, 106, 108, 133, 134]. FPGAs are irradiated with neutrons in References [95, 96, 103, 108, 131–133] and with protons in Reference [135]. References [132] and [135] have applied fault-aware training to DNNs and studied its impact under radiation. HyperRAM, which includes constant and dynamic variables (e.g., weights and biases) is bombarded with ionizing particles in References [106, 134]. The research works set up the configuration of the system before the experiment mostly based on HW/SW co-design and save the results for further analysis. Figure 14 shows an example of the setup of the FPGA irradiation.

***Evaluation*:** Radiation experiments enable reliability evaluation by SER or FIT metrics [103, 106, 108, 134]. To formulate the SER, cross-section is defined as the proportion of observed faults (*errors*) over all particles collided to the surface (*Flux*), as expressed in Equation (6) [108]. Cross-section σ is expressed as a unit of cm^2 and is the probability that a particle may cause an observable

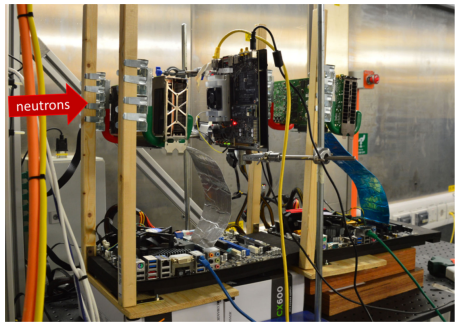

Fig. 15. Setup of neutron irradiation to GPU [10, 121, 136].

error [103]. The cross-section is exclusively adopted in References [131, 132].

$$\sigma = errors/Flux \tag{6}$$

The cross-section can lead to SER or FIT calculation by getting multiplied by the particle flux that the device will experience in the environment (ϕ). SER represents the number of failures of the device in 10^9 hours as shown in Equation (7).

$$SER = \sigma \times \phi \tag{7}$$

Most research works that study irradiation on FPGAs evaluate the reliability of devices under test by the above metrics. In addition, some works classify the faults radiated into FPGA by observing the outputs [103, 133, 135]. Here, both works provide fault classification based on output confidence scores of the neural network. Reference [103] sets up an HW/SW co-design implementation on a target board and identifies the faults causing no misclassification (tolerable) and misclassification (critical). Thereafter, the FIT of different classes of faults is obtained. References [133, 135] also present the cross-sections of the device for different classes of faults (including tolerable errors, critical errors, and crashes). Moreover, the reliability is estimated by the aforementioned metrics in Reference [95] as expressed in Equation (4).

5.1.3.2 GPU Platform. Reliability of DNNs on GPUs is assessed under neutron beam radiation in References [10, 115, 117, 121, 122, 136, 137]. All GPUs under test are manufactured by NVIDIA and have different architectures. They also provide tests by enabling and disabling ECC configurations, and different data representations. Each work has specified flux of neutrons and radiation time, e.g., Reference [137] tests the GPU equivalent to 2,000 years of exposure to terrestrial neutron, or Reference [10] reports data that cover more than 110,000 years of GPU operation. Figure 15 illustrates the radiation test setup in References [10, 121, 136].

***Evaluation*:** Research works of this group present reliability evaluation of DNNs on GPUs by FIT as well as fault classification similar to the works on FPGAs radiation. Authors in References [10, 121] identify faults that caused SDC and Crash and report their FIT, separately. References [115] and [122] report FIT of faults caused SDC and DUE separately in different data representations of the DNN, and in Reference [137] irradiated faults are classified based on Figure 13. SDC rate is also the adopted evaluation metric in Reference [117].

5.1.3.3 TPU Platform. The reliability of Google's **Tensor Processing Unit (TPU)** is studied under neutron beam radiation in References [139] and [138]. These works experimented Coral TPU chip, a low-power accelerator for DNNs, with several neural networks for image classification and object detection tasks.

Evaluation: The research works performing radiation experiments on Coral TPU have evaluated the reliability by FIT and cross-section as well as by fault classification. In this regard, SDC and DUE fault effects are reported based on FIT and cross-section.

5.2 Analytical Methods

Analytical methods in reliability assessment model the reliability mathematically and do not inject faults into the platform to be simulated to evaluate the reliability. These methods rely on the function and algorithm of DNNs, and if needed, also consider the structure of the accelerator. Nevertheless, they carry out fault injection to assess the efficacy of the methods. For the sake of generalization, all works in this group analyze the relations of neurons and layers to find their effect and contribution to the output. In this regard, they estimate the vulnerability of neurons and analyze how a faulty neuron may impact the output to find critical neurons. Therefore, they link the reliability of the network with the vulnerability of its neurons and provide an analytical model of calculating the reliability for DNNs.

We have identified four approaches in analytical methods:

— **Layerwise Relevance Propagation (LRP)**-based analysis [186–190],
— Gradient-based analysis [191–194],
— Estimation-based analysis [192, 193, 195],
— ML-based analysis [196].

In the first approach, DNNs are analyzed based on an algorithm called Layerwise Relevance Propagation (LRP) that leads to obtaining critical scores for neurons/fmaps. The second approach is based on the gradients of weights/fmaps with respect to the output leading to their sensitivity. Research works in the third approach estimate the vulnerability of DNNs by finding correlations between some information from DNNs and the vulnerability of layers/fmaps. In the last approach, ML-based techniques are adopted in the context of fault analysis in DNNs.

In the LRP-based analysis, a hypothesis is raised in Reference [189] proposing that the higher the contribution of neurons to the DNN's output, the more impact they have on the classification accuracy. Accuracy loss is one of the most important metrics in the reliability evaluation. Therefore, the more impact a neuron has on the accuracy, the more vulnerable it is, which means it has more influence on the reliability of the network, consequently. Hence, the authors adopted the LRP algorithm to obtain the value of the contribution of each neuron to the output. LRP indicates the proportion of each connected neuron in constructing the value of the target neuron and calculates this ratio for all neurons from the last layers to the first. LRP specifies $R_{i,j}(y_0, t)$ for each neuron j in layer i, which is its output contribution score between 0 and 1 with the input y_0 and output class t. Then, the average score of each neuron over the entire training set of M inputs is obtained representing the resilience of the corresponding neuron as Equation (8).

$$r_{i,j} = \frac{M}{\sum_{m=0}^{M-1} R_{i,j}(y_{0,m}, t_m)} \tag{8}$$

Thereafter, the sorted list of neurons regarding their $r_{i,j}$ represents the most to least vulnerable neurons that can lead to protecting the most vulnerable neurons to improve reliability. Furthermore, by this analytical method, another reliability improvement method is presented in Reference [190] based on balancing the resilience distribution inside the DNN. Similarly, Reference [186] proposes an approach to extract the saliency or importance of each neuron and proposes a mapping scheme for neurons on PEs of a systolic array to minimize the score of corrupted weights.

Authors in Reference [187] extend the LRP algorithm based on different output classes of input images and provide the list of neurons' resilience scores (score maps) for individual classes

separately, as well as the score map of the whole network regardless of the output classes. Then, all sorted score maps are combined in descending order to set the maximum score for each corresponding neuron. Subsequently, a scheduling algorithm is applied to map neurons to PEs of an MPSoC based on the score maps.

In gradient-based analysis, three papers are identified. Explainable AI, which explains how the network computes the output by the input, is exploited in Reference [194] to obtain the sensitivity of layers and the importance of weights. This work defines the sensitivity of layers in compliance to the difference of the two highest output confidence scores of the last layer. Therefore, they obtain the average sensitivity of all layers and relate it to the importance of weights. They provide the most important weights and their critical bits consequently to be protected.

The sensitivity of filters and weights are analyzed in Reference [191], which refers to the amount of accuracy drop with bit-flip occurrence in weights. In the proposed method in this paper, the gradient of weights with respect to the output is calculated over a dataset considering a cost function. Also, the expectation for the probability of weights to be faulty is obtained as a noise measurement (ε_w). The sensitivity of a weight w is measured as Equation (9).

$$Sensitivity_w = gradient_w \times \varepsilon_w \tag{9}$$

Sensitivity analysis in this work leads to allocation of robust hardware to the more sensitive weights.

References [192, 193] have presented three gradient-based approaches for vulnerability estimation of fmaps in a DNN. *Gradient* approach considers the absolute values of fmaps' gradients with respect to the cross-entropy loss at the output in a backpropagation as the vulnerability of fmaps. *Gain* approach measures the noise gain by obtaining the expectation for a set of corrupted neurons affecting the DNN's accuracy based on the derivatives of outputs with respect to the neurons over a set of data and the variance of noise source. *Modified Gain* is also proposed based on the *Gain* approach to violate the independence between neurons and noise. The three mentioned approaches evaluate the vulnerability of fmaps in a DNN.

Authors in References [192, 193] also presented three estimation-based approaches for the vulnerability of fmaps. They estimate the relative fmaps' vulnerability by calculating the *max neuron value*, *fmap range*, and *average L2* over the input samples. They have provided approximate yet scalable and fast approaches to estimate the vulnerability of fmaps.

Reference [195] presents an equation to estimate the misclassification rate of CNNs in case of soft error occurrence in a specific layer. The authors consider any operation resulting in a non-zero value as a critical computation, since soft errors may corrupt their results. The estimation is based on the proportion of critical operations (*Crit_OPs*) in the target layer i and subsequent layers relative to all operations in those layers, to model the misclassification rate (*SERN*) in a CNN with n layers. Equation (10) provides a representation of this estimation.

$$SERN = \frac{Crit\_OPs_i + \sum_{i+1}^{n} OPs}{\sum_{i}^{n} OPs} \tag{10}$$

An ML-based approach for analytical reliability analysis is presented in Reference [196] where **Open-Set Recognition (OSR)** methods are explored to analyze the criticality of faults in DNNs' parameters. The concept of OSR is to identify whether the output classification corresponds to the trained classes of the DNN. This concept is adapted to analyze the output logits (output of *softmax* in the last layer) of DNNs to identify the critical fault in the parameters. Four different OSR-based methods have been leveraged for this task and their efficacies are reported. In each method, a threshold for the output logits is obtained for identifying critical fault occurrence.

All the works in this group evaluate their analytical methods on the reliability by FI. The FI methods that are used in these works are similar to the FI methods presented and characterized in Section 5.1. It is shown that analytical methods can evaluate/estimate the vulnerability/sensitivity of different components of DNNs, including neurons, fmaps, and weights. Analytical methods are more lightweight than FI by far and are accelerator-agnostic. However, their analysis results can be utilized for designing robust DNN accelerators. Among the existing approaches, estimation-based analyses are faster than others while less accurate when the results are compared with FI experiments. LRP-based and gradient analyses provide more accurate results close to FI experiments, yet they are faster and incur less complexity.

5.3 Hybrid Methods

In hybrid methods, both FI and analytical methods are carried out to assess the reliability of DNNs. To that end, Reference [197] proposes a reliability assessment framework called Fidelity based on a hybrid method. This framework studies the transient faults in both data and control path of accelerators. Fidelity contains fault injection in the software framework TensorFlow to obtain the probability of masking faults in the DNN. In addition, the framework is capable of analyzing the architectural model of the accelerator and mapping **Flip Flops (FFs)** of datapath and control logic to the parameters of a high-level implementation of the DNN. By the fault injection and elaborate analysis, it models the probability of activeness/inactiveness of FFs during the execution time as well as the probability of masking faults. Subsequently, the framework provides the *FIT rate* of the accelerator. Furthermore, the framework is validated by analyzing the NVDLA [198], i.e., an open-source NVIDIA's DNN accelerator. To further improve this method, a software model for NVDLA is proposed in Reference [199] to enable reliability study of accelerators at the software level and provide a more accurate, more hardware-aware, and faster method to obtain *FIT rate* of the accelerator.

Zhang et al. [200] propose a hybrid of ML-based analysis and FI to estimate the vulnerability of all parameters in DNNs by a low number of fault injections. The proposed method involves selecting a set of random parameters of the DNN and evaluating their vulnerabilities by injecting bitflip faults and measuring the accuracy loss. Thereafter, some features for the selected parameters (absolute value, gradient, calculation times, and layer location) are extracted. A random forest as a machine learning approach is trained and tested using the features and vulnerability of the corresponding parameters so when it reaches a high accuracy, it can be used for vulnerability estimation of the entire set of parameters.

6 DISCUSSION

In this section, we will first discuss the reliability assessment methods for DNNs based on the works reviewed and presented in Section 5. Then, we will summarize the current status in the three main categories of reliability assessment: FI, analytical, and hybrid methods, respectively, and address their pros and cons in the research domain of this literature review. Thereafter, we will present a qualitative comparison of different reliability assessment methods for DNNs. Last, we will list the open challenges as well as major potential research directions for the future.

Table 2 lists the pros and cons of all the methods categorized in this work and described in Section 5.

Of the reviewed papers, FI, as a conventional method for reliability assessment, is frequently used for evaluating the DNNs' reliability. FI provides realistic results about how faults impact the system's execution. FI methods can be conducted for modeling various faults that can be injected at the different locations in the platform for reliability evaluation. Moreover, they are applicable to any platform at any system abstraction level and provide various reliability evaluations based

Table 2. Pros and Cons of Reliability Assessment Methods for DNNs

Method	Pros	Cons
Fault Simulation	– Low design time and fast execution in high-level software implementations – Adoptable for various DNNs, DHA models, and fault models – Enabling reliability study of variations of DNNs under approximation, quantization, encryption, etc. – The availability of open-source frameworks for high-level software simulation – No need for special facilities and capable of being run on regular PCs – Enabling a fast evaluation of reliability enhancement methods at high-level software implementations – Providing various reliability evaluation metrics	– High time complexity to achieve a sufficient confidence level – Not realistic model of fault effects in high-level software implementations – Inaccurate results at high-level software implementations – Time-consuming design and development for HDL implementations
Fault Emulation	– Providing realistic reliability analysis of DHA – Enabling experiments for real conditions of DHA operation – Providing full access to possible locations of the DHA for FI – Enabling realistic studying of faults in datapath – Providing fault-tolerant designs and evaluating them directly – Providing several evaluation metrics and fault classifications	– Time-consuming design and development – Need for the physical DHA – Different platforms need their own specific design and development to perform FI – Need for platform-specific frameworks for FI
Irradiation	– Performing realistic experiments as real physical faults are injected into the chip – Suitable for developing fault models – Enabling the study for validating simulation and emulation approaches – Providing the real behavior of the DHA when faced with a physical effect	– Need for specific facilities for performing radiation – Low control over accuracy of fault injection in terms of number and locations of occurred faults – Lack of the visibility of fault propagation
Analytical	– Implementable at software-level – Scalable and less complex than FI – Leading to fault-tolerant hardware designs – Providing information for algorithm-level resiliency for DNNs – DHA-agnostic	– Not providing quantitative evaluation metrics – Not considering DHA models – Inaccurate in estimating the vulnerabilities of DNN components (neurons, fmaps, etc.)
Hybrid	– Combining fast FI with an analytical approach – Capability of reliability study for DHAs – Possibility of evaluation by either vulnerability estimation or quantitative metrics	– Need for detailed information of the DHA (depending on the method) – Accuracy of the results could be low (depending on the method)

on metrics and fault classifications. Therefore, many research works choose FI as their primary method of DNNs' reliability assessment. Nevertheless, FI methods are accompanied by a prohibitively high complexity due to the need to consider several cases for fault occurrence and to iteratively repeat the executions.

Analytical methods have been proposed as a way to cope with the high complexity of FI methods. These methods study the function of DNNs and assess the model's reliability using mathematical equations, leading to less complex approaches. Since analytical methods are developed mathematically, they have the potential to be generalized and adapted to various DNNs. Notably, analytical methods have the potential to be exploited in the reliability assessment of the training phase.

Table 3. Qualitative Analysis Comparing Different Reliability Assessment Methods for DNNs

	Fault injection	Analytical	Hybrid
Time Complexity	High	Low to Moderate	Moderate
HDA-aware	Yes	No	Yes
Leading to fault-tolerant design	Yes	Yes	Yes
Fault models variety	All fault models	Few fault models	Few fault models
Implementation system level	Software and hardware	Software	Software
Evaluation accuracy	Moderate to high	Low to moderate	Moderate
Development time	Low to Moderate	Moderate	High
Evaluation metrics	Accuracy loss Fault classification Vulnerability factors SDC rate Reliability equations	Criticality scores Sensitivity Vulnerability estimation	FIT Rate Vulnerability estimation

However, current analytical methods do not consider the accelerator models, and there is a gap in the use of reliability evaluation metrics. While this survey identifies a relatively small number of works relying on analytical methods for DNNs' reliability assessment, the future of research in this area should pay greater attention to the potential of analytical methods.

Finally, hybrid methods combine the strength of both FI and analytical methods. By applying analysis of the network or the accelerator in addition to conducting fault injection, hybrid methods are capable of obtaining a comprehensive and realistic evaluation of reliability. Although a limited number of research works are identified in this category in the present survey, yet there is a huge space to explore for proposing new hybrid methods in the future. Table 3 presents a qualitative comparison between the categorized methods of reliability assessment for DNNs regarding the papers included in this survey.

The analysis of statistics presented in Figure 9 highlights that the majority of the identified research works employ FI to assess the DNNs' reliability. This can be attributed to the fact that, while DNNs are an emerging topic in computer science, the problem of reliability has been a classic issue for a long time. In addition, the investigation of reliability over DNNs has started gaining traction since 2017, as indicated in Figure 8. As a result, it is not surprising that the early research in this area has primarily focused on conventional methods such as FI. This could be the main reason for the significant imbalance in the number of published papers across different method categories. However, in the future, the emergence of analytical and hybrid methods is expected to bridge this gap and increase their application in the field of DNN reliability assessment.

To address open challenges in reliability assessment methods for DNNs, this survey has identified the following main observations:

— Although some research works, such as Reference [201], have studied the impact of faulty data during training, no work on the reliability assessment of the training phase has been identified that considers faulty parameters or computational units. This issue should be studied in future research;
— Nearly all included works focus on CNNs, with image classification and object detection tasks excluding other types of DNNs, such as RNNs and LSTMs as well as different applications that should also be evaluated in terms of reliability;
— The survey has identified no software FI framework in hardware-aware platforms. Hence, DNN accelerator simulators could be exploited or developed for reliability assessment of DNNs in this platform;
— Fault emulation on FPGAs can take advantage of HLS designs. Therefore, a general FI framework for these platforms could be presented using HLS to minimize design time;

— Based on this survey, very few works study the reliability of the control part of DHAs, especially in FPGAs and ASICs. The control part may play a significant role in the reliability of DNN accelerators, and this should be explored in future studies;
— There is a limited number of analytical methods for DNNs reliability assessment in this survey, all of which rely on finding critical neurons for fault-tolerant designs. Also, only one work tries to predict the accuracy loss caused by soft errors, and ML-based approaches are proposed in one work. Nevertheless, none of them can estimate the reliability of DNNs on their own or evaluate the reliability using specific metrics. ML-based algorithms can significantly assist in efficient reliability assessment, and therefore, there is a huge potential for developing new analytical methods of reliability assessment for DNNs;
— Analytical methods could be generalized for other DNNs and applications rather than considering only CNNs and image processing;
— Hybrid methods appear to be powerful and capable of being exploited for developing reliability assessment frameworks. They can be one of the major methods for reliability assessment of DNNs in future works;
— Several FI research works carry out accuracy loss and fault classification as an evaluation of reliability. Also, some works considered FIT. However, there is still an urgent need to present DNN-specific metrics for reliability evaluation.

As an outcome of this survey, in addition to the listed open challenges, the major possible research directions for future studies in this domain are addressed below:

— Although analytical and hybrid methods have potential in the literature, they are not evolved to the extent that their effectiveness can be fully realized. Existing methods have shown that analytical and hybrid methods are capable of assessing the DNNs' reliability as realistically as FI and lead to effective fault-tolerant designs. Moreover, ML-based approaches in conjunction with analytical and hybrid methods are emerging. Therefore, researchers can be directed to develop novel analytical and hybrid methods, especially those that adopt ML-based algorithms, for reliability assessment of DNNs that are faster, less complex, more scalable, and more specific to DNNs than the conventional FI approaches.
— Bringing reliability as a classical issue into an emerging topic such as DNNs requires new tools to respond to the requirements of the new domain. Therefore, the new research not only needs to adopt commonly used metrics in the reliability domain, but also requires the introduction and proposal of novel DNNs-specific reliability evaluation metrics.
— There are several IoT and edge applications for DNNs emerging day by day, and reliability is not only a concern for safety-critical applications. New research can focus on the unstudied applications of DNNs while taking reliability into consideration.

7 CONCLUSION

DNNs are being utilized in an increasingly diverse range of applications in our daily lives. Consequently, their deployment in safety-critical applications has emerged to be expanding day by day. However, threats to reliability are one of the major issues that they experience in the real world. To address this, several studies have been published in recent years to assess the reliability of DNNs, with or without the use of accelerators, resulting in the development of various assessment methods. In this work, we conduct a systematic literature review to present a categorization of the reliability assessment methods for DNNs.

Out of the 139 papers related to the subject of the review, three major approaches to reliability assessment of DNNs were identified, i.e., Fault Injection, Analytical, and Hybrid methods. Since the majority of works assess the reliability using conventional fault injection methods, the related

works relying on FI methods are characterized based on different approaches and platforms. In addition, we have addressed the advantages and disadvantages of the different methods and highlighted the open challenges that may become the focus of future studies in this domain. Based on the analysis of this survey, future research could focus on developing lightweight, DNN-specific analytical and hybrid methods for assessing reliability, as well as providing new quantitative evaluation metrics that take into account emerging applications for DNNs.

REFERENCES

[1] Alberto Bosio, Ian O'Connor, Marcello Traiola, Jorge Echavarria, Jürgen Teich, Muhammad Abdullah Hanif, Muhammad Shafique, Said Hamdioui, Bastien Deveautour, Patrick Girard, Arnaud Virazel, and Koen Bertels. 2021. Emerging computing devices: Challenges and opportunities for test and reliability. In *IEEE European Test Symposium (ETS'21)*. IEEE, 1–10.

[2] Håkan Forsberg, Joakim Lindén, Johan Hjorth, Torbjörn Månefjord, and Masoud Daneshtalab. 2020. Challenges in using neural networks in safety-critical applications. In *AIAA/IEEE 39th Digital Avionics Systems Conference (DASC'20)*. IEEE, 1–7.

[3] Alessandra Nardi and Antonino Armato. 2017. Functional safety methodologies for automotive applications. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD'17)*. IEEE, 970–975.

[4] Younis Ibrahim, Haibin Wang, Junyang Liu, Jinghe Wei, Li Chen, Paolo Rech, Khalid Adam, and Gang Guo. 2020. Soft errors in DNN accelerators: A comprehensive review. *Microelectron. Reliab.* 115 (2020), 113969.

[5] Muhammad Shafique, Mahum Naseer, Theocharis Theocharides, Christos Kyrkou, Onur Mutlu, Lois Orosa, and Jungwook Choi. 2020. Robust machine learning systems: Challenges, current trends, perspectives, and the road ahead. *IEEE Des. Test* 37, 2 (2020), 30–57.

[6] Stéphane Burel, Adrian Evans, and Lorena Anghel. 2021. MOZART: Masking outputs with zeros for architectural robustness and testing of DNN accelerators. In *IEEE 27th International Symposium on On-Line Testing and Robust System Design (IOLTS'21)*. IEEE, 1–6.

[7] Krishna Teja Chitty-Venkata and Arun K. Somani. 2020. Model compression on faulty array-based neural network accelerator. In *IEEE 25th Pacific Rim International Symposium on Dependable Computing (PRDC'20)*. IEEE, 90–99.

[8] Navid Khoshavi, Arman Roohi, Connor Broyles, Saman Sargolzaei, Yu Bi, and David Z. Pan. 2020. SHIELDeNN: Online accelerated framework for fault-tolerant deep neural network architectures. In *57th ACM/IEEE Design Automation Conference (DAC'20)*. IEEE, 1–6.

[9] Elbruz Ozen and Alex Orailoglu. 2020. Low-cost error detection in deep neural network accelerators with linear algorithmic checksums. *J. Electron. Test.* 36, 6 (2020), 703–718.

[10] Fernando Fernandes dos Santos, Pedro Foletto Pimenta, Caio Lunardi, Lucas Draghetti, Luigi Carro, David Kaeli, and Paolo Rech. 2018. Analyzing and increasing the reliability of convolutional neural networks on GPUs. *IEEE Trans. Reliab.* 68, 2 (2018), 663–677.

[11] Sparsh Mittal. 2020. A survey on modeling and improving reliability of DNN algorithms and accelerators. *J. Syst. Archit.* 104 (2020), 101689.

[12] Annachiara Ruospo, Ernesto Sanchez, Lucas Matana Luza, Luigi Dilillo, Marcello Traiola, and Alberto Bosio. 2023. A survey on deep learning resilience assessment methodologies. *Computer* 56, 2 (2023), 57–66.

[13] Fei Su, Chunsheng Liu, and Haralampos-G. Stratigopoulos. 2023. Testability and dependability of AI hardware: Survey, trends, challenges, and perspectives. *IEEE Des. Test* 40, 2 (2023).

[14] Cesar Torres-Huitzil and Bernard Girau. 2017. Fault and error tolerance in neural networks: A review. *IEEE Access* 5 (2017), 17322–17341.

[15] Antonio Cicchetti, Federico Ciccozzi, and Alfonso Pierantonio. 2019. Multi-view approaches for software and system modelling: A systematic literature review. *Softw. Syst. Model.* 18, 6 (2019), 3207–3233.

[16] Mathieu Lavallée, Pierre-N. Robillard, and Reza Mirsalari. 2013. Performing systematic literature reviews with novices: An iterative approach. *IEEE Trans. Educ.* 57, 3 (2013), 175–181.

[17] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S. Emer. 2017. Efficient processing of deep neural networks: A tutorial and survey. *Proc. IEEE* 105, 12 (2017), 2295–2329.

[18] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.

[19] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. ImageNet classification with deep convolutional neural networks. *Adv. Neural Inf. Process. Syst.* 25 (2012), 1097–1105.

[20] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going deeper with convolutions. In *IEEE Conference on Computer Vision and Pattern Recognition*. 1–9.

[21] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).
[22] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*. 770–778.
[23] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. 2016. You only look once: Unified, real-time object detection. In *IEEE Conference on Computer Vision and Pattern Recognition*. 779–788.
[24] C. J. B. Yann, Y. LeCun, and C. Cortes. The MNIST DATABASE of Handwritten Digits. Retrieved from: http://yann.lecun.com/exdb/mnist/
[25] A. Krizhevsky, v. Nair, and G. Hinton. 2009. The CIFAR-10 Dataset. Retrieved from: https://www.cs.toronto.edu/~kriz/cifar.html
[26] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. ImageNet: A large-scale hierarchical image database. In *IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 248–255.
[27] Moritz Menze and Andreas Geiger. 2015. Object scene flow for autonomous vehicles. In *IEEE Conference on Computer Vision and Pattern Recognition*. 3061–3070.
[28] Mark Everingham, Luc Van Gool, Christopher K. I. Williams, John Winn, and Andrew Zisserman. 2010. The Pascal visual object classes (VOC) challenge. *Int. J. Comput. Vis.* 88, 2 (2010), 303–338.
[29] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2017. Quantized neural networks: Training neural networks with low precision weights and activations. *J. Mach. Learn. Res.* 18, 1 (2017), 6869–6898.
[30] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2016. Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or −1. *arXiv preprint arXiv:1602.02830* (2016).
[31] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. 265–283.
[32] Keras: The Python deep learning API. 2015. Retrieved from: https://keras.io/
[33] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An imperative style, high-performance deep learning library. *Adv. Neural Inf. Process. Syst.* 32 (2019), 8026–8037.
[34] Joseph Redmon. Darknet: Open Source Neural Networks in C. Retrieved from: http://pjreddie.com/darknet/
[35] Tiny-CNN Framework. 2012. Retrieved from: https://github.com/tiny-dnn/tiny-dnn
[36] Geancarlo Abich, Jonas Gava, Ricardo Reis, and Luciano Ost. 2020. Soft error reliability assessment of neural networks on resource-constrained IoT devices. In *27th IEEE International Conference on Electronics, Circuits and Systems (ICECS'20)*. IEEE, 1–4.
[37] Manar Abu Talib, Sohaib Majzoub, Qassim Nasir, and Dina Jamal. 2021. A systematic literature review on hardware implementation of artificial intelligence algorithms. *J. Supercomput.* 77 (2021), 1897–1938.
[38] Kaiyuan Guo, Shulin Zeng, Jincheng Yu, Yu Wang, and Huazhong Yang. 2019. A survey of FPGA-based neural network inference accelerators. *ACM Trans. Reconfig. Technol. Syst.* 12, 1 (2019), 1–26.
[39] Neng Hou, Xiaohu Yan, and Fazhi He. 2019. A survey on partitioning models, solution algorithms and algorithm parallelization for hardware/software co-design. *Des. Automat. Embed. Syst.* 23, 1 (2019), 57–77.
[40] Meriam Dhouibi, Ahmed Karim Ben Salem, Afef Saidi, and Slim Ben Saoud. 2021. Accelerating deep neural networks implementation: A survey. *IET Comput. Digit. Techniq.* 15, 2 (2021), 79–96.
[41] Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre-luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, C. Richard Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Daniel Killebrew, Andy Koch, Naveen Kumar, Steve Lacy, James Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire Mahony, Kieran Miller, Rahul Nagarajan, Ravi Narayanaswami, Ray Ni, Kathy Nix, Thomas Norrie, Mark Omernick, Narayana Penukonda, Andy Phelps, Jonathan Ross, Matt Ross, Amir Salek, Emad Samadiani, Chris Severn, Gregory Sizikov, Matthew Snelham, Jed Souter, Dan Steinberg, Andy Swing, Mercedes Tan, Gregory Thorson, Bo Tian, Horia Toma, Erick Tuttle, Vijay Vasudevan, Richard Walter, Walter Wang, Eric Wilcox, and Doe Hyun Yoon. 2017. In-datacenter performance analysis of a Tensor Processing Unit. In *44th Annual International Symposium on Computer Architecture*. 1–12.
[42] Diksha Moolchandani, Anshul Kumar, and Smruti R. Sarangi. 2021. Accelerating CNN inference on ASICs: A survey. *J. Syst. Archit.* 113 (2021), 101887.

[43] Jon Perez-Cerrolaza, Jaume Abella, Leonidas Kosmidis, Alejandro J. Calderon, Francisco Cazorla, and Jose Luis Flores. 2022. GPU devices for safety-critical systems: A survey. *Comput. Surv.* 55, 7 (2022), 1–37.
[44] Younis Ibrahim, Haibin Wang, Man Bai, Zhi Liu, Jianan Wang, Zhiming Yang, and Zhengming Chen. 2020. Soft error resilience of deep residual networks for object recognition. *IEEE Access* 8 (2020), 19490–19503.
[45] Liangzhen Lai, Naveen Suda, and Vikas Chandra. 2018. CMSIS-NN: Efficient neural network kernels for arm cortex-M CPUs. *arXiv preprint arXiv:1801.06601* (2018).
[46] Mohammad Saeid Mahdavinejad, Mohammadreza Rezvan, Mohammadamin Barekatain, Peyman Adibi, Payam Barnaghi, and Amit P. Sheth. 2018. Machine learning for Internet of Things data analysis: A survey. *Digit. Commun. Netw.* 4, 3 (2018), 161–175.
[47] Ramon Sanchez-Iborra and Antonio F. Skarmeta. 2020. TinyML-enabled frugal smart objects: Challenges and opportunities. *IEEE Circ. Syst. Mag.* 20, 3 (2020), 4–18.
[48] Robert C. Baumann. 2005. Radiation-induced soft errors in advanced semiconductor technologies. *IEEE Trans. Device Mater. Reliab.* 5, 3 (2005), 305–316.
[49] G. C. K. Y. Chen, K. Y. Chuah, M. F. Li, Daniel S. H. Chan, C. H. Ang, J. Z. Zheng, Y. Jin, and D. L. Kwong. 2003. Dynamic NBTI of PMOS transistors and its impact on device lifetime. In *41st IEEE International Reliability Physics Symposium*. IEEE, 196–202.
[50] Shekhar Borkar. 2005. Designing reliable systems from unreliable components: The challenges of transistor variability and degradation. *IEEE Micro* 25, 6 (2005), 10–16.
[51] Israel Koren and C. Mani Krishna. 2007. Fault-tolerant Systems. (2007).
[52] Barry Johnson. 1984. Fault-tolerant microprocessor-based systems. *IEEE Micro* 4, 06 (1984), 6–21.
[53] Arijit Biswas, Paul Racunas, Razvan Cheveresan, Joel Emer, Shubhendu S. Mukherjee, and Ram Rangan. 2005. Computing architectural vulnerability factors for address-based structures. In *32nd International Symposium on Computer Architecture (ISCA'05)*. IEEE, 532–543.
[54] Mohammad Eslami, Behnam Ghavami, Mohsen Raji, and Ali Mahani. 2020. A survey on fault injection methods of digital integrated circuits. *Integration* 71 (2020), 154–163.
[55] Annachiara Ruospo, Lucas Matana Luza, Alberto Bosio, Marcello Traiola, Luigi Dilillo, and Ernesto Sanchez. 2021. Pros and cons of fault injection approaches for the reliability assessment of deep neural networks. In *IEEE 22nd Latin American Test Symposium (LATS'21)*. IEEE, 1–5.
[56] Régis Leveugle, A. Calvez, Paolo Maistri, and Pierre Vanhauwaert. 2009. Statistical fault injection: Quantified error and confidence. In *Design, Automation & Test in Europe Conference & Exhibition*. IEEE, 502–506.
[57] Alfredo Benso and Stefano DiCarlo. 2011. The art of fault injection. *J. Contr. Eng. Appl. Inform.* 13, 4 (2011), 9–18.
[58] Annachiara Ruospo, Angelo Balaara, Alberto Bosio, and Ernesto Sanchez. 2020. A pipelined multi-level fault injector for deep neural networks. In *IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT'20)*. IEEE, 1–6.
[59] Muhammad Salman Ali, Tauhid Bin Iqbal, Kang-Ho Lee, Abdul Muqeet, Seunghyun Lee, Lokwon Kim, and Sung-Ho Bae. 2020. ERDNN: Error-resilient deep neural networks with a new error correction layer and piece-wise rectified linear unit. *IEEE Access* 8 (2020), 158702–158711.
[60] Chandramouli Amarnath, Mohamed Mejri, Kwondo Ma, and Abhijit Chatterjee. 2022. Soft error resilient deep learning systems using neuron gradient statistics. In *IEEE 28th International Symposium on On-Line Testing and Robust System Design (IOLTS'22)*. IEEE, 1–7.
[61] Austin P. Arechiga and Alan J. Michaels. 2018. The effect of weight errors on neural networks. In *IEEE 8th Annual Computing and Communication Workshop and Conference (CCWC'18)*. IEEE, 190–196.
[62] Austin P. Arechiga and Alan J. Michaels. 2018. The robustness of modern deep learning architectures against single event upset errors. In *IEEE High Performance Extreme Computing Conference (HPEC'18)*. IEEE, 1–6.
[63] Stéphane Burel, Adrian Evans, and Lorena Anghel. 2021. Zero-overhead protection for CNN weights. In *IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT'21)*. IEEE, 1–6.
[64] Stéphane BurelT, Adrian EvansT, and Lorena Anghel. 2022. Improving DNN fault tolerance in semantic segmentation applications. In *IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT'22)*. IEEE, 1–6.
[65] Riccardo Cantoro, Nikolaos I. Deligiannis, Matteo Sonza Reorda, Marcello Traiola, and Emanuele Valea. 2020. Evaluating data encryption effects on the resilience of an artificial neural network. In *IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT'20)*. IEEE, 1–4.
[66] Zitao Chen, Guanpeng Li, and Karthik Pattabiraman. 2021. A low-cost fault corrector for deep neural networks through range restriction. In *51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'21)*. IEEE, 1–13.
[67] Nikolaos Ioannis Deligiannis, Riccardo Cantoro, Matteo Sonza Reorda, Marcello Traiola, and Emanuele Valea. 2021. Towards the integration of reliability and security mechanisms to enhance the fault resilience of neural networks. *IEEE Access* 9 (2021), 155998–156012.

[68] Zhen Gao, Xiaohui Wei, Han Zhang, Wenshuo Li, Guangjun Ge, Yu Wang, and Pedro Reviriego. 2020. Reliability evaluation of pruned neural networks against errors on parameters. In *IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT'20)*. IEEE, 1–6.

[69] Behnam Ghavami, Mani Sadati, Zhenman Fang, and Lesley Shannon. 2022. FitAct: Error resilient deep neural networks via fine-grained post-trainable activation functions. In *Design, Automation & Test in Europe Conference & Exhibition (DATE'22)*. IEEE, 1239–1244.

[70] Brunno F. Goldstein, Sudarshan Srinivasan, Dipankar Das, Kunal Banerjee, Leandro Santiago, Victor C. Ferreira, Alexandre S. Nery, Sandip Kundu, and Felipe M. G. França. 2020. Reliability evaluation of compressed deep learning models. In *IEEE 11th Latin American Symposium on Circuits & Systems (LASCAS'20)*. IEEE, 1–5.

[71] Hui Guan, Lin Ning, Zhen Lin, Xipeng Shen, Huiyang Zhou, and Seung-Hwan Lim. 2019. In-place zero-space memory protection for CNN. In *33rd International Conference on Neural Information Processing Systems*. 5734–5743.

[72] Le-Ha Hoang, Muhammad Abdullah Hanif, and Muhammad Shafique. 2020. FT-ClipAct: Resilience analysis of deep neural networks and improving their fault tolerance using clipped activation. In *Design, Automation & Test in Europe Conference & Exhibition (DATE'20)*. IEEE, 1241–1246.

[73] Myeungjae Jang and Jeongkyu Hong. 2021. MATE: Memory-and retraining-free error correction for convolutional neural network weights. *J. Inf. Commun. Converg. Eng.* 19, 1 (2021), 22–28.

[74] Suyong Lee, Insu Choi, and Joon-Sung Yang. 2022. Bipolar vector classifier for fault-tolerant deep neural networks. In *59th ACM/IEEE Design Automation Conference*. 673–678.

[75] Elaheh Malekzadeh, Nezam Rohbani, Zhonghai Lu, and Masoumeh Ebrahimi. 2021. The impact of faults on DNNs: A case study. In *IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT'21)*. IEEE, 1–6.

[76] Mohamed A. Neggaz, Ihsen Alouani, Pablo R. Lorenzo, and Smail Niar. 2018. A reliability study on CNNs for critical embedded systems. In *IEEE 36th International Conference on Computer Design (ICCD'18)*. IEEE, 476–479.

[77] Mohamed A. Neggaz, Ihsen Alouani, Smail Niar, and Fadi Kurdahi. 2019. Are CNNs reliable enough for critical applications? An exploratory study. *IEEE Des. Test* 37, 2 (2019), 76–83.

[78] Elbruz Ozen and Alex Orailoglu. 2021. SNR: Squeezing numerical range defuses bit error vulnerability surface in deep neural networks. *ACM Trans. Embed. Comput. Syst.* 20, 5s (2021), 1–25.

[79] Jonathan Ponader, Kyle Thomas, Sandip Kundu, and Yan Solihin. 2021. MILR: Mathematically induced layer recovery for plaintext space error correction of CNNs. In *51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'21)*. IEEE, 75–87.

[80] Majid Sabbagh, Cheng Gongye, Yunsi Fei, and Yanzhi Wang. 2019. Evaluating fault resiliency of compressed deep neural networks. In *IEEE International Conference on Embedded Software and Systems (ICESS'19)*. IEEE, 1–7.

[81] Rizwan Tariq Syed, Markus Ulbricht, Krzysztof Piotrowski, and Milos Krstic. 2021. Fault resilience analysis of quantized deep neural networks. In *IEEE 32nd International Conference on Microelectronics (MIEL'21)*. IEEE, 275–279.

[82] Jinyu Zhan, Ruoxu Sun, Wei Jiang, Yucheng Jiang, Xunzhao Yin, and Cheng Zhuo. 2021. Improving fault tolerance for reliable DNN using boundary-aware activation. *IEEE Trans. Comput.-Aid. Des. Integ. Circ. Syst.* 41, 10 (2021), 3414–3425.

[83] Arash Azizimazreah, Yongbin Gu, Xiang Gu, and Lizhong Chen. 2018. Tolerating soft errors in deep learning accelerators with reliable on-chip memory designs. In *IEEE International Conference on Networking, Architecture and Storage (NAS'18)*. IEEE, 1–10.

[84] Brunno F. Goldstein, Victor C. Ferreira, Sudarshan Srinivasan, Dipankar Das, Alexandre S. Nery, Sandip Kundu, and Felipe M. G. França. 2021. A lightweight error-resiliency mechanism for deep neural networks. In *22nd International Symposium on Quality Electronic Design (ISQED'21)*. IEEE, 311–316.

[85] Masoomeh Jasemi, Shaahin Hessabi, and Nader Bagherzadeh. 2020. Enhancing reliability of emerging memory technology for machine learning accelerators. *IEEE Trans. Emerg. Topics Comput.* 9, 4 (2020), 2234–2240.

[86] Jae-San Kim and Joon-Sung Yang. 2019. DRIS-3: Deep neural network reliability improvement scheme in 3D die-stacked memory based on fault analysis. In *56th ACM/IEEE Design Automation Conference (DAC'19)*. IEEE, 1–6.

[87] Guanpeng Li, Siva Kumar Sastry Hari, Michael Sullivan, Timothy Tsai, Karthik Pattabiraman, Joel Emer, and Stephen W. Keckler. 2017. Understanding error propagation in deep learning neural network (DNN) accelerators and applications. In *International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–12.

[88] Wenshuo Li, Guangjun Ge, Kaiyuan Guo, Xiaoming Chen, Qi Wei, Zhen Gao, Yu Wang, and Huazhong Yang. 2020. Soft Error Mitigation for Deep Convolution Neural Network on FPGA Accelerators. In *2nd IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS'20)*. IEEE, 1–5.

[89] Elbruz Ozen and Alex Orailoglu. 2020. Boosting bit-error resilience of DNN accelerators through median feature selection. *IEEE Trans. Comput.-Aid. Des. Integ. Circ. Syst.* 39, 11 (2020), 3250–3262.

[90] Elbruz Ozen and Alex Orailoglu. 2020. Just say zero: Containing critical bit-error propagation in deep neural networks with anomalous feature suppression. In *IEEE/ACM International Conference on Computer Aided Design (ICCAD'20)*. IEEE, 1–9.

[91] Elbruz Ozen and Alex Orailoglu. 2019. Sanity-check: Boosting the reliability of safety-critical deep neural network applications. In *IEEE 28th Asian Test Symposium (ATS'19)*. IEEE, 7–75.

[92] Vitor Bandeira, Felipe Rosa, Ricardo Reis, and Luciano Ost. 2019. Non-intrusive fault injection techniques for efficient soft error vulnerability analysis. In *IFIP/IEEE 27th International Conference on Very Large Scale Integration (VLSI-SoC'19)*. IEEE, 123–128.

[93] Panayiotis Corneliou, Panagiota Nikolaou, Maria K. Michael, and Theocharis Theocharides. 2021. Fine-grained vulnerability analysis of resource constrained neural inference accelerators. In *IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT'21)*. IEEE, 1–6.

[94] Anahita Hosseinkhani and Behnam Ghavami. 2021. Improving soft error reliability of FPGA-based deep neural networks with reduced approximate TMR. In *11th International Conference on Computer Engineering and Knowledge (ICCKE'21)*. IEEE, 459–464.

[95] Fabio Benevenuti, Fabiano Libano, Vincent Pouget, Fernanda Lima Kastensmidt, and Paolo Rech. 2018. Comparative analysis of inference errors in a neural network implemented in SRAM-based FPGA induced by neutron irradiation and fault injection methods. In *31st Symposium on Integrated Circuits and Systems Design (SBCCI'18)*. IEEE, 1–6.

[96] Fabio Benevenuti, Márcio Gonçalves, Evaldo Carlos Fonseca Pereira Junior, Rafael Galhardo Vaz, Odair Lelis Gonçalez, José Rodrigo Azambuja, and Fernanda Lima Kastensmidt. 2021. Neutron-induced faults on CNN for aerial image classification on SRAM-based FPGA using softcore GPU and HLS. In *21th European Conference on Radiation and Its Effects on Components and Systems (RADECS'21)*. IEEE, 1–4.

[97] Corrado De Sio, Sarah Azimi, and Luca Sterpone. 2020. An emulation platform for evaluating the reliability of deep neural networks. In *IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT'20)*. IEEE, 1–4.

[98] Corrado De Sio, Sarah Azimi, and Luca Sterpone. 2022. FireNN: Neural networks reliability evaluation on hybrid platforms. *IEEE Trans. Emerg. Topics Comput.* 10, 2 (2022), 549–563.

[99] Boyang Du, Sarah Azimi, Corrado De Sio, Ludovica Bozzoli, and Luca Sterpone. 2019. On the reliability of convolutional neural network implementation on SRAM-based FPGA. In *IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT'19)*. IEEE, 1–6.

[100] Zhen Gao, Yi Yao, Xiaohui Wei, Tong Yan, Shulin Zeng, Guangjun Ge, Yu Wang, Anees Ullah, and Pedro Reviriego. 2022. Reliability evaluation of FPGA based pruned neural networks. *Microelectron. Reliab.* 130 (2022), 114498.

[101] Navid Khoshavi, Connor Broyles, and Yu Bi. 2020. Compression or corruption? A study on the effects of transient faults on BNN inference accelerators. In *21st International Symposium on Quality Electronic Design (ISQED'20)*. IEEE, 99–104.

[102] Navid Khoshavi, Connor Broyles, Yu Bi, and Arman Roohi. 2020. Fiji-FIN: A fault injection framework on quantized neural network inference accelerator. In *19th IEEE International Conference on Machine Learning and Applications (ICMLA'20)*. IEEE, 1139–1144.

[103] Fabiano Libano, Brittany Wilson, J. Anderson, Michael J. Wirthlin, Carlo Cazzaniga, Christopher Frost, and Paolo Rech. 2018. Selective hardening for neural networks in FPGAs. *IEEE Trans. Nuclear Sci.* 66, 1 (2018), 216–222.

[104] Fabiano Libano, Brittany Wilson, Michael Wirthlin, Paolo Rech, and John Brunhaver. 2020. Understanding the impact of quantization, accuracy, and radiation on the reliability of convolutional neural networks on FPGAs. *IEEE Trans. Nuclear Sci.* 67, 7 (2020), 1478–1484.

[105] Lucas Matana Luza, Annachiara Ruospo, Alberto Bosio, Ernesto Sanchez, and Luigi Dilillo. 2021. A model-based framework to assess the reliability of safety-critical applications. In *24th International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS'21)*. IEEE, 41–44.

[106] Lucas Matanaluza, Annachiara Ruospo, Daniel Soderstrom, Carlo Cazzaniga, Maria Kastriotou, Ernesto Sanchez, Alberto Bosio, and Luigi Dilillo. 2021. Emulating the effects of radiation-induced soft-errors for the reliability assessment of neural networks. *IEEE Trans. Emerg. Topics Comput.* 10, 4 (2021).

[107] Ioanna Souvatzoglou, Athanasios Papadimitriou, Aitzan Sari, Vasileios Vlagkoulis, and Mihalis Psarakis. 2021. Analyzing the single event upset vulnerability of binarized neural networks on SRAM FPGAs. In *IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT'21)*. IEEE, 1–6.

[108] H.-B. Wang, Y.-S. Wang, J.-H. Xiao, S.-L. Wang, and T.-J. Liang. 2021. Impact of single-event upsets on convolutional neural networks in Xilinx Zynq FPGAs. *IEEE Trans. Nuclear Sci.* 68, 4 (2021), 394–401.

[109] Khalid Adam, Izzeldin Ibrahim Mohamed, and Younis Ibrahim. 2021. A selective mitigation technique of soft errors for DNN models used in healthcare applications: DenseNet201 case study. *IEEE Access* 9 (2021), 65803–65823.

[110] Khalid Adam, Izzeldin I. Mohd, and Younis Ibrahim. 2021. Analyzing the instructions vulnerability of dense convolutional network on GPUS. *Int. J. Electric. Comput. Eng.* 11, 5 (2021), 4481–4488.

[111] Khalid Adam, Izzeldin I. Mohd, and Younis M. Younis. 2021. The impact of the soft errors in convolutional neural network on GPUs: Alexnet as case study. *Procedia Comput. Sci.* 182 (2021), 89–94.
[112] Cristiana Bolchini, Luca Cassano, Antonio Miele, and Alessandro Nazzari. 2022. Selective hardening of CNNs based on layer vulnerability estimation. In *IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT'22)*. IEEE, 1–6.
[113] Niccolò Cavagnero, Fernando Dos Santos, Marco Ciccone, Giuseppe Averta, Tatiana Tommasi, and Paolo Rech. 2022. Transient-fault-aware design and training to enhance DNNs reliability with zero-overhead. In *IEEE 28th International Symposium on On-Line Testing and Robust System Design (IOLTS'22)*. IEEE, 1–7.
[114] Josie E. Rodriguez Condia, Fernando Fernandes dos Santos, Matteo Sonza Reorda, and Paolo Rech. 2021. Combining architectural simulation and software fault injection for a fast and accurate CNNs reliability evaluation on GPUs. In *IEEE 39th VLSI Test Symposium (VTS'21)*. IEEE, 1–7.
[115] Fernando Fernandes Dos Santos, Angeliki Kritikakou, Josie E. Rodriguez Condia, Juan-David Guerrero-Balaguera, Matteo Sonza Reorda, Olivier Sentieys, and Paolo Rech. 2022. Characterizing a neutron-induced fault model for deep neural networks. *IEEE Trans. Nuclear Sci.* 70, 4 (2022).
[116] Tyler Garrett and Alan D. George. 2021. Improving dependability of onboard deep learning with resilient TensorFlow. In *IEEE Space Computing Conference (SCC'21)*. IEEE, 134–142.
[117] Siva Kumar Sastry Hari, Michael Sullivan, Timothy Tsai, and Stephen W. Keckler. 2021. Making convolutions resilient via algorithm-based error detection techniques. *IEEE Trans. Depend. Sec. Comput.* 19, 4 (2021).
[118] Younis Ibrahim, Haibin Wang, and Khalid Adam. 2020. Analyzing the reliability of convolutional neural networks on GPUs: GoogLeNet as a case study. In *International Conference on Computing and Information Technology (ICCIT'20)*. IEEE, 1–6.
[119] Younis Ibrahin, Junyang Liu, Xuanxuan Yang, Hongwei Sha, and Haibin Wang. 2020. Analyzing the impact of soft errors in deep neural networks on GPUs from instruction level. *WSEAS Trans. Syst. Contr.* 15 (2020), 699–708.
[120] Rubens Luiz Rech and Paolo Rech. 2020. Impact of layers selective approximation on CNNs reliability and performance. In *IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT'20)*. IEEE, 1–4.
[121] Fernando Fernandes dos Santos, Lucas Draghetti, Lucas Weigel, Luigi Carro, Philippe Navaux, and Paolo Rech. 2017. Evaluation and mitigation of soft-errors in neural network-based object detection in three GPU architectures. In *47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W'17)*. IEEE, 169–176.
[122] Fernando Fernandes dos Santos, Philippe Navaux, Luigi Carro, and Paolo Rech. 2019. Impact of reduced precision in the reliability of deep neural networks for object detection. In *IEEE European Test Symposium (ETS'19)*. IEEE, 1–6.
[123] Geancarlo Abich, Ricardo Reis, and Luciano Ost. 2021. The impact of precision bitwidth on the soft error reliability of the MobileNet network. In *IEEE 12th Latin America Symposium on Circuits and System (LASCAS'21)*. IEEE, 1–4.
[124] Geancarlo Abich, Jonas Gava, Rafael Garibotti, Ricardo Reis, and Luciano Ost. 2021. Applying lightweight soft error mitigation techniques to embedded mixed precision deep neural networks. *IEEE Trans. Circ. Syst. I: Reg. Pap.* 68, 11 (2021), 4772–4782.
[125] Geancarlo Abich, Rafael Garibotti, Jonas Gava, Ricardo Reis, and Luciano Ost. 2022. Impact of thread parallelism on the soft error reliability of convolution neural networks. In *IEEE 13th Latin America Symposium on Circuits and System (LASCAS'22)*. IEEE, 1–4.
[126] Geancarlo Abich, Rafael Garibotti, Ricardo Reis, and Luciano Ost. 2022. The impact of soft errors in memory units of edge devices executing convolutional neural networks. *IEEE Trans. Circ. Syst. II: Express Briefs* 69, 3 (2022), 679–683.
[127] Jonas Gava, Guilherme Dorneles, Ricardo Reis, Rafael Garibotti, and Luciano Ost. 2022. Soft error assessment of CNN inference models running on a RISC-V processor. In *29th IEEE International Conference on Electronics, Circuits and Systems (ICECS'22)*. IEEE, 1–4.
[128] Zhi Liu, Zhen Deng, and Xinni Yang. 2022. Using checksum to improve the reliability of embedded convolutional neural networks. *Microelectron. Reliab.* 136 (2022), 114666.
[129] Zhi Liu and Xinni Yang. 2022. An efficient structure to improve the reliability of deep neural networks on ARMs. *Microelectron. Reliab.* 136 (2022), 114729.
[130] Zhi Liu, Yuhong Liu, Zhengming Chen, Gang Guo, and Haibin Wang. 2021. Analyzing and increasing soft error resilience of Deep Neural Networks on ARM processors. *Microelectron. Reliab.* 124 (2021), 114331.
[131] Dimitris Agiakatsikas, Nikos Foutris, Aitzan Sari, Vasileios Vlagkoulis, Ioanna Souvatzoglou, Mihalis Psarakis, Mikel Luján, Maria Kastriotou, and Carlo Cazzaniga. 2021. Evaluation of the Xilinx deep learning processing unit under neutron irradiation. In *21st European Conference on Radiation and Its Effects on Components and Systems (RADECS'21)*. IEEE, 1–4.
[132] Giulio Gambardella, Nicholas J. Fraser, Ussama Zahid, Gianluca Furano, and Michaela Blott. 2022. Accelerated radiation test on quantized neural networks trained with fault aware training. In *IEEE Aerospace Conference (AERO'22)*. IEEE, 1–7.

[133] F. Libano, P. Rech, B. Neuman, J. Leavitt, M. Wirthlin, and J. Brunhaver. 2021. How reduced data precision and degree of parallelism impact the reliability of convolutional neural networks on FPGAs. *IEEE Trans. Nuclear Sci.* 68, 5 (2021), 865–872.

[134] Lucas Matana Luza, Daniel Söderström, Georgios Tsiligiannis, Helmut Puchner, Carlo Cazzaniga, Ernesto Sanchez, Alberto Bosio, and Luigi Dilillo. 2020. Investigating the impact of radiation-induced soft errors on the reliability of approximate computing systems. In *IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT'20)*. IEEE, 1–6.

[135] Pierre Maillard, Yanran P. Chen, Jason Vidmar, Nicholas Fraser, Giulio Gambardella, Minal Sawant, and Martin L. Voogel. 2022. Radiation tolerant deep learning processor unit (DPU) based platform using Xilinx 20nm Kintex UltraScale™ FPGA. *IEEE Trans. Nuclear Sci.* 70, 4 (2022).

[136] Pedro Martins Basso, Fernando Fernandes dos Santos, and Paolo Rech. 2020. Impact of tensor cores and mixed precision on the reliability of matrix multiplication in GPUs. *IEEE Trans. Nuclear Sci.* 67, 7 (2020), 1560–1565.

[137] Atieh Lotfi, Saurabh Hukerikar, Keshav Balasubramanian, Paul Racunas, Nirmal Saxena, Richard Bramley, and Yanxiang Huang. 2019. Resiliency of automotive object detection networks on GPU architectures. In *IEEE International Test Conference (ITC'19)*. IEEE, 1–9.

[138] Rubens Luiz Rech Junior, Sujit Malde, Carlo Cazzaniga, Maria Kastriotou, Manon Letiche, Christopher Frost, and Paolo Rech. 2022. High energy and thermal neutron sensitivity of Google Tensor Processing Units. *IEEE Trans. Nuclear Sci.* 69, 3 (2022), 567–575.

[139] Rubens Luiz Rech and Paolo Rech. 2022. Reliability of Google's tTensor Processing Units for embedded applications. In *Design, Automation & Test in Europe Conference & Exhibition (DATE'22)*. IEEE, 376–381.

[140] Alberto Bosio, Paolo Bernardi, Annachiara Ruospo, and Ernesto Sanchez. 2019. A reliability analysis of a deep neural network. In *IEEE Latin American Test Symposium (LATS'19)*. IEEE, 1–6.

[141] Seo-Seok Lee and Joon-Sung Yang. 2022. Value-aware parity insertion ECC for fault-tolerant deep neural network. In *Design, Automation & Test in Europe Conference & Exhibition (DATE'22)*. IEEE, 724–729.

[142] Annachiara Ruospo, Alberto Bosio, Alessandro Ianne, and Ernesto Sanchez. 2020. Evaluating convolutional neural networks reliability depending on their data representation. In *23rd Euromicro Conference on Digital System Design (DSD'20)*. IEEE, 672–679.

[143] Annachiara Ruospo, Ernesto Sanchez, Marcello Traiola, Ian O'connor, and Alberto Bosio. 2021. Investigating data representation for efficient and reliable convolutional neural networks. *Microprocess. Microsyst.* 86 (2021), 104318.

[144] Stephane Burel, Adrian Evans, and Lorena Anghel. 2022. Mozart+: Masking outputs with zeros for improved architectural robustness and testing of DNN accelerators. *IEEE Trans. Device Mater. Reliab.* 22, 2 (2022), 120–128.

[145] Le-Ha Hoang, Muhammad Abdullah Hanif, and Muhammad Shafique. 2021. TRe-Map: Towards reducing the overheads of fault-aware retraining of deep neural networks by merging fault maps. In *24th Euromicro Conference on Digital System Design (DSD'21)*. IEEE, 434–441.

[146] Thai-Hoang Nguyen, Muhammad Imran, Jaehyuk Choi, and Joon-Sung Yang. 2021. Low-cost and effective fault-tolerance enhancement techniques for emerging memories-based deep neural networks. In *58th ACM/IEEE Design Automation Conference (DAC'21)*. IEEE, 1075–1080.

[147] Ayesha Siddique, Kanad Basu, and Khaza Anuarul Hoque. 2021. Exploring fault-energy trade-offs in approximate DNN hardware accelerators. In *22nd International Symposium on Quality Electronic Design (ISQED'21)*. IEEE, 343–348.

[148] Yung-Yu Tsai and Jin-Fu Li. 2021. Evaluating the impact of fault-tolerance capability of deep neural networks caused by faults. In *IEEE 34th International System-on-Chip Conference (SOCC'21)*. IEEE, 272–277.

[149] Ussama Zahid, Giulio Gambardella, Nicholas J. Fraser, Michaela Blott, and Kees Vissers. 2020. FAT: Training neural networks for reliable inference under hardware faults. In *IEEE International Test Conference (ITC'20)*. IEEE, 1–10.

[150] Yingnan Zhao, Ke Wang, and Ahmed Louri. 2022. FSA: An efficient fault-tolerant systolic array-based DNN accelerator architecture. In *IEEE 40th International Conference on Computer Design (ICCD'22)*. IEEE, 545–552.

[151] Cheng Liu, Cheng Chu, Dawen Xu, Ying Wang, Qianlong Wang, Huawei Li, Xiaowei Li, and Kwang-Ting Cheng. 2021. HyCA: A hybrid computing architecture for fault-tolerant deep learning. *IEEE Trans. Comput.-aid. Des. Integ. Circ. Syst.* 41, 10 (2021), 3400–3413.

[152] Dawen Xu, Cheng Chu, Qianlong Wang, Cheng Liu, Ying Wang, Lei Zhang, Huaguo Liang, and Kwang-Ting Cheng. 2020. A hybrid computing architecture for fault-tolerant deep learning accelerators. In *IEEE 38th International Conference on Computer Design (ICCD'20)*. IEEE, 478–485.

[153] Jeff Jun Zhang, Kanad Basu, and Siddharth Garg. 2019. Fault-tolerant systolic array based accelerators for deep neural network execution. *IEEE Des. Test* 36, 5 (2019), 44–53.

[154] Jeff Jun Zhang, Tianyu Gu, Kanad Basu, and Siddharth Garg. 2018. Analyzing and mitigating the impact of permanent faults on a systolic array based neural network accelerator. In *IEEE 36th VLSI Test Symposium (VTS'18)*. IEEE, 1–6.

[155] Giulio Gambardella, Johannes Kappauf, Michaela Blott, Christoph Doehring, Martin Kumm, Peter Zipf, and Kees Vissers. 2019. Efficient error-tolerant quantized neural network accelerators. In *IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT'19)*. IEEE, 1–6.

[156] Behzad Salami, Osman S. Unsal, and Adrian Cristal Kestelman. 2018. On the resilience of RTL NN accelerators: Fault characterization and mitigation. In *30th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD'18)*. IEEE, 322–329.

[157] Josie E. Rodriguez Condia, Juan-David Guerrero-Balaguera, Fernando F. Dos Santos, Matteo Sonza Reorda, and Paolo Rech. 2022. A multi-level approach to evaluate the impact of GPU permanent faults on CNN's reliability. In *IEEE International Test Conference (ITC'22)*. IEEE, 278–287.

[158] Juan-David Guerrero-Balaguera, Josie E. Rodriguez Condia, and Matteo Sonza Reorda. 2022. Neural network's reliability to permanent faults: Analyzing the impact of performance optimizations in GPUs. In *29th IEEE International Conference on Electronics, Circuits and Systems (ICECS'22)*. IEEE, 1–4.

[159] Juan-David Guerrero-Balaguera, Robert Limas Sierra, and Matteo Sonza Reorda. 2022. Effective fault simulation of GPU's permanent faults for reliability estimation of CNNs. In *IEEE 28th International Symposium on On-Line Testing and Robust System Design (IOLTS'22)*. IEEE, 1–6.

[160] Juan-David Guerrero-Balaguera, Luigi Galasso, Robert Limas Sierra, Ernesto Sanchez, and Matteo Sonza Reorda. 2022. Evaluating the impact of permanent faults in a GPU running a deep neural network. In *IEEE International Test Conference in Asia (ITC-Asia'22)*. IEEE, 96–101.

[161] Zheyu Yan, Yiyu Shi, Wang Liao, Masanori Hashimoto, Xichuan Zhou, and Cheng Zhuo. 2020. When single event upset meets deep neural networks: Observations, explorations, and remedies. In *25th Asia and South Pacific Design Automation Conference (ASP-DAC'20)*. IEEE, 163–168.

[162] Dawen Xu, Ziyang Zhu, Cheng Liu, Ying Wang, Huawei Li, Lei Zhang, and Kwang-Ting Cheng. 2020. Persistent fault analysis of neural networks on FPGA-based acceleration system. In *IEEE 31st International Conference on Application-specific Systems, Architectures and Processors (ASAP'20)*. IEEE, 85–92.

[163] Dawen Xu, Ziyang Zhu, Cheng Liu, Ying Wang, Shuang Zhao, Lei Zhang, Huaguo Liang, Huawei Li, and Kwang-Ting Cheng. 2021. Reliability evaluation and analysis of FPGA-based neural network acceleration system. *IEEE Trans. Very Large Scale Integ. Syst.* 29, 3 (2021), 472–484.

[164] Abdulrahman Mahmoud, Neeraj Aggarwal, Alex Nobbe, Jose Rodrigo Sanchez Vicarte, Sarita V. Adve, Christopher W. Fletcher, Iuri Frosio, and Siva Kumar Sastry Hari. 2020. PyTorchFi: A runtime perturbation tool for DNNs. In *50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W'20)*. IEEE, 25–31.

[165] Zitao Chen, Niranjhana Narayanan, Bo Fang, Guanpeng Li, Karthik Pattabiraman, and Nathan DeBardeleben. 2020. TensorFI: A flexible fault injection framework for TensorFlow applications. In *IEEE 31st International Symposium on Software Reliability Engineering (ISSRE'20)*. IEEE, 426–435.

[166] Guanpeng Li, Karthik Pattabiraman, and Nathan DeBardeleben. 2018. TensorFI: A configurable fault injector for TensorFlow applications. In *IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW'18)*. IEEE, 313–320.

[167] Niranjhana Narayanan, Zitao Chen, Bo Fang, Guanpeng Li, Karthik Pattabiraman, and Nathan Debardeleben. 2022. Fault injection for TensorFlow applications. *IEEE Trans. Depend. Sec. Comput.* 20, 4 (2022).

[168] Sabuj Laskar, Md Hasanur Rahman, and Guanpeng Li. 2022. TensorFI+: A scalable fault injection framework for modern deep learning neural networks. In *IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW'22)*. IEEE, 246–251.

[169] Sabuj Laskar, Md Hasanur Rahman, Bohan Zhang, and Guanpeng Li. 2022. Characterizing deep learning neural network failures between algorithmic inaccuracy and transient hardware faults. In *IEEE 27th Pacific Rim International Symposium on Dependable Computing (PRDC'22)*. IEEE, 54–67.

[170] Brandon Reagen, Udit Gupta, Lillian Pentecost, Paul Whatmough, Sae Kyu Lee, Niamh Mulholland, David Brooks, and Gu-Yeon Wei. 2018. Ares: A framework for quantifying the resilience of deep neural networks. In *55th ACM/ESDA/IEEE Design Automation Conference (DAC'18)*. IEEE, 1–6.

[171] Zitao Chen, Guanpeng Li, Karthik Pattabiraman, and Nathan DeBardeleben. 2019. BinFI: An efficient fault injector for safety-critical machine learning systems. In *International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–23.

[172] Udit Kumar Agarwal, Abraham Chan, and Karthik Pattabiraman. 2022. LLTFI: Framework agnostic fault injection for machine learning applications (tools and artifact track). In *IEEE 33rd International Symposium on Software Reliability Engineering (ISSRE'22)*. IEEE, 286–296.

[173] Elvis Rojas, Diego Pérez, Jon C. Calhoun, Leonardo Bautista Gomez, Terry Jones, and Esteban Meneses. 2021. Understanding soft error sensitivity of deep learning models and frameworks through checkpoint alteration. In *IEEE International Conference on Cluster Computing (CLUSTER'21)*. IEEE, 492–503.

[174] N2D2 CAD framework for DNNs. 2016. Retrieved from: https://github.com/cea-list/N2D2

[175] Ananda Samajdar, Yuhao Zhu, Paul Whatmough, Matthew Mattina, and Tushar Krishna. 2018. SCALE-Sim: Systolic CNN accelerator simulator. *arXiv preprint arXiv:1811.02883* (2018).

[176] Yu-Hsin Chen, Joel Emer, and Vivienne Sze. 2016. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. *ACM SIGARCH Comput. Archit. News* 44, 3 (2016), 367–379.

[177] XILINX. 2021. SoCs with Hardware and Software Programmability. Retrieved from: https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html

[178] Yaman Umuroglu, Nicholas J. Fraser, Giulio Gambardella, Michaela Blott, Philip Leong, Magnus Jahre, and Kees Vissers. 2017. FINN: A framework for fast, scalable binarized neural network inference. In *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. 65–74.

[179] Juan-David Guerrero-Balaguera, Luigi Galasso, Robert Limas Sierra, and Matteo Sonza Reorda. 2022. Reliability assessment of neural networks in GPUs: A framework for permanent faults injections. In *IEEE 31st International Symposium on Industrial Electronics (ISIE'22)*. IEEE, 959–962.

[180] NVIDIA Corporation. 2021. NVIDIA TensorRT. Retrieved from: https://developer.nvidia.com/tensorrt

[181] Josie E. Rodriguez Condia, Boyang Du, Matteo Sonza Reorda, and Luca Sterpone. 2020. FlexGripPlus: An improved GPGPU model to support reliability analysis. *Microelectron. Reliab.* 109 (2020), 113660.

[182] Timothy Tsai, Siva Kumar Sastry Hari, Michael Sullivan, Oreste Villa, and Stephen W. Keckler. 2021. NVBitFI: Dynamic fault injection for GPUs. In *51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'21)*. IEEE, 284–291.

[183] Daniel Oliveira, Laércio Pilla, Nathan DeBardeleben, Sean Blanchard, Heather Quinn, Israel Koren, Philippe Navaux, and Paolo Rech. 2017. Experimental and analytical study of Xeon Phi reliability. In *International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–12.

[184] Cristiana Bolchini, Luca Cassano, Antonio Miele, and Alessandro Toschi. 2022. Fast and accurate error simulation for CNNs against soft errors. *IEEE Trans. Comput.* 72, 4 (2022).

[185] Siva Kumar Sastry Hari, Timothy Tsai, Mark Stephenson, Stephen W. Keckler, and Joel Emer. 2017. SASSIFI: An architecture-level fault injection tool for GPU application resilience evaluation. In *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS'17)*. IEEE, 249–258.

[186] Muhammad Abdullah Hanif and Muhammad Shafique. 2020. SalvageDNN: Salvaging deep neural network accelerators with permanent faults through saliency-driven fault-aware mapping. *Philosoph. Trans. Roy. Societ. A* 378, 2164 (2020), 20190164.

[187] Annachiara Ruospo and Ernesto Sanchez. 2021. On the reliability assessment of artificial neural networks running on AI-oriented MPSoCs. *Appl. Sci.* 11, 14 (2021), 6455.

[188] Annachiara Ruospo, Gabriele Gavarini, Ilaria Bragaglia, Marcello Traiola, Alberto Bosio, and Ernesto Sanchez. 2022. Selective hardening of critical neurons in deep neural networks. In *25th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS'22)*. IEEE, 136–141.

[189] Christoph Schorn, Andre Guntoro, and Gerd Ascheid. 2018. Accurate neuron resilience prediction for a flexible reliability management in neural network accelerators. In *Design, Automation & Test in Europe Conference & Exhibition (DATE'18)*. IEEE, 979–984.

[190] Christoph Schorn, Andre Guntoro, and Gerd Ascheid. 2019. An efficient bit-flip resilience optimization method for deep neural networks. In *Design, Automation & Test in Europe Conference & Exhibition (DATE'19)*. IEEE, 1507–1512.

[191] Wonseok Choi, Dongyeob Shin, Jongsun Park, and Swaroop Ghosh. 2019. Sensitivity based error resilient techniques for energy efficient deep neural network accelerators. In *56th Annual Design Automation Conference*. 1–6.

[192] Abdulrahman Mahmoud, Siva Kumar Sastry Hari, Christopher W. Fletcher, Sarita V. Adve, Charbel Sakr, Naresh Shanbhag, Pavlo Molchanov, Michael B. Sullivan, Timothy Tsai, and Stephen W. Keckler. 2020. HarDNN: Feature map vulnerability evaluation in CNNs. *arXiv preprint arXiv:2002.09786* (2020).

[193] Abdulrahman Mahmoud, Siva Kumar Sastry Hari, Christopher W. Fletcher, Sarita V. Adve, Charbel Sakr, Naresh R. Shanbhag, Pavlo Molchanov, Michael B. Sullivan, Timothy Tsai, and Stephen W. Keckler. 2021. Optimizing selective protection for CNN resilience. In *International Symposium on Software Reliability Engineering (ISSRE'21)*. 127–138.

[194] Muhammad Sabih, Frank Hannig, and Jürgen Teich. 2021. Fault-tolerant low-precision DNNs using explainable AI. In *51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W'21)*. IEEE, 166–174.

[195] Liqi Ping, Jingweijia Tan, and Kaige Yan. 2020. SERN: Modeling and analyzing the soft error reliability of convolutional neural networks. In *Great Lakes Symposium on VLSI*. 445–450.

[196] G. Gavarini, D. Stucchi, A. Ruospo, G. Boracchi, and E. Sanchez. 2022. Open-set recognition: An inexpensive strategy to increase DNN reliability. In *IEEE 28th International Symposium on On-Line Testing and Robust System Design (IOLTS'22)*. IEEE, 1–7.

[197] Yi He, Prasanna Balaprakash, and Yanjing Li. 2020. Fidelity: Efficient resilience analysis framework for deep learning accelerators. In *53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'20)*. IEEE, 270–281.

[198] NVIDIA Corporation. 2021. NVDLA Open Source Project. Retrieved from: http://nvdla.org
[199] Alessandro Veronesi, Francesco Dall'Occo, Davide Bertozzi, Michele Favalli, and Milos Krstic. 2022. Exploring software models for the resilience analysis of deep learning accelerators: The NVDLA case study. In *25th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS'22)*. IEEE, 142–147.
[200] Yangchao Zhang, Hiroaki Itsuji, Takumi Uezono, Tadanobu Toba, and Masanori Hashimoto. 2022. Estimating vulnerability of all model parameters in DNN with a small number of fault injections. In *Design, Automation & Test in Europe Conference & Exhibition (DATE'22)*. IEEE, 60–63.
[201] Abraham Chan, Arpan Gujarati, Karthik Pattabiraman, and Sathish Gopalakrishnan. 2022. The fault in our data stars: studying mitigation techniques against faulty training data in machine learning applications. In *52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'22)*. IEEE, 163–171.

Received 8 May 2023; accepted 15 December 2023