Microelectronics Reliability 115 (2020) 113969

Contents lists available at ScienceDirect

Microelectronics Reliability

journal homepage: www.elsevier.com/locate/microrel

Review paper

Soft errors in DNN accelerators: A comprehensive review

Younis Ibrahim [a], Haibin Wang [a,f,*], Junyang Liu [a], Jinghe Wei [b], Li Chen [c], Paolo Rech [d], Khalid Adam [e], Gang Guo [f]

[a] College of IoT Engineering, Hohai University, Changzhou, Jiangsu, China
[b] No.58 Research Institute, China Electronics Technology Group Co., Wuxi, Jiangsu, China
[c] University of Saskatchewan, 57 Campus Drive, Saskatoon, SK, Canada
[d] Institute of Informatics, University of UFRGS, Porto Alegre, Rio Grande do Sul, Brazil
[e] Faculty of Electrical and Electronics Engineering, University Malaysia Pahang (UMP), Malaysia
[f] China Institute of Atomic Energy, Beijing, China

ARTICLE INFO

Keywords:
Deep learning
DNN accelerators
GPU
FPGA
ASIC
Soft errors
Reliability

ABSTRACT

Deep learning tasks cover a broad range of domains and an even more extensive range of applications, from entertainment to extremely safety-critical fields. Thus, Deep Neural Network (DNN) algorithms are implemented on different systems, from small embedded devices to data centers. DNN accelerators have proven to be a key to efficiency, as they are even more efficient than CPUs. Therefore, they have become the major executing hardware for DNN algorithms. However, these accelerators are susceptible to several types of faults. Soft errors pose a particular threat because the high-level parallelism in these accelerators can propagate a single failure to multiple errors in the next levels until the model predictions' output is affected. This article presents a comprehensive review of the reliability of the DNN accelerators. The study begins by reviewing the widely assumed claim that DNNs are inherently tolerant to faults. Then, the available DNN accelerators are systematically classified into several categories. Each is individually analyzed; and the commonly used accelerators are compared in an attempt to answer the question, *which accelerator is more reliable against transient faults*? The concluding part of this review highlights the gray areas of the DNNs and predicts future research directions that will enhance its applicability. This study is expected to benefit researchers in the areas of deep learning, DNN accelerators, and reliability of this efficient paradigm.

1. Introduction

Deep Neural Networks (DNNs) have been progressing at a phenomenal pace and gaining unprecedented popularity in recent years. Following the latest report undertaken by International Data Corporation (IDC), spending on Artificial Intelligence (AI) systems alone was $24.0 billion in 2019 and expected to reach $97.9 billion in 2023, with the most considerable portion spent on AI hardware [1]. The report indicates the steady growth of this field. One reason is that DNNs have become the predominant approach for most of the previous machine learning tasks [2]. DNNs have been adopted in a variety of applications [3], such as computer vision [4], speech recognition [5], and natural language processing [6]. This technology has far outstripped human abilities in the area of vision [7–9]. DNN technology has attracted intense research interest in almost every field that involves advanced data analytics and intelligent control. Consequently, DNNs are currently widely used in complex mission-critical systems, such as autonomous vehicles [10] and healthcare [11]. Besides, other fields are vigorously seeking this efficient paradigm, such as NASA's space applications [12,13].

Artificial Neural Networks (ANNs) are considered to be tolerant to transient faults (i.e., soft errors). This is based on some observations commonly claimed to be facts. *First*, as ANNs contain a higher number of neurons (thousands to millions), more than required to accomplish a specific task [14], they can still perform their overall purpose even if some of the neurons are not working. *Second*, since ANNs mimic the human brain, which can tolerate some level of neuron faults or even use noise as a source of computation [15], ANNs are considered to have this intrinsic feature (fault tolerance) of biological systems [16]. *Third*, maximizing performance is the main issue that dominates the research work in the field of AI. [17]. *Fourth*, the learning capability of ANNs during and after the training process seems to indicate that ANNs have a

* Corresponding author.
E-mail address: wanghaibin@hhuc.edu.cn (H. Wang).

https://doi.org/10.1016/j.microrel.2020.113969
Received 11 November 2019; Received in revised form 22 September 2020; Accepted 12 October 2020
Available online 28 October 2020

high degree of fault tolerance to handle incomplete data and noise. This claim derived from the fact that learning allows ANNs to find solutions to problems they were not trained for [18].

However, it is not easy and straightforward to extend these features to today's DNN systems for the following reasons:

A. The modern DNN models have much more complex structures (i.e., more layers and operations) and different topologies, such as Convolutional Neural Networks (CNNs), compared to traditional counterparts.
B. Safety-critical systems, which continue to bring new safety standards, such as ISO 26262, have not been considered in previous reliability studies [19,29].
C. DNN accelerators, which have become the leading executing hardware of DNN models, have not been considered in early studies [20].
D. Edge devices of the Internet of Things (IoT) are bringing many constraints, such as storage and power issues, to deep learning [21,22].

Therefore, the prior assessments of fault tolerance across different ANNs do not provide much insight into the robustness of modern DNN systems. Hence, it can be concluded that the most remarkable research direction in the field of neural networks is the optimization of algorithms and network architectures. In contrast, their reliability is rarely investigated. Consequently, it is essential to review and understand newer DNN systems' behavior, considering the abovementioned points.

Due to the massive computational power requirements and high memory accesses that DNNs manifest, a Central Processing Unit (CPU) alone is no longer the best choice for executing DNN algorithms. Accordingly, customized accelerators (i.e., co-processors) have emerged to overcome the computational bottleneck of CPUs. Nowadays, DNN models are executed on heterogeneous systems, but accelerators handle most of the computations [20,23]. Consequently, a large number of accelerators have been proposed in the last few years to accelerate the implementation of DNN algorithms, including Graphics Processing Units (GPUs), Field-Programmable Gate Arrays (FPGAs), and Application Specific Integrated Circuits (ASICs). However, with the continuous downscaling of the fabrication process to smaller dimensions for denser integration, these accelerators have become even more susceptible to faults [22,40].

Many studies have been conducted on the performance and power issues of DNN accelerators and their applications [24–28]. However, relatively few works studied DNNs' reliability issues through their accelerators. According to the ISO 26262 standard [29], automotive applications' accelerators must be compatible with Automotive Safety Integrity Level (ASIL). Especially in ASIL-D, the failure rate must be less than 10 Failures In Time (< 10 FIT) and Single-point Fault Metric (SPFM) at least 99%. SPFM is one of the metrics used to detect faults and FIT, which is the effectiveness of failure detection.

As experimentally shown by many studies, one of the major sources of unreliability in modern systems is soft errors [30,31]. They are typically caused by several uncontrollable sources, such as high-energy particles strike and temperature [32]. These errors can impact the behavior of DNN accelerators. Correspondingly, the reliability of DNN models executed on top of these accelerators would be affected as well, producing wrong predictions. Fig. 1 shows the general overview of how faults propagate from the hardware (i.e., DNN accelerator) level to the software (i.e., DNN model) level. Transient faults can occur in hardware components, such as memory and processing/control elements. They eventually lead to wrong predictions, such as misclassification and misdetection of objects in DNNs. Thus, the consequences could be disastrous. For instance, in [20], the study has experimentally shown that a truck misclassified as a bird under a soft-errors effect. Such results can have practical implications for safety-critical applications. If such a scenario is encountered in self-driving cars, it would be a tragedy. Therefore, it is a severe issue that should be addressed for each DNN accelerator.

A few prior studies have examined DNN accelerators, targeting DNN models. Unfortunately, the wide range of fabrication technologies and proposed architectures of these accelerators made it a considerable gap in their reliability. Furthermore, there are a large number of design companies and startups dedicated to providing AI chips. These points

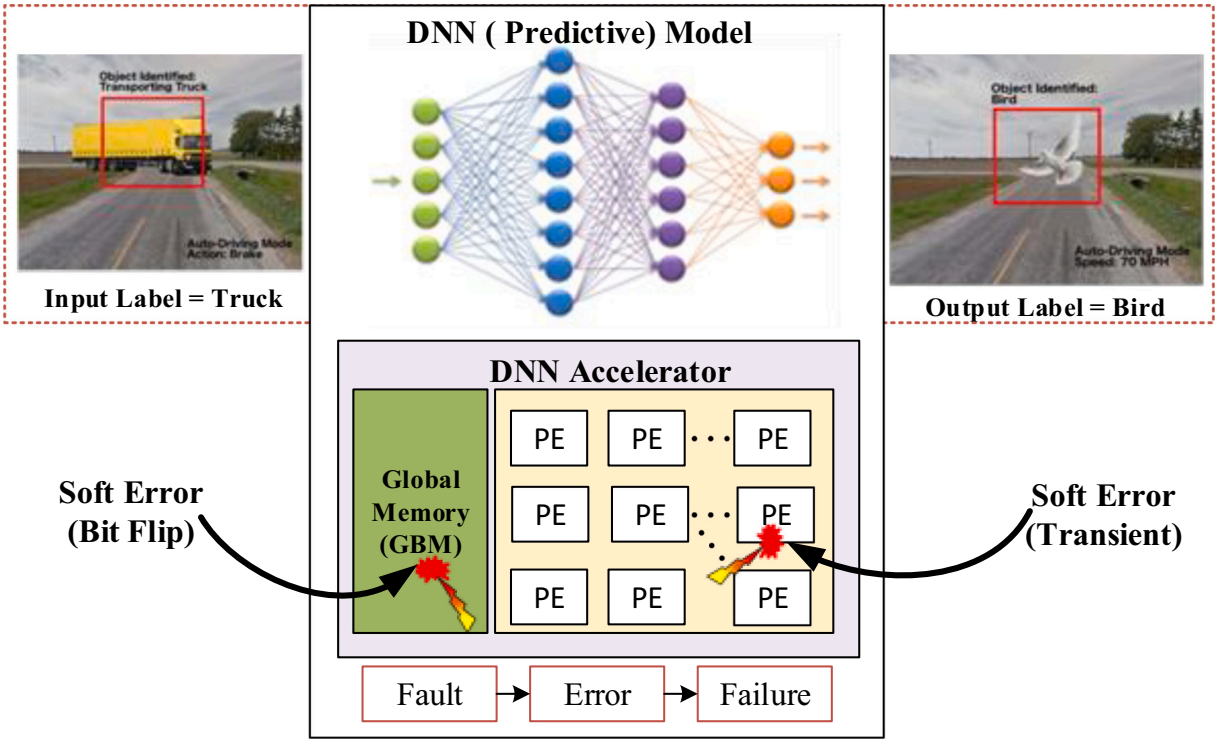

Fig. 1. Errors propagation from a DNN accelerator to the output of a neural network model [20,22,44].

have controlled the main effort to deliver DNN accelerators with low-power consumption and high performance. As a result, the reliability issue is often overlooked, which encouraged us to call attention to the missing research gap.

Therefore, this article aims to review the AI accelerators in terms of their reliability regarding soft errors. The main contributions of this paper are as follows:

- Presenting a thorough review of the available DNN accelerators, oriented towards reliability issues.
- Classifying DNN accelerators into categories based on several criteria for an in-depth analysis purpose.
- Presenting a comprehensive analysis for each category individually and comparing them.
- Highlighting the future challenges of DNN accelerators for further research to enhance the DNNs' resilience.

The rest of this paper is organized as follows: Section 2 provides a brief overview of deep learning, DNN accelerators, and reliability threats; alongside the related works. Section 3 provides a taxonomy of the DNN accelerators. Section 4 introduces the methodology followed for this review. Section 5 demonstrates the impact of soft errors in DNN accelerators in detail by analyzing each category separately. Section 6 presents further discussion related to the analysis. Section 7 calls attention to the future challenges of the area and directions for further research. Finally, Section 8 concludes our work.

2. Background and motivation

This section briefly provides the relevant background on DNN algorithms (Section 2.1), DNN accelerators (Section 2.2), and soft errors (Section 2.3). It also presents related works by discussing why the existing reviews provide limited insights into DNN accelerators' reliability and how this review can go beyond the existing ones (Section 2.4).

2.1. Deep neural networks

An artificial neural network is a subfield of machine learning that aims to mimic the human brain in analyzing and processing information to solves problems [26]. ANNs consist of inputs and outputs, which are organized into layers. Three types of layers construct an ANN, an input layer, a hidden layer(s), and an output layer [24]. Each layer is made up of smaller units called neurons, which are the fundamental computational units of the network for performing a task [28].

A deep neural network or deep learning is essentially an ANN with many hidden layers, where the term "deep" refers to extra layers [27,33]. Thus, a DNN is a network composed of multiple-computational layers. This involves numerous simple computations on a weighted sum of the input values [34].

Based on the internal structure of the network, there are several architectures for DNNs, including Multi-layer Perceptron (MLP) and Convolutional Neural Networks (CNNs) [37]. These two types are considered in this review because they are the basis of the DNNs and the most commonly used types [35,36]. These two architectures are widespread in all domains, and they have received most of the attention, both in research and industry [6,18,21,35]. Consequently, MLP and CNN are currently the backbones of deep learning. Therefore, these two popular DNN architectures are briefly described.

Whether it is MLP or CNN, a DNN should have inputs, neurons, layers, activation functions, multiply-sum operations, loss functions, parameters, and a specific topology for being a network. Indeed, DNNs are models created with linear algebra at their cores, and then later optimized with calculus (i.e., learning process). As a result, DNNs are fundamentally a chain of matrix operations applied to input data and a set of parameters required to map the output to the input [37]. Therefore, the basic operation in DNNs (e.g., CNN and MLP) is a series of matrix multiplications.

As conceptually shown in Fig. 2, each neuron receives some inputs and performs a dot-product or convolutional operation between the input and its weights (i.e., multiply-and-sum). Then, it adds the biases to the intermediate output to obtain the activation (i.e., g) before passing it to an activation function (i.e., f(g)) for non-linearity, which eventually gives the final output of this neuron. Therefore, a compulsory operation in any DNN is a multiply-accumulate (MAC) operation, suitable for all kinds of matrix operations, such as dot product and convolution (for filtering). Although it needs a large amount of data to be performed, MAC is the primary and most important operation in DNNs.

The difference between MLPs and CNNs is how neurons of a layer connect to the next layer's neurons. MLPs consist of fully-connected (FC) layers, where each neuron in this layer is connected with every other neuron of the previous layer [8]. By contrast, CNNs have convolutional layers, where blocks of neurons (i.e., 2-D filters) are taken as local connectivity; filters are slid across the input (image) space repeatedly applied over each patch of the input [17]. Convolutional layers are sparsely/partially connected rather than fully connected. In MLPs, FC computations are MAC operations, whereas, in CNNs, convolutional computations are MAC operations [38].

CNNs are computationally more intensive than MLPs because they take a 3-D tensor as an input to understand spatial relation between pixels of images/videos. Whereas MLPs take a 1-D vector as an input and transform it through a series of hidden layers [39]. From another angle, memory requirements for CNNs less than for MLPs. That is due to the higher number of parameters in MLPs that come from all connected neurons. For a detailed MLP and CNN description, readers are referred to [8,17].

The main feature of DNNs is the learning capability [40]. For supervised learning, both MLP and CNN are deployed in two steps. ***Training step***, after a DNN topology is built, the model is fed with input data and random parameters. With every learning step, parameters (i.e., weights) are learned for the designed model to solve a specific task [41]. Learning means tuning the parameters to determine their appropriate values. ***Inference step***, this trained model is used for predictions with new data.

The learning process requires many computations and enough memory to hold the input, intermediate, and output values [18]. In turn, this requires a high level of parallelism, which is usually achieved using some AI accelerators. Therefore, with the emergence of deep learning, hardware accelerators' importance has significantly increased [42].

If an error occurred due to a fault in any part of the neuron's body (see Fig. 2), the output value might be affected. From a hardware perspective, a change in neuron's output due to a change in intermediate components depends on the hardware implementing this neuron, which is a DNN accelerator in our case. This concept is applied to the entire layer and model.

2.2. DNN accelerators

This section provides an overview of the different hardware

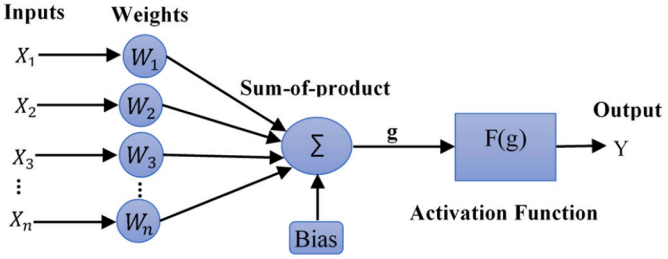

Fig. 2. Single neuron in a DNN and its main components [8].

platforms used for accelerating neural networks in the past decades, thus emphasizing DNN accelerators.

AI's full value will only be achieved when the proper chipsets are enabled to provide the necessary computational and storage capacities. Chipsets take a wide variety of forms, from the traditional general-purpose CPUs to extremely customized accelerators [43].

It is worth mentioning that CPUs are a vital part of the execution pipeline of DNN algorithms. They are responsible for the data pre-processing, controlling the execution tasks, and synchronizing the tasks for any accelerator attached to it. Furthermore, CPUs provide the highest degree of flexibility and ease of use. Notwithstanding all these features, CPUs perform inefficiently compared to accelerators in parallel computations, such as DNNS [44–46]. Since the main topic of this study is DNN accelerators, CPUs are excluded from this review.

The term "accelerator" refers to any silicon chip that works in tandem with a CPU and has massive computing ability. A DNN accelerator is essentially a programmable architecture designed for compute-intensive applications, including deep learning algorithms. It provides even higher performance than a multi-core CPU [47]. As described in Section 2.1, DNNs perform small and simple functions on a large scale at super speeds. This means, by design, DNN algorithms require a massive amount of data and compute capacity. Therefore, DNN accelerators are well suited to fulfill these requirements.

A DNN accelerator is composed of global memory and an array of processing elements (PEs). The overall architecture of DNN accelerators is shown in Fig. 3. It is noteworthy that computational units within each PE can be either Arithmetic logic units (ALUs) or Floating-point Units (FPUs). However, for simplicity purposes, the term ALU will be used throughout the paper. ALUs of each PE consist of a multiplier and an adder as execution units for performing MAC operations [46]. The ALU is where the vast majority of computations occur [20]. The computational complexity of DNN algorithms matches the parallelized structure of these accelerators. Thus, each PE computes the MAC operations that are described in Section 2.1.

2.3. Soft errors

In electronic devices, the reliability of a system/device is the ability to persist working even if faults are there. It refers to the resilience of that device against vulnerabilities to faults in the electronic components. This definition agrees with those used in other works [20,48,49]. DNN accelerators are vulnerable to many uncontrollable sources of faults, including:

- Radiation-induced errors: Are errors caused by ionizing radiation, which can permanently damage a device or lead to system failure [50–53].
- Aging: Degradation in circuit characteristics, such as delay over time [48,54].
- Process Variations: variations in the attributes of transistors in the chip fabrication process [54,55].
- Temperature: Thermal issues impact reliability by increasing the Soft Error Rate, as well as the rate of aging of a circuit [48], [56–58].

The scope of this review is limited to soft errors, regardless of the source of the faults. These errors cannot permanently damage the device but can modify a value stored in a memory element or change a signal status [48,50,51]. Soft errors have become one of the most crucial design concerns in modern electronics due to aggressive technology scaling, high frequency, and low design costs [59]. They may be the worst type of faults [31,53,60]. These non-destructive events are able to provoke an accelerator to produce wrong computation results or incorrect predictions in DNNs, leading to catastrophic consequences [61]. In mission-critical systems on earth and all applications in space, soft errors are already a major issue [51].

Unlike traditional electronic devices, DNN accelerators are co-processors built with cutting edge technologies and numerous parallel structures. They can be extremely prone to corruption by soft errors [62]. Furthermore, accelerators are the life force of all complex computer systems nowadays, such as DNN models [63].

There are several ways that transient faults can disrupt an accelerator's operations. However, two reasons making transient faults in DNN accelerators deserve more attention: (1) the complex memory hierarchy within these accelerators that are used for improved latency [57,64]; and (2) the massively parallel structure of DNN accelerators, which tends to spread the single fault to multiple faults [64–66].

2.4. Related prior work

In the literature, the impact of transient faults in DNN systems (i.e., accelerators and algorithms) has been reviewed by a few researchers. The most relevant existing works are included, and the differences are highlighted.

Zhang *et al.* reviewed the error resilience of DNNs as a part of many challenges in the domain of robust systems for deep learning applications [67]. However, this study designated a significant portion of the work to review the neural networks' security and adversarial attacks. Reliability issues caused by soft errors were highlighted in a very general manner. Although DNN accelerators are the leading platforms for DNN algorithms nowadays, this study only considered ASIC accelerators (i.e., TPU). Other accelerators such as GPUs and FPGAs have not been covered in this paper.

Torres-Huitzil *et al.* reviewed the fault tolerance in artificial neural networks [22]. This work's two central points are:

First, they classified literature works based on fault models and metrics that have been used for fault tolerance evaluation. Since the fault model emulates the physical fault mechanisms in electronic devices, it determines some critical factors, including: (1) Physical components in a system to be evaluated; (2) types of faults that should be considered; and (3) the way components react to these faults in neural network systems.

FP: Floating Point, FxP: Fixed Point, SIMD: Single Instruction

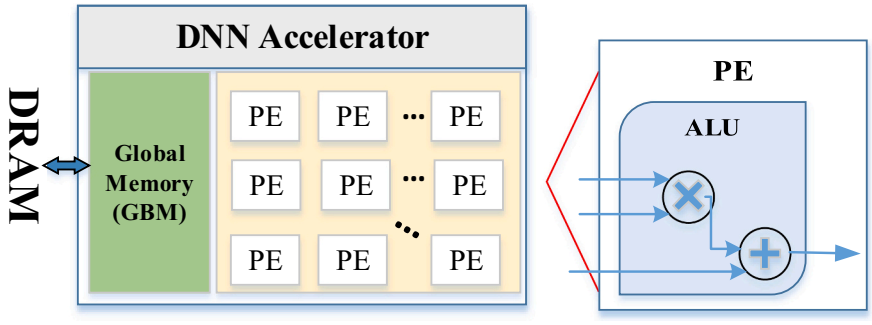

Fig. 3. The underlying architecture of DNN accelerators and its main components [44].

Multiple Data, VLIW: Very Long Instruction Word.

Second, the authors also classified the fault tolerance in neural networks as passive and active. The taxonomy was made based on the mechanisms that each class utilizes to achieve fault tolerance. In particular, the methods and techniques used for each category. Finally, representative studies for improving fault tolerance have been reviewed for each category.

However, two main aspects were not considered in this work. (1) Various DNN architectures exhibit diverse behaviors concerning soft errors. For instance, MLPs and CNNs have entirely different structures and workflows. As the authors stated in their future research directions, this aspect has not been considered. (2) The reliability of some key DNN accelerators such as FPGA and ASIC has not been covered. These two points will be our research questions.

Very recent and comprehensive work is presented by Mittal *et al.* [68]. Authors have reviewed the reliability of DNN algorithms and accelerators in an extensive body of work, addressing several design decisions on the reliability for both software and hardware levels. In particular, they emphasize the significance of design for reliability. Based on various ideas they summarized from literature works, the authors reviewed error resilience characteristics of DNN by three factors: layers, data-structures, and datatypes. They discussed different metrics and strategies used by different works. Importantly, they also covered some reliability studies on GPUs and FPGAs. They also reviewed error mitigation techniques for DNNs, discussing them from different levels (e.g., the algorithm and architecture levels). Also, they provided representative works for each level.

However, DNN accelerators are the main concern when it comes to soft error issues. These accelerators are produced in various fabrication technologies, including GPUs, FPGAs, and ASICs. In consequence, diversity in hardware resources, architectures, and characteristics will lead to different responses and behaviors to transient faults. To the best of our knowledge, none of the existing review works consider DNN accelerators with such diversity. Further, none of these reviews provide a reliability analysis of DNN models through their accelerators under the effect of transient faults. Therefore, it is worth assigning a review work specifically for the reliability of DNN accelerators, considering the diversity among them, and exclusively for soft errors.

From the observations mentioned above, the motivation to fill this research gap has risen. The main focus of this work will be on reviewing the error resilience of DNN models through their accelerators. In turn, this motivated us to classify the existing DNN accelerators into several categories. As each category has a unique internal structure, resources, and characteristics, it is worth addressing each separately, and that is by organizing literature works of each in a group. In this way, the review can shed light on each category by analyzing its reliability individually and drawing attention. Also, the study compares the commonly used DNN accelerators from a reliability perspective.

3. DNN accelerator taxonomy

This section arranges the existing DNN reliability studies into groups to better organize the review for the reader. Based on the IC technology in which the DNN accelerator is fabricated, this section classifies DNN accelerators into three categories: GPUs (Section 3.1), ASICs (Section 3.2), and FPGAs (Section 3.3). Moreover, a fourth category, *Others* (Section 3.4), which depends on different criteria, is added. Finally, Section 3.5 summarizes this section.

3.1. GPUs

GPUs are currently the most prominent [69] and dominant DNN accelerators [23,69]. This is due to GPUs' computational power and their massively parallel architecture, which can easily meet the requirements of DNNs. GPUs have been used to accelerate AI algorithms for more than a decade for both training and inference phases [50]. Several GPU manufacturers, such as NVIDIA, Intel, and AMD, are involved in this innovation.

The basic training systems today are adopting the off-line training approach on GPUs [70]. For example, NVIDIA's Tesla V100 is specially designed for deep learning training, powering the world's most powerful computers, as shown by the latest statistics in [69]. However, NVIDIA's recent Xavier GPUs are dedicated to embedded products for empowering systems such as autonomous vehicles, which need to detect objects in the real time, and usually used for inference.

The internal structure of GPUs is hugely complex. Streaming Multiprocessors (SMs) are the fundamental idea of the parallelism in GPUs. Each SM may have hundreds to thousands of cores, which are the fundamental processing units. Also, the GPUs' memory hierarchy is highly parallelized and shared among various resources [71]. However, this degree of parallelism in GPUs makes them extra vulnerable to faults.

GPUs have their limitations, such as high-power consumption, and that is where innovation in other accelerators is emerging. Due to the wide adoption of edge devices that provide mobility features produced by IoT, the power consumption and response time are two substantial reasons that make GPUs unsuitable accelerators in some scenarios [72–76].

3.2. ASICs

As the name suggests, an ASIC is a processor explicitly customized for one task, and this task cannot be changed over time. The purpose of using ASICs to accelerate AI algorithms is to solve the power constraints imposed by GPUs [32,77]. As the key bottleneck in DNN computations is memory accesses, ASIC accelerators utilize a data reuse pattern to reduce off-chip memory access, making them superfast [78]. Therefore, these accelerators provide performance closer to GPUs, but with higher energy efficiency [79]. Thus, this category occupies an integral part of today's DNN accelerators.

Examples of ASIC accelerators include Google's TPU [80], which is currently used for training and inference in Google's cloud platform and data centers [81]. Other examples include the DianNao series [79,82], which is aimed at broader support for AI algorithms mainly in inference and some training phase, and Cambricon-X from the Chinese giant, Cambricon [83], Eyeriss [44], and MAERI [84]. This category's general class (i.e., TPU) uses a systolic array with a 256×256 grid of MAC units as its core and a large on-chip memory [85,86]. However, with such density per chip in advanced technologies, soft error rates will escalate as well.

3.3. FPGAs

The characteristics of reconfigurability, versatility, and low-power consumption allow FPGAs accelerators to become a wise choice to accelerate DNN algorithms in particular scenarios. Examples of FPGAs utilized as DNN accelerators can be found in [45,46,77], [87–89], [90–92]. Since IoT and mobile devices fabricated with FPGA technology are known with energy efficiency, this class of accelerators is an excellent choice for inference purposes of the deep learning [93]. Moreover, FPGAs recently have become popular even for data centers (e.g., Microsoft Brainwave) for their outstanding inference capabilities [23]. Nevertheless, to some extent, they are used for training purposes as well [92].

FPGAs use arrays of logic, Look-Up Tables (LUTs), SRAM or Block RAM (BRAM), and specialized analog blocks to execute deep learning algorithms, such as CNNs and MLPs. There are two methods of implementing MAC operations in FPGA: by using Configurable Logic Block (CLB) resources of the FPGA (LUTs, FFs, and CARRY logics) or by Digital Signal Processing (DSP) slices [93]. However, both of these units are susceptible to soft errors, which could eventually lead to a failure in the DNN model's output. Thus, the reliability problem of DNNs through FPGAs is undeniable.

3.4. Others

It is worth noting that DNN accelerators can also be developed in the form of Digital Signal Processors (DSPs) [94–97]. However, it is not surprising to find DSPs cooperating with CPUs, FPGAs, and GPUs in heterogeneous systems [98,99]. Therefore, boundaries between a DSP used as a standalone DNN accelerator and that used as a part of another accelerator is not entirely clear. Consequently, this uncertainty has caused a significant gap in the reliability of DSPs as DNN accelerators. Consequently, DSPs cannot be classified as independent accelerators in our reliability review.

Instead, studies that provide solutions applicable to more than one accelerator are presented in this category. Specifically, studies address the reliability issues in specific components without considering the other parts of the accelerator. For instance, studies [40,100,101] proposed reliability solutions only for the accelerators' memory elements. More importantly, these solutions are not exclusive to a particular category (i.e., DNN accelerator). Instead, they are partial solutions that can be applied to various DNN accelerators in different categories. Thus, this will be the focus of the "*Others*" category in Sec 5.4.

3.5. Summary

DNN accelerators come in different architectures with different capabilities to fulfill the computational demand and memory bandwidth of the DNNs. The existing DNN accelerators have been classified based on the fabrication technologies, from general-purpose accelerators (i.e., GPUs) to programmable accelerators (i.e., FPGAs) to extremely specialized accelerators (i.e., ASICs). A comparison of some significant factors among these categories is shown in Table 1. Furthermore, major manufacturers of each category are listed in Table 2. Based on the reasons explained in Section 3.4, the reliability analysis of DSPs is not covered in this paper. Therefore, in the remainder of this paper, the three explicit categories are reviewed. Accelerator-independent works, which do not explicitly refer to a specific category, are reviewed in the fourth "*Others*" category.

4. Review methodology

This section describes the structure of this review. The overall framework and the scope of this study are demonstrated in Fig. 4. As this review considers the reliability from two views, DNN models and DNN accelerators, it is worth defining necessary components/elements of both, which must be considered for reliability evaluation.

4.1. DNN components

In order to evaluate each category's reliability, at least one of the following DNN components/metrics should be considered:

***Activations and parameters*:** Data required by a DNN model to accomplish its task. Activations refer to input, intermediate output (feature map), and output signals. Parameters refer to weights (i.e., connections) and biases of a model.

Table 1
Comparison of the main parameters of DNN accelerators.

Parameter	GPU	ASIC	FPGA	Others (DSP)
Performance per watt	Low	High	Very High	Very High
Latency	High	Very Low	Low	Medium
Complexity	High	Very High	Very low	Very High
Architecture design	SIMD	Systolic	SIMD & Systolic	VLIW & SIMD
FP/FxP support	More on FP	More on FxP	FxP & FP	More on FxP

Table 2
Key manufacturers of DNN accelerators [23].

Accelerator	Key manufacturers
GPU	NVIDIA, AMD, ARM, Qualcomm
FPGA	Xilinx, Intel
ASIC	Google, Cambricon
Others (DSP)	Qualcomm, ARM, CEVA, Analog Devices, NXP

Activation functions: This introduces nonlinearity to the computations of DNN networks. An activation function follows each neuron/layer in the network. Outputs from the activation functions are referred to as activations. Examples are ReLU, Sigmoid, and Tanh. They have a high ability to mask propagated errors in the middle.

Layers: Computations of a DNN are performed depending on the type of the layer. A layer performs MAC operations on a collection of neurons at once at a specific depth within a neural network. Layers are repetitive operations and separated by nonlinear (activation) functions. In each DNN, there are one input layer, one output layer, and hidden layers in between. Some layers can mask most of the errors and stop them from reaching the output layer, while others do not.

DNN phases: The deployment of DNNs is performed in two distinct phases: (1) *The training phase* is the process of creating a tuned DNN model. This involves the use of a dataset and parameters adjusting (learning). (2) *The inference phase* uses a trained model for giving predictions with new (unseen) data. Thus, reliability evaluation can be in either phase.

Data formats: Any data of a DNN (i.e., Activations and parameters) represented using either fixed-point (FxP) or floating-point (FP) formats for storage in memory elements. In FP, there are half-precision (FP16), single-precision (FP32), and double-precision (FP64) formats. The way value is stored in a memory element makes a difference. Thus, a fault that changes the sign bit, exponent bits, or mantissa bits will leave different impacts.

Considered dataset: As a foundation in DNNs, to build a model with high accuracy, it should be trained on more input data samples [17]. Therefore, the size of a dataset is critical in the model's performance. Reliability factors in a dataset include the size of the dataset, noisy features, label errors, omitted values, and the suitability of a dataset for a specific problem. Benchmark datasets include ImageNet, MNIST, CIFAR-10, and Pascal VOC [17].

Purpose of the DNN: The ultimate goal for which the DNN model has been optimized. (1) Image classification: the capability of a model to recognize the class in which an image belongs to; (2) Object detection: model's ability to locate objects in an image/scene and to identify each object. For both types, Accuracy, Precision, and Recall are the primary metrics in which the DNN models have been optimized.

DNN architecture: This refers to the arrangement of DNN components in a network, such as neurons into layers and connection patterns between layers. Various architectures (i.e., MLP vs. CNN) perform DNN operations differently and are composed of different types of layers. Therefore, the DNN structure (architecture) can affect the reliability assessment.

4.2. Accelerator elements

In order to evaluate each category's reliability, at least one of the accelerator's elements/factors should be considered:

Memory element: This indicates where data of a DNN is temporarily stored. It includes DRAM (main memory), SRAM (on-chip memory), and Registers. The impact of a fault in the value/address of that element is analyzed.

Processing/Control element: It performs MAC operations and other logical calculations of the DNN. It also coordinates the operations/instructions of the accelerator. DNN accelerators have multiple parallel processing/control elements, including Arithmetic Logic Units (ALU),

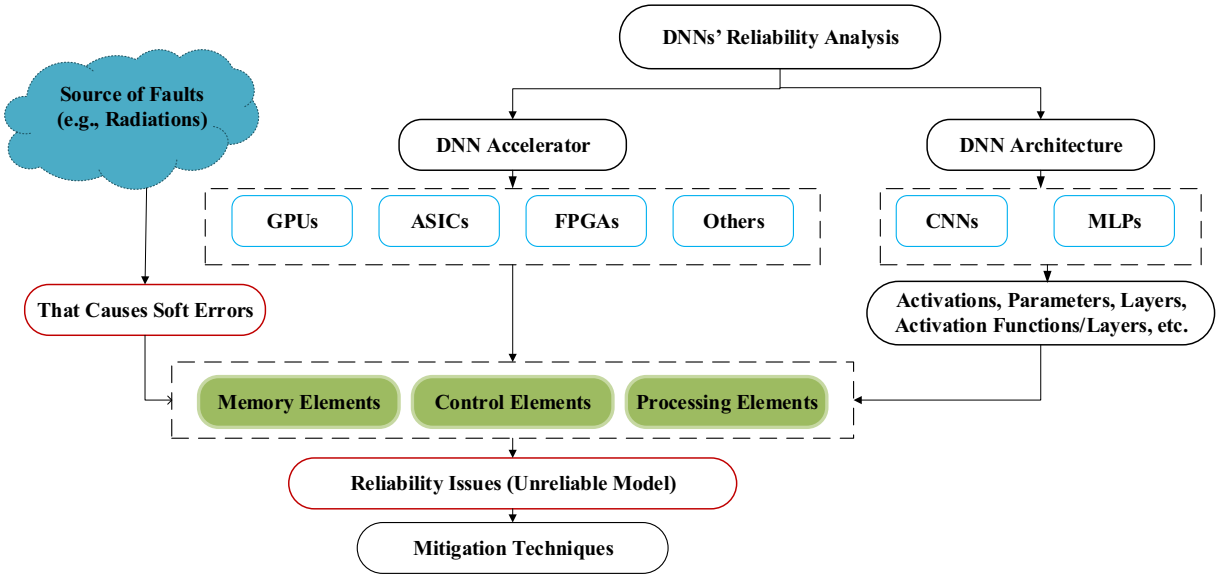

Fig. 4. General framework of our study and its scope.

Floating-point Units (FPU), Load-store Units (LSU), flip-flops, logic gates, fetch and decode unit, Schedulers, and configuration bits of SRAM-based FPGAs.

Evaluation approaches: There are two main strategies to evaluate the vulnerability of DNN accelerators: hardware fault injection (i.e., beam experiments) and software fault injection (i.e., emulating tools). Factors that can impact the evaluation include fault-injection (FI) tools used by a particular study, the level at which faults are injected, and FI rates (high vs. low).

Fault/Error model assumed: It is one of the most influential factors in reliability evaluation. Because it determines the direction and scope of the evaluation in a specific study, it answers questions such as (1) what components of an accelerator will be considered and how they will misbehave; (2) what types of errors will be considered (e.g., SEUs or SETs); and (3) how is the error criticality defined in a particular study?

Error propagation behaviors: Once an error occurs in one of the earlier layers, it may propagate through the various upcoming layers until it reaches the model's output. The error propagation process behaves differently based on each category's underlying hardware structure. Thus, errors propagate distinctively among various DNN accelerators.

***Mitigation techniques*:** They are the way to protect DNN systems against transient faults. Different mitigation approaches are needed for each DNN accelerator category to protect its components from soft errors.

5. Soft errors in DNN accelerators

This section reviews and analyzes works on the reliability of ANNs. More specifically, it focuses on the studies that address DNN reliability through DNN accelerators, from a soft error perspective.

Many previous works [21,35,47] have claimed that ANNs are inherently tolerant to faults. As a result, one possible approach for handling soft errors in DNN accelerators is to ignore them, hoping that the DNN model itself is intrinsically tolerant. The reasons behind these beliefs have been demonstrated in Section 1. However, at the hardware (accelerator) level, errors such as Single Event Upsets (SEUs) and Single Event Transients (SETs) can occur in the executing hardware. Consequently, they can have effects capable of undermining the inherent fault tolerance in DNNs, as demonstrated in the findings of [19,20,102].

Accelerators are more prone to transient faults than other electronic devices because the parallelism they contain tends to propagate a single fault to multiple output elements [19]. Several studies have examined the reliability-related issues, and our main goal in this section is to analyze the reliability of the DNN accelerators existing today. To achieve this, a summary of some representative works, which fall under each category, will be presented.

5.1. GPU

Much effort has been made by researchers [32,52,62,66], [103–105] to improve the reliability of GPUs. However, while previous works have paid full attention to the reliability of High Computing Performance (HPC) applications in GPUs, DNN models' reliability is left unstudied. DNN algorithms have unique computational characteristics distinct from HPC applications [23]. For example, unstructured data for the training stage and enormous memory to handle activations and learnable weights. Furthermore, Since ANN algorithms predict specific percentages of accuracy, DNNs themselves are not entirely accurate [43]. For these reasons, prior reliability studies cannot be expanded to include advanced DNN cases. Therefore, the focus will be on studies evaluating GPUS's resilience during the implementation of DNN models. For each study, some of the elements for DNN models/ accelerators (as defined in Section 4.1 and 4.2) will be considered. Thus, Tables 3 and 4 list the representative works of this category as well as the components and metrics considered in each work.

Santos *et al.* evaluated the well-known real-time object detection system's reliability, You Only Look Once (YOLO) [106]. As this system is implemented on Darknet's open-source framework, the authors intended to exploit the intensive use of the General Matrix to Matrix

Table 3
DNN component whose reliability is analyzed on GPU accelerators.

Component/Factor	Ref.
Activations & parameters	[110,112]
Layers	[19,102,106,108,111]
Data formats	[112]
Considered dataset	ImageNet [19,102,108,111], Caltech [106,112], PASCAL VOC [19,106,109]
DNN Architecture	CNN [almost all], MLP [19]
Purpose of the DNN	Classification [19, 102, 108, 110, 111], Object Detection [19,106,109,112]
DNN phase	Inference [all], training [none]

Table 4
GPU accelerator's element whose reliability is analyzed.

Element/Factor	Ref.
Memory element	[102,108–112]
Processing/control element	[19,102,108,111,112]
Evaluation approaches	SW [all], Beam [19,106,109,112]

Multiplication (GEMM) by this framework and take it as the primary metric to evaluate the resilience of it. The analysis considered three different GPU architectures. The notable finding of this study is that they could distinguish the tolerable errors from critical errors (errors that could have an impact on real-time system execution). That is, by analyzing the neural network's incorrect output.

They then proposed a strategy adapted from the already developed Algorithm-Based Fault-Tolerance (ABFT) [107] to detect and mitigate these errors, thus improving the DNN reliability. Two important observations noted from this study: (1) transistor layouts of the three GPUs affect the soft error rate differently; and (2) the architectural differences among GPUs have different impacts regarding the error propagation process.

The authors concluded that the reliability of the DNN is accelerator dependent, and one cannot generalize a specific accelerator's case to all others in the same category. Since the ECC is a memory protection technique in GPU's memory hierarchy, consideration should be given to disabling or enabling it when implementing CNNs in particular.

In our previous work [108], we proposed a comprehensive analysis methodology for CNN-based classification models to confidently identify the only vulnerable parts of the source code. To achieve this goal, a technique call Layer Vulnerability Factor (LVF) has been proposed, and another technique, Kernel Vulnerability Factor (KVF), has been adopted.

LVF is a metric used to evaluate each layer's vulnerability, which is the probability of faults in a kernel that can affect the computations. Whereas KVF is a metric used to evaluate each kernel's vulnerability, the probability of faults in a kernel can affect the computations.

To validate the proposed methodology, the two techniques have applied to the GoogLeNet model, a well-known CNN architecture used for image classification. The obtained results show that only tiny portions of the GoogLeNet algorithm are susceptible to soft errors. Thus, the GoogLeNet model parts that need to be hardened were precisely identified. Hence, only three kernels, *Im2col*, *Normalize*, and *Add_bias,* were identified as top vulnerable kernels. The proposed methodology likely to be valid for other CNN models. Although the authors have not proposed mitigation solutions, this can be a pre-step for selective-hardening techniques, which help avoid unnecessary duplication overheads.

Although Lunardi *et al.* exclusively investigated the effectiveness of using ECCs in GPUs [109]. One of the eight benchmark applications used in this study to validate the proposed method is YOLO object detector. Although ECC is one of the most effective solutions for masking the impact of soft errors, the authors reported that some memory resources, such as flip-flops in pipelines and queues, are not covered by the ECC protection technique. Therefore, it would be useful to see an analysis on this subject regarding DNN algorithms. The authors also highlighted that all eight applications used in their test, except for YOLO and GEMM, are experiencing data reuse cases. The reason is that these two applications are more computationally intensive than the others.

This means that YOLO is designed to process data instantly, diminishing the time that data resides in memory elements, which would reduce the protection with ECCs. As a result, based on this study's finding, DNN applications (i.e., YOLO) will not benefit from the ECC technique as other applications do, such as sorting algorithms. More interestingly, among the eight benchmarks, the error rates of Silent Data Corruption (SDC) for YOLO is the only one that has not been reduced by ECCs. Moreover, the authors reported that compute-bound codes, YOLO is one of them, could encounter a higher crash rate than other benchmarks. Finally, authors have empirically observed that transistor technologies, CMOS and FinFET, have different impacts on resisting soft errors, showing that adopting FinFET is more effective than even enabling ECC to reduce SDC rates.

Arechiga *et al.* have examined the robustness of DNN architectures when their parameters experience disruption (i.e., bit flips) [110]. Three CNN-based models have been evaluated VGG16, ResNet50, and Inception V3. It should be noted that the number of parameters (i.e., trained weights and biases) for each architecture is substantially different. Where VGG16, ResNet50, and InceptionV3 have 138 M, 25.6 M, and 23.8 M, respectively. The author studied the impact of faults on these parameters and how the various given architectures behave. Keras, with the TensorFlow framework, was used to achieve the fault injection process. Although the "device under test" of this study is DRAM, GPUs were used to implement the given models. Accordingly, the target DRAM could be the off-chip RAM of the GPU. Therefore, this work has been classified as belonging to this category.

After injecting faults into weights of these models, they reported three observations: (1) Even if only a small percentage (i.e., 0.0000001%) of parameters get affected, it is enough to cause significant performance degradation. (2) The classification accuracy for all three networks shows only two choices: either the accuracy remains as it was, or a massive loss is observed. (3) Among the three models, VGG16 shows the less reliable against random bit flips in its weights. One reason is that ResNet50 and InceptionV3 utilize batch normalization between their main layers, but VGG16 does not. Batch normalization helps to normalize the inputs to each layer. Therefore, this technique probably masks some errors and enhances the model's robustness. The other reason is that again VGG16 is the only model that does not have "shortcut connections," which map the data from the input layer directly to the concatenation layer by skipping weight layers in the middle. This increases the resilience of the architecture by providing error-free data to the subsequent layers.

Santos *et al.* analyzed how radiation-induced errors in GPU architectures impact the reliability of CNN models [19]. In particular, they considered three CNN's architectures, YOLO, Faster R-CNN, and ResNet. The study shows that a single fault occurring in a GPU tends to propagate to multiple active threads, which leads to dramatic degradation in CNN's reliability. One of the best solutions provided by this work is that especial Max-pooling layers are designed instead of CNN's regular Max-pooling layers. These layers improved reliability by detecting errors at runtime.

Unlike [106], in which authors relied on only incorrect output on their study to evaluate the SDC. In this study, evaluation metrics such as Precision and Recall were used to distinguish between critical and tolerable errors in object detection frameworks. Accordingly, the authors have proven that relying on ECCs significantly reduces the number of SDCs but does not reduce the number of critical errors that could impact mission-critical applications. In contrast, in [106], the authors reported that the ECC should be enabled whenever available in a GPU. Again, this different perspective on enabling or disabling the ECC confirms that various GPU applications have different resilience capability to soft errors.

During the analysis, they observed that 67% of the GPU processing in YOLO, 80% in ResNet, and 82% in Faster R-CNN is spent in operations relevant to matrix multiplication. Based on this observation, they found that utilizing ABFT would be a useful technique for error mitigation. This is because ABFT is based on GEMM. By applying ABFT, they were able to detect and correct 87% of critical errors. This study concluded that the proposed technique is more efficient than the standard ECC solution and costs lower runtime overhead.

In another work of ours [111], we studied the impact of soft errors in CNN models implemented on NVIDIA GPUs. We specifically examined the error resilience of three CNN-based classification models, VGG-16, AlexNet, and ResNet-18. We performed intensive FI campaigns through the SASSIFI tool. To better understand and analyze the error propagation concept, SDC is also further classified to (1) Faultless (it is

masked in the middle); (2) Light-Faulty (it reaches the final output but does not cause misclassification); and (3) Faulty (it causes misclassification) SDC. By analyzing the obtained results, we show that 19.7% of the injected faults are Faulty.

Moreover, we analyzed error dependency from two perspectives: (1) The dependency on the SASSIFI's fault model (RF, IOA, and IOV). How different fault models affect the model's accuracy is investigated. (2) The dependency on the model's architecture (i.e., VGG-16, AlexNet, and ResNet-18). Whether the architecture of CNN has an impact on the model's reliability is investigated.

For the fault models, errors at RF mode, CNN models tend to generate DUE more than SDC errors, whereas errors at IOA and IOV mode are likely to produce more SDC than DUE errors. The reason is that errors at RF mode represent a more realistic injection than errors at instruction output modes (i.e., a higher level of injections). By comparing the LVF results of three fault models, the fault distribution among layers appears similarities. The error percentage of each layer in the RF mode is smaller than in IOV and IOA modes. The reason is that RF is injected only by one instruction, while IOV and IOA cover many instruction groups. Therefore, KVF and LVF are highly dependent on the fault model used.

For the network architecture, the three CNN models showed different trends. Where ResNet-18 and VGG-16 demonstrate that *Im2col* is the most vulnerable kernels in both, but AlexNet disagreed. This is because convolutional layers of these models are implemented with the *Im2col* kernel, which is invoked many times based on the CNN network's depth. As a result, ResNet-18 and VGG-16 have extra convolutional layers, which means extra invocations of *Im2col* kernel. Therefore, KVF and LVF are also dependent on CNN network architecture.

By hardening only a single kernel (i.e., *Im2col*) of VGG-16, the model's resilience significantly improved, reducing the rate of Faulty SDCs from 19.7% to approximately 0.4%.

Santos *et al.* studied the reliability of GPUs executing benchmarks in different floating-point precisions [112]. As modern GPUs support half-, single-, and double-precision, this work evaluated YOLO v3 on GPUs, targeting the impact of mixed-precision operations on the reliability behaviors. Both software fault injection and beam experiments have been reported in this study.

The fault-injection campaign has been carried on a GPU micro-architecture called Volta, which has mixed-precision operations with 2688 cores for double and 5376 cores for half/single format. Authors noticed that for single-precision (FP32) and half-precision (FP16) operations use the same number of registers, whereas twice this number for double-precision (FP64) operations. This indicates that by injecting the same number of faults into three different precisions, fewer cores would get affected for FP64.

For this purpose, the authors defined a metric called Mean Executions Between Failures (MEBF), which is the number of error-free executions accomplished before an error occurred. YOLOv3 is one of the five benchmarks used in this study. Yet, they observed that (for all used benchmarks) while reducing the precision (i.e., from double to single to half), the MEBF increases. This is due to a decrease in the execution time.

This study reported that detection errors are less dependent on data formats than classification errors. The justification is that, in detection case, bounding boxes (i.e., coordinates) are represented in integer values. Hence, it has nothing to do with injected digits in the exponent or mantissa. Therefore, reduced-precision formats significantly improve the reliability of the CNN model. Since lower precisions require fewer memory resources, they also improve the model's performance by reducing memory access.

In a very recent work of ours [102], we studied the reliability of a CNN architecture called Residual Network (ResNet) on GPUs. More specifically, we analyzed three popular ResNet models, namely, ResNet-50, ResNet-101, and ResNet-152. SASSIFI is the fault injector used for evaluation, which allows us to inject errors at memory elements (i.e., Register File) and compute elements (i.e., Instruction Outputs). The generated error is then either DUEs, SDCs, or Masked.

The reason behind examining three ResNet models is to investigate whether the number of layers in a model impacts the model's reliability, due to the number of layers in these models are different 69, 138, and 205 layers, respectively. Once we injected faults into the three architectures, we observed that the depth had affected the models' resilience differently. For depth analysis, we concluded that the more profound the model, the more errors it can produce. The logic behind this occurrence is that the same kernel's execution time is different from model to another. Therefore, we can find that ResNet-152 has the highest FLOPS and more layers than the two others; accordingly, it generates more errors.

Next, we carried out an in-depth vulnerability analysis on the ResNet's kernels. The kernel is the portion of the parallel code (i.e., function) executed on GPUs (i.e., not CPUs). To accomplish this, we first gathered all the static kernels built in this model required for inference. The analysis shows that the ResNet model's different kernels make a big difference regarding their vulnerabilities against soft errors. It is beneficial to determine which kernels tend to produce more errors than the others to be selected for hardening.

Then, we analyzed the vulnerability of ResNet's layer and how errors propagate through these layers. ResNet-50 has been chosen to analyze error propagation, which is composed of 69 layers in total. The analysis results reveal that layers 0, 4, 8, and 12 generate 37% of the model's critical SDCs. It makes them the most susceptible layers. By looking at these layers, we find that their input sizes and the number of filters are more significant than others. Also, about 92% of the SDC errors occurred at Convolutional and Shortcut layers. Because these layers have several kernels, and the number of invocations per kernel is equal to the number of layers. Softmax is found to be the most reliable layer. The reason is that the execution time in which errors could be injected is relatively short (i.e., only one invocation).

Finally, based on the analysis results, we proposed a "selective-hardening TMR" solution to address soft error problems in ResNet models. By applying the proposed approach to four kernels (out of 11), up to 93.38% of the injected errors were masked with performance overhead less than 5.35%. Additionally, the percentage of critical errors was reduced from 4.2% to 0.104%. Thereby, the reliability of the model increased remarkably.

5.2. ASIC

This part reviews and presents studies evaluating the resilience of ASICs during the implementation of DNN models. For each study, various elements for DNN models/ accelerators might be considered. Therefore, Tables 5 and 6 represent and list this category's representative works along with the components and metrics considered in each work.

Li *et al.* analyzed and evaluated the fault-tolerance characteristics of DNN models running on a dedicated ASIC accelerator called Eyeriss [20]. Error resilience analysis has been conducted on four CNN-based models, AlexNet, CaffeNet, ConvNet, and NiN. This work is one of the pioneering studies of concepts such as error propagation in modern DNN systems. It thoroughly diagnoses the way errors propagate from the

Table 5
DNN component whose reliability is analyzed on ASIC accelerators.

Component/Factor	Ref.
Activations & parameters	[20,85,86,114,115]
Layers	[20,85,114]
Data formats	[20,113,115]
Considered dataset	ImageNet [20,85,114], MNIST [85,86,113,114], CIFAR-10 [20, 86, 114], Reuters [85,86], TIMIT [85,115], IMDB [86]
DNN Architecture	CNN [20,85,113,114], MLP [85, 86, 113, 115]
Purpose of the DNN	Classification [all of them], Object Detection [none]
DNN phase	Inference [20,86,113], training [85,114,115]

Table 6
ASIC accelerator's element whose reliability is analyzed.

Element/Factor	Ref.
Memory element	[20,114,115]
Processing/control element	[20,85,86,113,114]
Evaluation approaches	SW [all of them], Beam [none]

accelerator level to the DNN model's level.

After injecting errors and performing an in-depth analysis, the authors came to some significant outcomes. They concluded that the error resilience of a DNN depends on several factors, including (1) the network topology; (2) data format used for representation (i.e., FxP and FP); (3) reuse of data in DNN accelerators' dataflows; and (4) the position and number of layers in the DNN network. Therefore, DNNs differing in their internal structures usually exhibit different sensitivity behaviors towards SDCs.

This research has conducted a wide-ranging investigation of memory elements and datapath (i.e., not the compute elements themselves). Nevertheless, it did not consider faults occurring in other computing resources, such as compute elements, arguing that they are much less sensitive to transient faults than storage elements. However, in other accelerators, such as GPUs, a failure in computing elements causes more severe consequences than those in memory elements. Hence, crashes in a device caused by a failure in these elements are more than those caused by storage components [19].

Authors observed that when using the same data format with NiN, AlexNet, and CaffeNet models, they obtain roughly the same rates of SDC errors, whereas ConvNet gives a very high rate. The reason is that the three models work on the ImageNet dataset, which has 1000 classes. If the injected error is one of the top-ranked outputs, the confidence scores will likely change by more than 20%, so the new ranking likely does not belong to the top-five elements. On the other hand, ConvNet works on the CIFAR dataset, which has only ten classes, and if the injected error is one of the top-ranked outputs, it may still belong to the top-five elements. Thus, fewer classes per dataset can lead to a higher vulnerability of the model.

Based on their analysis results, the authors proposed two error protection strategies to mitigate SDCs effects: (1) Symptom-based Error Detectors (SED) to detect SDC errors in memory systems. They defined Precision and Recall as metrics for evaluating SED. (2) Selective Latch Hardening (SLH) for detecting and correcting SDC errors in the datapath. With the SED technique, the obtained average precision is 90.21%, and the average Recall is 92.5% for three given models. Thus, the FIT rates of Eyeriss can be significantly reduced. With the SLH technique, it turned out that the overhead area required by SLH is near equal to the area required by ECC mitigation for larger SRAM structures. Applying the two proposed techniques demonstrates that they can significantly reduce the SDC error rate in DNN systems with a small accuracy loss and area overheads.

Pandey *et al.* studied the error resilience of TPU-based accelerators [86]. More specifically, they evaluated the timing errors resulting from the near-threshold operations of TPU. This study initially intended to suggest a TPU design paradigm called GreenTPU. Its primary purpose is to restrict the energy consumption of TPUs. In the systolic array of MACs, the proposed design can predict the activation sequences' patterns that cause errors. Thus, it can prevent other errors from the same sequence by increasing the operating voltage of the specific MAC units in the TPU.

TPUs compute 8-bit integer arithmetic with an extremely homogeneous architecture (i.e., a transparent dataflow model). This means a MAC unit can only see a small number of sensitized path delays. In a MAC unit, the multiplier portion has a deeper logic depth than the accumulator portion. Thus, the idea is to model the delay of the MAC to the multiplier, i.e., 8-bit activation and the weight. They noticed that by applying the same activation sequence in two sequential cycles, no paths are sensitized. Further, most of the multiplication operations sensitize paths have tiny delays. This helps to predict the repeating errors from the same input sequence.

They realized that an input sequence that causes an error in a MAC unit is very potential to cause errors in following MACs as well. That is because the sequence continues to be in the row. Thereby, the proposed technique predicts errors regulated by the input sequences. In a given row of TPU, a unit is inserted to link activation memory to the MAC units row. This unit is responsible for detecting and mitigating these errors.

The same as Razor, a flip-flop unit is used to detect a timing error in each MAC unit. When a timing error occurs in any row, this unit logs the input pattern that caused this error. Concurrently, it increases the supply voltage of the following MACs in the same row, to block any future timing error. They adopted the mitigation technique from [86], then adapted and embedded it in GreenTPU. Comparing the proposed technique with the timing-error drop introduced in [86] shows that their technique mitigates timing errors at near-threshold two orders of magnitude. Significantly increases the model's reliability.

Jiao *et al.* evaluated the impact of soft errors in processing elements of an ASIC-like accelerator executing DNN models [113]. Soft errors here are Time Errors caused due to temperature and dynamic voltage variations. The evaluated DNNs architectures are a 3-layer FC (MLP) and a LeNet-5 (CNN). The processing elements are Adder and Multiplier units because they are the two most frequently used computation units. The authors hypothesized that if a time error is injected, the processing element will return a random value.

To simulate the hardware errors, they run the accelerator under 20 different pairs of (voltage and temperature) conditions. This step is to recognize the timing error behavior of FP Adder and Multiplier for these twenty conditions. They then evaluated the accuracy of the two models on the MNIST dataset at each condition. They injected faults in the two units in three modes: Adder alone, Multiplier alone, and both at the same time. They defined multiple error rates to see when the model's accuracy starts to drop. They observed that for both MLP and CNN, a fault in the Adder leaves a higher impact on the model's accuracy than a fault in the Multiplier. Moreover, injecting faults in both units at the same time will produce accuracy almost identical to injecting faults in the Adder alone. This shows that errors caused by changing Adder's value lead to most of the inaccurate predictions.

The reason is that the intermediate convolutional or dot-product results are fed to a nonlinear function (i.e., Sigmoid). While errors coming from Multipliers will be averaged, those coming from Adders immediately influence the Sigmoid function's input. With increasing the error rate, the accuracy falls dramatically.

Their results show that, in general, variation-induced errors can significantly influence the classification accuracy of both networks. Also, both voltage and temperature appear to influence the prediction accuracy of the models equally. By fixing one of them and varying the other, the accuracy drops significantly, from 85.15% to 48.64% and 70.34% to 48.64%. When comparing CNN with MLP, at a defined error rate, it shows that the accuracy reduction of CNN is much than that of MLP, showing that CNN might be more vulnerable to errors than MLP. One reason is that since CNN has more arithmetic computations than MLP, for a given error rate, it can produce more errors.

Choi *et al.* introduced error resilience techniques for DNN accelerators based on the sensitivity difference of DNN filters and weights [114]. They analyzed the sensitivity using first-order Taylor expansion, which is a tool that helps to approximate the DNN component's sensitivity from actual FI simulation for deeper DNNs. Consequently, they defined a DNN component's sensitivity as the amount of accuracy loss caused by injecting an error in the filter weight.

The author's main idea is to take advantage of the sensitivity difference among filter weights to modify the DNN accelerator design. Thus, computations with high sensitivity weights are mapped to more reliable MAC units of the accelerator. In contrast, those that come with low sensitivity are assigned to moderate reliable MAC units.

Accordingly, they proposed changes in the architecture of a TPU-like systolic array. To verify the proposed approach, the authors evaluated it on four CNN-based networks, LeNet-5, VGG-9, VGG-16, and ResNet-50. The proposed technique demonstrates that the classification accuracy of the given models remained very high. The worst case of the VGG-9 model revealed only a 3% accuracy loss.

Lee *et al.* widely examined the fault tolerance of DNNs implemented on a CMOS digital circuit [115]. They mainly focused on data representation formats and specifically analyzed the fixed-point format. This study does not directly address soft errors. However, it addresses the weights' sensitivity problem through a technique called "weights dropout" instead of the already existing technique "neurons dropout." Nevertheless, even from a fault tolerance perspective, a DNN trained with neurons dropout is more tolerant than the ordinary DNNs [116]. Therefore, this study can be classified as reliability work as well.

The authors hypothesized that the fault tolerance of neural networks depends on several factors, and three of these factors have been studied in this work: (1) the fault model, in which two matters are considered, namely, interconnection with weights and processing units such as neurons; (2) the network size, specifically the impact when the number of units in each layer varies; (3) the training method, which is the impact of changing the training technique. Based on their analysis, the authors proposed a new technique that randomly disconnects weights either stuck-at-zero or random-error, during the training to increase the tolerance of faults. The findings indicate that, with appropriate training methods, DNNs' prediction accuracy can be reduced by only a small amount if the error rate is less than 10%. It is worth mentioning that the same technique can be evaluated under soft errors.

Zhang *et al.* studied the resilience of DNNs and types of reliability faults that arise at the architecture of a systolic array-based DNN accelerator [85]. More specifically, the propagation of soft error that occurs due to timing errors in voltage scaling. They observed that each single MAC computation contributes with a tiny proportion to the output of a neuron.

Based on the analysis results, they introduced a novel technique called timing-error drop (TE-drop) to enable aggressive voltage underscaling in AI accelerators without affecting the model's accuracy. More explicitly, to mitigate soft errors in MAC units of this ASIC accelerator. The idea is that TE-drop provides each MAC unit with "Razor flip-flops" to detect these errors. It masks the errors without the need to re-execute the faulty MAC operation to eliminate the error. Once a MAC unit experiences an error, TE-drop steals the very next clock cycle from its next-in-line MAC unit to add its contribution to high-order bits of MAC outputs. Thus, it either bypasses or drops the following MAC operation.

Besides "Razor flip-flops," the TE-drop approach also uses a multiplexor, which its output is controlled by the "error signal" from the preceding MAC unit. If the preceding MAC unit experienced an error, then the multiplexor forwards the correct value of the partial sum obtained from the preceding MAC to the succeeding MAC unit; otherwise, this MAC unit computes the "partial sum" and passes it without the need of all these details. Dropout and dropconnect are well-known techniques applied in DNN models to avoid overfitting during the training phase. The same idea is applied for TE-drop, which randomly drops neurons and connections.

5.3. FPGA

This part reviews and presents studies evaluating the resilience of FPGAs during the implementation of DNN models. For each study, various elements for DNN models/ accelerators are considered. Therefore, Tables 7 and 8 represent and list this category's representative works along with the components and metrics considered in each work.

Clemente *et al.* provided a hardware solution for FPGAs to enhance the reliability of a specific type of ANNs called Hopfield Neural Networks (HNNs) [117]. The proposed solution aims to harden the adders and multipliers. Spatial redundancy is used for additions, whereas temporal redundancy is used for multiplications. Once the proposed design results are compared to standard HNNs and the solution based-on TMR, it outlined both of them.

Table 7
DNN component whose reliability is analyzed on FPGA accelerators.

Component/Factor	Ref.
Activations & parameters	[70,112,117–119,121]
Layers	[118,119,121,122]
Data formats	[70,112,118,119,121]
Considered dataset	ImageNet [70], MNIST [112,117,118,121], CIFAR-10 [121], GTSRB [119]
DNN Architecture	CNN [70,112,119,121,122], MLP [109,117,118,121,122]
Purpose of the DNN	Classification [all of them], Object Detection [none]
DNN phase	Inference [all others], training [70]

Table 8
FPGA accelerator's element whose reliability is analyzed.

Element/Factor	Ref.
Memory element	[70,112,117–119,121,122]
Processing/control element	[112,117–119,121]
Evaluation approaches	SW [almost all], Beam [112,119,122]

Their findings show that faulty results decreased from 2.36% to 0.14% in the case of SEUs, from 2.18% to 0.032% for SETs, from 9.44% to 3.07% for stuck-at-zero, and from 8.67% to 3.06% for stuck-at-one. Since this study targets HNNs that developed specifically for hazardous environments (satellite), as in [113], a specific architectural design might be a suitable choice. Nonetheless, due to its high level of customization, the associated hardware resources overhead, about +50%, could restrict other scenarios. Furthermore, the impact of different portions of ANN, such as layers and activation functions, and how they respond to the injected faults is not documented. Thus, this solution covers a narrow prototype of ANN, which is difficult to be generalized to DNNs.

Xing *et al.* proposed a strategy to tackle the issue of training with overclocked accelerators, which directly leads to accuracy loss (i.e., reliability problem) [70]. They observed that when the accelerator's frequency goes overclocking, the error rate can go quite high. In consequence, the accuracy of a DNN accelerator may drop dramatically. To tackle this problem, they proposed a technique that works during the training phase. So, to re-train with the 'unstable' accelerator to mitigate soft errors.

They first assumed that the off-line training is achieved on a CPU or GPU, and inference can be performed on an FPGA accelerator. They validated their technique on Xilinx KCU1500 FPGA using four CNN architectures: LeNet, AlexNet, VGG-16, and VGG-19. The standard frequencies for these models are 210, 210, and 190 MHz, respectively. For the Forward propagation, weights with FxP data formats are applied on the FPGA accelerator to preserve resource utilization. For the Backward propagation, weights with FP data formats are applied on the CPU or GPU to ensure tiny weight Adjustments get accumulated.

Once the models AlexNet, VGG-16, and VGG-19 are retained with the proposed strategy, their accuracy improved by 3.4%, 1.8%, and 2%, respectively, at the extreme overclocking frequency. However, for LeNet, it does not guarantee its accuracy until the frequency crosses 260 MHz and suddenly falls at 270 MHz. Therefore, retraining LeNet leaves no effect on its accuracy; thus, it is considered reliable compared to the other three. One drawback of this technique is that, while updating the weights during the training, this requires converting between the FxP and FP formats in every single iteration. As it takes extra converting time, this is a severe problem in large networks.

Lopes *et al.* analyzed the impact of soft errors by disrupting the memory control bits, that is, the configuration bits of SRAM-based FPGAs [118]. In some cases, changes in these bits provoke errors in

the output of DNN and even system failures in other cases. The authors analyzed the critical bits (out of millions) of configuration bits and found that the model's output can differ once they are flipped. To validate their assumptions, they proposed two different ANNs, namely Dilation Neural Network (DNN), used as the design under test, and Erosion Neural Network (ENN), which is used to analyze the propagation of the errors.

The analysis was done by taking an image generated by the DNN to be processed by the ENN. Subsequently, to verify whether DNN's errors were propagated to ENN. Since the case study considered was for classification tasks, the authors defined their fault model as follows: a wrong classification means failure; a difference in the outputs means error.

They took activation functions and layers as their targets for evaluation. After the faults are injected, the results showed some trends, that neurons in layers closer to the output seem to be affected more than neurons in earlier layers. This denotes that layers near the output are more prone to soft errors, and thus, are the critical components of the network.

The proposed neural network achieves a decent fault-making capability. However, as it cannot easily be extended due to the scalability issue, this approach can be applied only in relatively small networks. On the other hand, most of the safety-critical systems are likely implemented in more extensive networks, which require higher standards of error-detection capabilities.

Lopes *et al.* evaluated the reliability of a CNN implemented on a small chip, the All-Programmable System-on-Chip (APSoC), which is mainly an SRAM-based FPGA [119]. As in [118], the programmable logic's configuration bits are targeted to disrupt the operations to evaluate the impact of soft errors, particularly SEUs. In their assumed fault model, authors defined bits responsible for its Architecturally Correct Execution (ACE), whereas the rest are called un-ACE bits. So, flipping any of the ACE bits could result in SDC errors.

CNNs use plenty of LUTs to hold the weights and routing signals to map outputs to inputs. They noticed that if the injected fault in the configuration bits is associated with these LUTs, it does not result in a system failure. However, only SDCs (i.e., often critical SDCs) are generated, which cause misclassification in a traffic-sign recognition system. Accordingly, if the LUTs implement the state machines' logic or the routing signals used by them, they are more likely to generate time-out errors.

They also performed a neutrons beam at Los Alamos Neutron Science Center (LANSCE). Their experiment findings indicate that the failure rate could be tolerable in some cases, but in the worst case, the failure rate is unacceptable. Based on this analysis, the authors proposed a timing-multiplexing architecture as a solution, where the model's inputs and weights fed into MAC units are multiplexed in time.

This study's main limitation is that it does not evaluate the output layer, which is Softmax. The authors justify that due to the large number of hardware resources required by Softmax, it was implemented in software only. However, considering other studies such as in [120], the authors show that transient faults increase closer to the output layer. Thus, it is interesting to see the evaluation of Softmax with such a technique.

Khoshavi *et al.* [121] studied the impact of soft errors in a particular type of DNNs called Binarized Neural Networks (BNNs), which was implemented on FPGA (Zynq-7000) accelerators. BNNs are DNNs that use binary values for its data, instead of full precision values. In BNNs, all activations and weights are represented with 1-bit and 2-bit. This compresses the network information in a reduced memory. Consequently, BNNs are much smaller in size than the full-precision versions. The benefit is that it reduces execution time but in the cost of their accuracy.

The study considers two different BNN architectures, (1) a VGG-like CNN, which is used to classify the CIFAR-10 dataset, and (2) MLP is used to classify the MNIST dataset. For CNN, there are two formats to represent weights and activations: cnvW1A1 uses 1-bit to store the parameters, while cnvW2A2 uses 2-bit. There are approximately 1.6 and 3.2 million vulnerable bits to transient faults in W1A1 and W2A2 networks, respectively. For MLP, also there are two formats for the same purposes, lfcW1A1 and lfcW1A2. There are approximately 3 million vulnerable bits to transient faults in both topologies.

They examine the behavior of Single-Event Upset (SEU) and Multi-Bit Upset (MBU) on both memory and compute elements. Apart from the different topologies, they also evaluate the impact of soft errors on data formats used to represent the weights and activations.

By performing fault injection, authors demonstrate that for the worst-case scenarios, the classification accuracy can dramatically drop from 98.4% to 22.92% in the lfcW1A1 network and from 80.5% to 62.0% in cnvW1A1. From the view of the different factors have considered in this study, results show that injecting faults in the activation functions with both (SEUs and MBUs) error models is more vulnerable than injecting them in weights. Further, the MBU showed a higher effect than SEU. Also, earlier layers showed a higher impact against transient faults than the later counterparts.

Libano *et al.* [122] proposed a selective hardening strategy as an alternative to the conventional Triple Modular Redundancy (TMR). The authors used two different neural networks and implemented them on SRAM-based FPGAs to evaluate their outputs' correctness. According to their analysis, they discovered that whether or not the errors affect the neural network's functionality, soft errors could change the output of these networks. Most of the observed errors were tolerable, modifying the output without affecting the functionality. Therefore, the proposed solution can be achieved by determining the most critical layers in each neural network and triplicating only the most vulnerable layers of a given neural network, instead of using full triplication with TMR, thus minimizing the overhead.

A notable point in their experimental results is that, although the two networks, ANN and CNN, are very different in their topologies, the proposed solution can equally influence them. Further, their overall reliability improvements follow a similar trend as a result of a selective approach. Thus, in the ANN case, the selectively hardened strategy masked 68% of faults with a 45% area overhead, whereas in the CNN case, 40% fault-masking was achieved with only 8% overhead.

Santos *et al.* studied the impact of mixed precision on the reliability of the FPGA executing CNNs using different precisions [112]. They evaluated two benchmark algorithms on FPGA, matrix-multiplication and CNN-based (MNIST) ones. Only CNN algorithms are considered for this review.

Since the implementation of a particular FPGA is determined by the bitstream stored in configuration memory, soft errors always leave a specific impact on FPGAs. Injecting faults into configuration memory changes its status and sticks to it until the current bitstream is updated to a new one. This implies that after any error observation, the FPGA should be reprogrammed in order to consider a new bitstream. Therefore, unlike other accelerators, data formats in FPGA are straightforward.

As seen in the GPU category (Section 5.1), MEBF is used here also as an evaluation metric. Remember that the larger the value of MEBF, the reliable the accelerator. They also defined another metric called Acceptable Relative Error (ACRE), which is the largest tolerable difference from the golden output. For FPGA executing the MNIST model, Failure in Time (FIT) is measured as well.

When they injected faults with different precisions and change the ACRE from 0% to 0.1% with all precisions, they obtained results with the following trends: (1) for double-precision, a considerable reduction in the FIT; (2) for single-precision, a smaller reduction in the FIT; (3) for half-precision, a slight reduction in the FIT. The justification is that, with smaller precision, a fault seems to change the data's MSB, which is reasonable to cause a more harmful error. Although critical errors were much less than their non-critical counterparts, they increase when moving from double to single to half-precision. Furthermore, MEBF was increased when moving in the same direction. Confirming similar trends with GPU accelerators case (see Section 5.1 the same reference), which

is that "reduced precision means increased MEBF."

5.4. Others

As demonstrated in Section 3.4, this category does not address specific accelerators. Instead, it presents works that do not explicitly belong to a particular category. Therefore, studies in this category provide solutions that can be applied to various DNN accelerators in different categories. Tables 9 and 10 list representative works of this category along with their metrics and components considered in each work.

Azizimazreah *et al.* studied the impact of soft errors on DNN accelerators' reliability from a memory-only perspective, assuming that 75% of a DNN accelerator is used as an on-chip buffer [40]. They proposed a new technique to solve the limited capabilities of existing fully-hardened SRAM cells. This alternative solution is called a Zero-Biased MNU-Aware SRAM Cell (ZBMA). The former technique is limited because it can resist only faults produced by single-node-upsets, but it cannot prevent transient faults caused by multiple-node-upsets.

The authors targeted errors that disturb feature maps and weights stored in memory elements. The proposed approach is constructed on two observations: (1) data (i.e., feature maps and weights) in DNNs has a strong bias towards zero; and (2) data flipping from 0 to 1 has a higher probability of causing a failure in outputs of the DNN than from 1 to 0. The findings indicate that once the proposed technique is applied to a DNN accelerator, 99.99% of the faults caused by soft errors will be reduced across different DNNs. However, while this solution is very efficient for inference devices, it is not suitable for training devices. The number of computations required in the training process is exceptionally high, making the positive feedback between the nodes of this approach problematic. Therefore, this solution can be more appropriate for inference devices.

Schorn *et al.* studied the error resilience of individual neurons in DNNs [123]. Accordingly, they propose an approach for predicting DNN error resilience on an individual neuron level. As all neurons in a network cooperate to produce a single output, changing the value of a single neuron can have a little impact on the final accuracy. Thus, the neuron's contribution to the model's output reflects its resilience.

They formulated a method to estimate every neuron's contribution to the final accuracy of a DNN by using some algorithm. This algorithm can calculate every neuron's contribution in this layer and send it back to the neurons in the preceding layer. This step is vital to identify which neurons have the lowest resilience to be protected. Hence, to map more resilient neurons to less reliable compute elements of the accelerator. The average resilience of overall neurons per layer is equal to the number of neurons in that layer.

They validate their approach on two networks, VGG16 and All-CNN. The prediction methods and FI campaigns are implemented in Keras framework, using the Theano backend. Finally, they compare their method with two other methods. The results reflect the superiority of their technique achieving higher accuracy with the number of faulty neurons increased.

Neggaz *et al.* studied the impact of soft errors on CNN models' accuracy for embedded systems [100]. Authors have discovered that a previous work proposed a technique for predicting neurons' error resilience in a DNN. However, multi-bit faults and layer-wise evaluations were not considered. Therefore, their goal is to evaluate the inherent fault tolerance of CNNs by altering the floating-point bits of processing and memory elements.

Their evaluation method is that faults in weights and activations are compared to approximate computing. Note that errors in memory affect weights stored in SRAMs, while errors in computing elements for incorrect propagation can produce corrupted layer activations. To validate this methodology, LeNet5 has been utilized. They considered layers as a volume of activations; the three layers of LeNet5 have been considered. No specific accelerator has been targeted in this study; instead, the FI has been performed at the application/model level through the PyTorch framework.

After injecting errors in each of the two elements, obtained data indicate that faults in processing elements (i.e., activations) only have a slight effect. The network itself can mask it. On the other hand, errors in memory elements (i.e., weights) result in a considerable accuracy loss.

Moreover, the first layer of LeNet5 seems to be the most resilient compared to the other two layers. Also, FC layers have shown higher error rates compared to convolutional layers. Because errors might get masked by layers that follow convolutional layers, such as max-pooling layers, in contrast, in FC layers, if an error did not get masked by an activation function (e.g., ReLU), it would directly impact the output. Flipping mantissa bits shows only a limited impact. Even multi-bit flips in activations can produce only a few errors.

Marques *et al.* analyzed the sensitivity of the DNN, specifically CNN, by injecting faults into the feature map blocks in the memory during the training process to evaluate the training error degradation [124]. They focused mainly on data types (i.e., floating and fixed points). As a solution, the authors proposed a technique to protect the most significant bit (MSB) in the stored data.

Hoang *et al.* studied the error resilience of DNNs to determine the impacts of hardware faults on classification accuracy [125]. In particular, they analyzed the impact of faults in the weight memory by perturbing the model's parameters and activations stored on this memory.

They note that the memory consumption of DNN models varies greatly depending on the number of parameters used by each model. Consequently, they evaluated their proposed technique on two models with different parameter numbers, AlexNet and VGG-16. They performed per-layer FI to evaluate the susceptibility of individual layers and the effects of the faults on the produced activations. Some observations have been drawn from their analysis: (1) Model's accuracy decreases with an increase in the fault rate. At the same time, it remains close to the baseline accuracy at lower fault rates before dropping drastically. (2) At higher fault rates, the distribution of activations also has values of higher intensities.

To mitigate errors, based on the analysis and the pulled observations, they proposed a new version of the ReLU activation function to squash high-intensity activation values to zero; it is called Clipped Activation Function. It replaces the unbounded activation functions in the DNN with their clipped alternatives.

Finally, to validate the proposed method, they compared the accuracy of the DNNs, protected with clipped activation functions (at low and fault rates) and unprotected DNNs, for both AlexNet and VGG-16. Their findings indicate that by applying the proposed method, networks significantly improved the error resilience of the DNN. For AlexNet, the accuracy reached 69.36% at fault rate $5 \times 10–7$ compared

Table 9
DNN component whose reliability is analyzed on Others accelerators.

Component/Factor	Ref.
Activations & parameters	[40,100,101,124–128]
Layers	[40,100,123,125–128]
Data formats	[100,124,127,128]
Considered dataset	ImageNet [40,123], MNIST [100,101,124,126–128], CIFAR-10 [101, 123, 125, 126], PASCAL VOC [127]
DNN Architecture	CNN [40,100,101,123–127], MLP [101,126,128]
Purpose of the DNN	Classification [all others], Object Detection [127]
DNN phase	Inference [all others], training [124]

Table 10
Others accelerator's element whose reliability is analyzed.

Element/Factor	Ref.
Memory element	[nearly all]
Processing/control element	[100,123,128]
Evaluation approaches	SW [almost all], Beam [115,124,128]

to 51.16% for the unprotected version. While for VGG-16, the technique provided even better improvement.

Liu *et al.* studied the impact of adversarial attack determined to a DNN model [126]. For this purpose, they target the model's parameters (i.e., weights and biases) to investigate how adversarially robust are the models when their parameters are attacked. The authors assumed that no matter the type of accelerator used to implement DNN algorithms, the parameters are eventually stored in memory. Thus, evaluating the resilience of the model by modifying the parameters in memory will be sufficient. Disturbing these parameters will disturb the DNN resilience. It is worth noting that this study is basically on adversarial attacks for DNNs. However, as the authors stated that attackers could only cause misclassification to the predictions, the capability of a DNN to tolerate faults against random faults is the goal. Consequently, it is the same as soft errors in memory elements.

Based on these observations and assumptions, they proposed two fault-injection techniques: (1) single-bias attack (SBA) is achieved by modifying one parameter (i.e., one bias) because they observed that the model's output could linearly rely on some parameters. (2) Gradient descent attack (GDA) is used to control the fault injection (i.e., parameter modification).

Faults have been injected in the memory with the two widely used techniques: Laser beam and row hammer. They then examined the impact of changing biases in the model's accuracy. The results demonstrate that the proposed techniques can cause misclassification on both DNN models. For SBA, the average of the accuracy is less than 60%, while GDA's accuracy is about 81.66%. The reason is that the SBA imposes significant changes on a single bias, while the GDA imposes small changes on multiple biases (parameters). Thus, the total amount of changes in parameters imposed by GDA remains smaller than changes by SBA.

Bosio *et al.* analyzed and provided a methodology to evaluate the impact of faults affecting the connection weights of CNNs [127]. Two CNN topologies have been considered, LeNet5 for classification and YOLO for object detection, to validate the proposed methodology. This study is independent of any hardware architecture; instead, it performs fault injection at the software level. It is worth mentioning that this study evaluated permanent faults. However, since the adopted strategy does not assume stuck-at errors, the same methodology would be applied for soft errors as well. Thus, it has been included in this category.

When a fault is injected, authors classify its effects into Safe, Unsafe, and SFAD "Safe Faults Application Dependent." The SFAD is the union of masked and non-critical faults. Hence, the crucial step is to identify SFAD faults to neglect them and focus on the faults that can cause failures. The Ultimate goal of this study is to evaluate the distribution of SFAD, which helps to identify the most sensible layers of a CNN. They also identify the most critical bits of the variable storing the weights.

For LeNet5, the FC layers produce less critical than Convolutional ones. Note that These findings confirm other findings found on GPU accelerators [102]. Another interesting observation is that no dependency on the input image has been observed. For YOLO, FC and convolutional layers tend to show a similar capability of masking faults. Unlike LeNet, there is a high dependency on the input image that has been observed in YOLO case. The reason behind this is that there is only one object in the classification case, while in the detection case, there are many objects within one image. Thus,

While the two networks show different trends, the evaluation of floating-point bits is the same. The analysis results show that all the critical faults occurred due to changes in the exponent bits. Particularly, using a single-precision floating-point format by the IEEE 754 standard, bits from 30 down to 23 are critical.

Although this work does not provide any error mitigation technique, it guides designers to choose the most appropriate hardware architecture for a given DNN model. Thus, it will lead to reducing the cost of the test solution by determining the exact critical parts of the model for the test engineer.

Breier *et al.* studied the reliability of a DNN implemented on a microcontroller unit (MCU) [128]. This study targets the activation functions of DNNs to change the prediction accuracy of the DNN, that is, to achieve misclassifications. The four most commonly used activation functions, ReLU, Softmax, Sigmoid, and Tanh, were considered. As they provide non-linearity behaviors, they form an essential building block of any DNN to be examined.

Their results show that by injecting faults into these functions, misclassification can be achieved. Additionally, disturbing activation functions in the layers closer to the output layer have more impacts. The authors assumed that the attacker does not care about the outcome class, only making a change in the correct value. Thus, even if this work targets adversarial attacks, the evaluation method could be applied for transient hardware faults. Furthermore, the simulated attacking process is performed with a laser fault injection, which is typically used to emulate radiation-educed errors.

As this study evaluated the resilience by performing beam experiments through a laser, the evaluation metrics were not well demonstrated because it has been done without software fault injection for further analysis. Therefore, the authors stated that they could not analyze the Softmax function because it produces either no output or output with invalid values. However, since Softmax is used for the classification task at the output layer of the DNN, it is essential to be evaluated.

Arechiga *et al.* evaluated the error resilience of DNN models when errors occur in their weights [101]. Specifically, MLP and CNNs have been considered because they have different workflows, especially how each architecture arranges its weights to interact with their inputs. As weights are stored in memory elements, this study aims to inject faults in weights memory and analyze its impact. When the training is completed, errors are injected by selecting a weight randomly from the trained-model weights. Hence, a bit flip is applied randomly in that weight.

They noticed that, for both networks, the model's depth directly affects its resilience. For MLPs, the larger the size of a network (i.e., more layers), the more reliable it is. If the number of injected weights is the same in multiple MLPs, the percentage of corrupted weights is larger in a shallower MLP. This means that smaller/shallower networks tend to be more susceptible to errors.

For CNNs, models with larger convolutional filters are likely to be reliable than those with smaller ones. Consequently, if different CNNs have the same depth, then the filter size will be crucial. For instance, a model with 7×7 filters shows more robustness than a model with 3×3 filters. However, if the filter size is fixed, CNNs with different depths do not seem to be affected by the increasing number of injected errors. In general, CNNs are shown to be much more vulnerable to weight errors than MLPs.

Summary of this section: As can be noticed, except for Breier *et al.* [128], which is specific to MCU accelerators, works in this category can be applied to any accelerator in the other three categories. Further, most of the studies in this category address DNN reliability issues only from a memory perspective. In other words, only a few studies cover processing/control elements of the accelerators. However, since on-chip memory occupies a nontrivial amount (i.e., 75%) of the accelerator's area, it is worth seeing a dedicated research work on memory elements.

6. Discussions

This section discusses reliability issues in DNN models through their accelerators to gain a deeper understanding of this topic.

6.1. Points in each category

GPU Category: Transistor layouts of different GPUs have different error rates. Thus, various GPU microarchitectures (i.e., Maxwell, Volta, and Turing) impact the propagation of an error from one level to another until it reaches the output. While ECC has proven to be a very effective

protection technique used for various memory elements, ABFT strategy could provide a better enhancement in GPUs to harden DNNs in a software-based solution. The reason is that the majority of DNN operations in GPUs are bound to matrix multiplication, which is the fundamental task on the ABFT.

ASIC Category: ASIC is gradually evolving towards smaller structures, low power consumption, and high performance, albeit at inflexibility cost. From a reliability point of view, unlike GPUs, an ECC strategy cannot be applied directly to ASIC's buffers because matrix multiplications and convolutions are better implemented using systolic arrays, whose elements do not have local memories. As a result, the underlying DNN architecture might be modified when a reliability solution is proposed. Besides, as Google's TPU is currently the only prominent accelerator of this category, the benchmark problem would be a nontrivial issue for designers.

FPGA Category: Flexibility and reconfiguration features make FPGAs easier to evaluate for DNN resilience compared to other accelerators. Most of FPGA's DNN accelerators are designed to target only one type of data stream. For instance, accelerators proposed in [89–92] are specific for CNN models only. Hence, it is not straightforward to map various layers of a DNN on these accelerators with the same efficiency. SRAM-based FPGAs are one of the most commonly used types. When a fault is encountered, such as a particle striking the configuration bits of this memory, it can lead to a change in the configuration of a LUT, a BRAM, or a routing connection. To address the impact of transient errors in this category, hardware redundancy solutions are usually adopted.

6.2. Comparison of categories

The architectural differences among DNN accelerators make the complete comparison is an unfair and challenging task. However, the major differences that indicate each category's weak and strong points in terms of reliability against soft errors can be highlighted.

In this paper, two commonly used DNN accelerators (i.e., GPUs and FPGAs) are compared. Our comparison is as a first attempt to answer the question, "which accelerator is more reliable against transient faults?". The purpose is to demonstrate how the same model's resilience may have different behavior when encountered transient fault in different accelerators. For this purpose, a specific DNN architecture should be chosen as a benchmark. The CNN architecture was chosen because it has been executed in both categories, and each category's reliability studies have already been analyzed. It is worth mentioning that the primary required operations to process any arbitrary CNN architecture (e.g., AlexNet, VGG, GoogLeNet, or ResNets) are Convolution, Pooling, Activation Functions, and Fully-connected layers, with varying input and filter sizes. As it is not straightforward to make a direct comparison, some crucial factors are considered, as follows:

6.2.1. Hardware resources

In GPUs, mainstream CNNs rely heavily on dense floating-point matrix multiplication (i.e., GEMM). Approximately 76% of the GPU operations for a CNN model are spent in GEMM-related operations. GPUs' hardware resources are used to execute these operations, including register files, caches, SMs, dispatchers, and schedulers.

In FPGAs, mainstream CNNs rely on hardware resources, such as LUTs, FFs, Carry logic, I/O blocks, and DSP units for MAC operations and Block RAMs as memory elements. Further, configuration bits are used for functioning and routing, where faults are injected.

6.2.2. Floating-point (FxP) vs. fixed-point (FP)

As resource utilization is more significant to FPGAs than GPUs, they tend to use FxP more than FP formats. In contrast, GPUs have the flexibility to use either FP or FP. In general, FP is more extensive in range than FxP format, and hence, a fault in its exponent can lead to a significant change in its value. Therefore, for the same length (i.e., number of bits), FP is more vulnerable than FxP [20]. Considering beam experiments on GPUs and FPGAs, reducing precision (i.e., from double to single to half), hardware resources' utilization is decreased [112]. Thus, the area exposed to radiation is reduced.

As a result, the FIT rate is reduced [112] [121]. This is true even with FxP formats [20]. However, the impact of errors in the Least Significant Bit (LSB) is not the same as errors in the Most Significant Bit (MSB) for the different precisions [106]. With reduced precision, the impact of soft errors is increased. Therefore, FPGAs susceptible to critical errors than GPUs. As can be seen, it is complex to compare these accelerators from a data-format perspective.

6.2.3. Error propagation behaviors

In CNN computations, data propagates from the input layer to the middle (hidden) layers to the output layer. Once an error occurs, it will also follow the same rule. The error is propagated through these different layers.

In GPUs, FC layers, for example, tend to propagate SDC errors, but they do not produce SDC errors by themselves. The reason behind this is that FC layers have highly-parallel (redundant) connectivity, which means the neurons are arranged in the shape of a vector (for both inputs and weights), rather than matrices [129]. Thus, data reusability is much less compared to convolutional layers. Since GPUs are general-purpose accelerators, they have fully spatial architectures (i.e., internally store and exchange data for higher throughput). Single-Instruction Multiple-Threads (SIMT) execution model is used for reducing memory latencies [78]. This denotes that there is no customized hardware in GPUs to execute FC layers; instead, all the available internal resources are assigned to accomplish any task [129]. Therefore, due to redundant connections and the lack of data reuse, FC layers' errors are unlikely to affect the final computations. Note that the FC layers' case confirms the inherent faults tolerance of the DNNs in GPUs, but not all DNN layers. This phenomenon has been reported in [19,102,109].

In FPGAs, FC layers tend to produce SDC errors even more than in convolutional and pooling layers. Since FPGAs are programmable accelerators, they can have both spatial and temporal architectures [129]. Moreover, the implemented design of an FPGA is determined by the bitstream, rather than SIMT, stored in configuration memory [112]. The specific design refers to customized hardware to execute layers, such as FC layers. Thus, the higher the density of the layer, the higher its susceptibility. Therefore, since layers in FPGAs usually rely on customized hardware resources (as listed in Section 6.2.1), FC layers' errors are more likely to affect the final computations than the other layers. This claim has been derived from the findings of [118,119,122].

As can be seen, the propagation process in GPUs is different from the one in FPGAs. Based on the underlying hardware structure of each category, errors propagate to the next level.

6.2.4. Mitigation techniques

In GPUs, although the ECC mitigation technique is a hardware-based solution (for memory elements), the most common approaches to improve the reliability of GPU applications are software-based solutions used to protect either memory or logic resources. That is due to GPUs' predesigned architectures that prevent researchers from accessing low-level layouts, Register-transfer Level (RTL). The proposed software-based techniques include (1) program modules duplication, such as DMR and TMR, which can be selective or full duplication or even triplication of operations [102,108,111]; (2) ABFT for matrix multiplication and Fast Fourier Transforms (FFT) [19,106]. Our review explores that none of the mitigation techniques reported in the literature (in the GPU case) provides solutions for the training phase. All the proposed approaches are for the inference phase only.

In FPGAs, mitigation techniques are usually hardware-based solutions at hardware description language (HDL), representing the RTL. Existing hardware-based strategies are redundancy-based solutions. That is the idea of building identical redundant hardware modules, along with the original module. This redundancy can be Duplication

with Comparison (DWC) or TMR [122]. Nevertheless, as DSPs are embedded processors in FPGA, software-based techniques can also be used [118,119]. Unlike in the GPU case, some solutions have been proposed in the FPGA case, particularly for the training phase, such as fault-aware training [70].

Therefore, from the given comparison factors, it can be concluded that it is straightforward to make point-by-point comparisons between these accelerators from a reliability perspective. However, to decide which accelerator is more resilient, all the four factors, as mentioned above, should be considered, as it is strictly dependent on the underlying architecture.

7. Future challenges and directions

A major challenge in future deep learning is that when its algorithms are implemented on susceptible accelerators. Accordingly, the applicability of DNNs in high-assurance applications is limited by reliability issues that arise from soft errors on DNN accelerators. In this section, we share our views on the challenges that can open trends in the future:

Other DNN architectures: Modern, customized deep learning systems consist of Hybrid DNNs to achieve real-life cognitive tasks. In other words, more than a DNN architecture in the same system. For instance, Parana architecture [39] is composed of CNNs, MLPs, and two others. Although Yin *et al.* proposed a technique (during the design) for addressing only the thermal problem of DRAM, the error resilience capability of such complex systems has not yet been evaluated [39]. Hence, reliability issues in such accelerators' processing and control elements cannot merely be evaluated with these existing techniques. Therefore, novel evaluation and mitigation techniques, which can smoothly fit into these hybrid architectures, are required.

Evaluation tools for DNN error-resilience: Many of the reviewed studies [19,49,85,114], reported that the fault injector or DNN framework that was used to assess the error resilience of DNN components was slow. Thus, it is imperative to optimize the evaluation time for these tools to be more realistic to simulate the reliability impacts accurately. These tools are beneficial for studying the unique characteristics of DNN accelerators/algorithms concerning reliability issues. This can be a valuable research direction for the community.

The rapid movement of the domain: As our review reveals, the evolution of AI models is ongoing. Hence, the availability of hardware to accelerate these computationally intensive models is a notable concern at the design level. As a result, full attention is paid to the throughput optimization of AI accelerators, while the reliability issue is far neglected. Therefore, reliability researchers should fill this gap by providing efficient solutions to each category (i.e., accelerator).

Evaluating different DNN phases: As can be noticed from Tables 3, 5, 7, and 9, most works have evaluated the reliability of DNNs in the inference phase. In contrast, only a few studies have studied on training phase. Moreover, there are two training types: online training (updating the parameters after each training data point) and off-line training (updating the parameters once after the entire training dataset). Researchers may need to determine whether these two types of training leave different impacts from a reliability view.

Size of the dataset: As it helps to ignore small noises, one of the main factors that affect the resilience of DNNs is the dataset used to train the model [113]. Thus, the more training data used to train the model, the higher the accuracy it gives. However, most of the reliability studies found in the literature have studied the reliability of DNNs trained with smaller datasets, such as CIFAR-10 and MNIST. Especially for *FPGA* and *Others* categories (see Tables 7 and 9). Therefore, studying the reliability of DNNs trained on more massive benchmark datasets, such as ImageNet, may have beneficial impacts.

8. Conclusion

DNN technology adds value in many aspects of life, and it is utilized in many different mission-critical systems, which necessarily require a high level of reliability. The computational power in DNN accelerators has shown a clear potential for acceleration, beyond what is possible on CPUs alone. Consequently, they have become the primary hardware for DNN algorithms. However, these accelerators are vulnerable to faults. In particular, DNN accelerators can produce multiple errors from a single fault due to the massive parallel architectures.

Moreover, the technology is shrinking to create more space on a die and low-power circuits. Thus, these accelerators are becoming even more susceptible to soft errors. These effects cannot be ignored in space or even on the ground.

In this paper, the reliability of DNN accelerators regarding transient faults has been reviewed. The study started by reviewing the common claim that "ANNs are inherently tolerant to faults" and showed that ANNs could be prone to transient faults in other cases. This paper provided a taxonomy of the DNN accelerators to shed light on each category and show the amount of attention they gain separately, from a reliability perspective. Then, each category's reliability from the literature studies has been analyzed, and representative works were presented in a uniformed figure. Next, the most commonly used categories have been compared, and factors on DNN accelerators that restrain a fair comparison were highlighted. This review work ended with some notable future challenges and directions that will further require some attention and effort.

Author statement

Younis Ibrahim: Conceived the presented idea, methodology, and the original draft. He kept writing, reviewing and editing the manuscript. He also worked as a coordinator to incorporate the contributions of co-authors into the manuscript.

Haibin Wang: He is the direct supervisor of the whole project. He devised the project and the main conceptual ideas.

Junyang Liu: Besides helping to write and revise the original draft, he also wrote most of the points in the future directions (i.e., Section 7).

Li Chen: Developed the idea of structuring the paper into categories, specifically the DNN accelerators (i.e., Section 5), and wrote its original draft. Besides, he reviewed and edited the manuscript several times.

Paolo Rech: Developed the idea of adding the discussion section (i.e., Section 6) to the manuscript, and wrote its original draft. Moreover, he reviewed and edited the manuscript several times.

Khalid Adam: Formulated the comparison tables and wrote the introduction section (i.e., Section 1). He also verified the analytical methods used for comparisons.

Jinghe Wei and **Gang Guo**: Cooperated to analyze and write the related work (i.e., Section 2.4). Also, **Haibin Wang** encouraged them to investigate the contents of the table (from Table 3 to Table 10). They worked out almost all of the analytical details.

All authors discussed the overall structure and reviewers' comments to be addressed. All of them contributed to the final manuscript by either writing or editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

This work is in part supported by the Fundamental Research Funds for the Central Universities (B200202216) and in part supported by Innovation Foundation of Radiation Application, China Institute of Atomic Energy (KFZC2020010401). The authors thank *Shijie Wen*, *Nihaar Mahatme*, and *Jianwei Han* for their help and feedback regarding DNN accelerators. The authors also acknowledge the valuable comments

and suggestions given by the anonymous reviewers of this paper.

References

[1] IDC, Worldwide Artificial Intelligence Spending Guide. https://www.idc.com/getdoc.jsp?containerId=IDC_P33198, Sep. 2019.

[2] E. Gibney, Google AI algorithm master's ancient game of go, Nature 529 (7587) (2016) 445–451.

[3] W. Liu, Z. Wang, X. Liu, N. Zeng, Y. Liu, F.E. Alsaadi, A survey of deep neural network architectures and their applications, Neurocomputing 234 (2017) 11–26.

[4] D. Yu, L. Deng, Deep learning and its applications to signal and information processing [exploratory DSP], IEEE Signal Process. Mag. 28 (1) (2011) 145–154.

[5] A. Graves, A.-r. Mohamed, G. Hinton, Speech recognition with deep recurrent neural networks, in: 2013 IEEE Int. Conf. Acoust. Speech Signal Process. (icassp), IEEE, 2013, pp. 6645–6649.

[6] T. Young, D. Hazarika, S. Poria, E Cambria, Recent trends in deep learning based natural language processing, IEEE Commun. Intell. Mag. 13 (3) (2018) 55–75.

[7] A. Krizhevsky, I. Sutskever, G.E. Hinton, ImageNet classification with deep convolutional neural networks, Commun. ACM 60 (6) (2017) 84–90.

[8] M. Nielsen, Neural Networks And Deep Learning, Det. press, USA, 2015 [online]. Available: http://neuralnetworksanddeeplearning.com.

[9] D. Silver, et al., Mastering the game of Go with deep neural networks and tree search, Nature 529 (7587) (2016) 484.

[10] D.J. Fagnant, K. Kockelman, Preparing a nation for autonomous vehicles: opportunities, barriers and policy recommendations, Transp. Res. Part A 77 (2015) 167–181.

[11] S.R. Islam, D. Kwak, M.H. Kabir, M. Hossain, K.-S. Kwak, The internet of things for health care: a comprehensive survey, IEEE Access 3 (2015) 678–708.

[12] E. Wyrwas, Proton Testing of nVidia GTX 1050 GPU, Part 2, NASA org., Maryland, USA, Apr. 2018. Rep No. 561.4. [Online].

[13] E. Wyrwas, Body of Knowledge for Graphics Processing Units (GPUs), NASA org., Maryland, USA, TN60884, Nov. 2018. Available at: https://nepp.nasa.gov/files/NEPP-BOK-2018-Wyrwas-GPU-TN60884.

[14] S. Han, et al., EIE: efficient inference engine on compressed deep neural network, in: *ACM SIGARCH Computer Architecture News* 44 (3), ACM, 2016, pp. 243–254.

[15] W. Maass, Noise as a resource for computation and learning in networks of spiking neurons, in Proc. IEEE, Torino, Italy 102 (5) (2014) 860–880.

[16] H.R. Mahdiani, S.M. Fakhraie, C. Lucas, Relaxed fault-tolerant hardware implementation of neural networks in the presence of multiple transient errors, IEEE Trans. Neural Netw. Learn. Syst. 23 (8) (2012) 1215–1228.

[17] Y. LeCun, Y. Bengio, G. Hinton, Deep learning, Nature 521 (2015) 436, 05/27/online.

[18] F. Su, P. Yuan, Y. Wang, C.J.P. Zhang, The superior fault tolerance of artificial neural network training with a fault/noise injection-based genetic, Protein Cell 7 (10) (June 2016) 735–748.

[19] F.F.d. Santos, et al., Analyzing and increasing the reliability of convolutional neural networks on GPUs, IEEE Trans. Rel. 68 (2) (June 2019) 663–677, https://doi.org/10.1109/TR.2018.2878387.

[20] G. Li, et al., Understanding error propagation in deep learning neural network (DNN) accelerators and applications, in: Proc. Int. Conf. High Performance Comput., Netw., Storage Anal., Nov. 2017, pp. 1–11. Denver, Colorado, USA.

[21] W.G. Hatcher, W. Yu, A survey of deep learning: platforms, applications and emerging research trends, IEEE Access 6 (2018) 24411–24432.

[22] C. Torres-Huitzil, B. Girau, Fault and error tolerance in neural networks: a review, IEEE Access 5 (2017) 17322–17341.

[23] Z. You, et al., White Paper on AI Chip Technologies (2018), Beijing Innovation Center Future Chips (ICFC), Beijing, China, 2018.

[24] E.O. Aboagye, G.C. James, R. Kumar, Evaluating the performance of deep neural networks for health decision making, Procedia Computer Science 131 (2018) 866–872.

[25] M.G. Amin, B. Erol, Understanding deep neural networks performance for radar-based human motion recognition, in: 2018 IEEE Radar Conf., 2018, pp. 1461–1465.

[26] O.I. Abiodun, A. Jantan, A.E. Omolara, K.V. Dada, N.A. Mohamed, H.J.H. Arshad, State-of-the-art in artificial neural network applications: a survey, Heliyon 4 (11) (2018) e00938.

[27] A. Rassadin, A. Savchenko, Deep neural networks performance optimization in image recognition, in: Proc. 3rd Int. Conf. Info. Technol. Nano-Technol. (ITNT), 2017. Nizhny Novgorod, Russia.

[28] W. Shang, K. Sohn, D. Almeida, H. Lee, Understanding and improving convolutional neural networks via concatenated rectified linear units, in: 2016 Int. Conf. Mach. Learn., 2016, pp. 2217–2225.

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

[30] J. Reason, Managing the Risks of Organizational Accidents, 1st ed., Routledge, London, U.K., 2016 https://doi.org/10.4324/9781315543543 [online].

[31] A. Dixit, A. Wood, The impact of new technology on soft error rates, in: Int. Reliability Phys. Symp. (IRPS), 2011, pp. 5B. 4.1-5B. 4.7.

[32] D.A.G. Oliveira, et al., Modern GPUs radiation sensitivity evaluation and mitigation through duplication with comparison, IEEE Trans. Nucl. Sci. 61 (6) (2014) 3115–3122.

[33] Alom M. Z., et al., The history began from AlexNet: a comprehensive survey on deep learning approaches, CoRR, vol. abs/180301164, 2018, [online]. Available: https://arxiv.org/abs/18030 1164.

[34] V. Sze, Y. Chen, T. Yang, J.S. Emer, Efficient processing of deep neural networks: a tutorial and survey, Proc. of the IEEE 105 (12) (2017) 2295–2329.

[35] G. Lacey, G.W. Taylor, S. Areibi, Deep learning on fpgas: past, present, and future, in: CoRR, 2016 vol. abs/1602.04283. [online]. Available: https://arxiv.org/abs/1602.04283.

[36] I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, MIT Press, 2016, pp. 1–800.

[37] Y. Bengio, Learning deep architectures for AI, Found. Trends Mach. Learn. 2 (1) (2009) 1–127.

[38] Guo K., Zeng S., Yu J., Wang Y., Yang H., A survey of FPGA based neural network accelerator, CoRR, vol. abs/1712.08934, 2017, [online]. https://arxiv.org/abs/1712.08934.

[39] S. Yin, et al., Parana: a parallel neural architecture considering thermal problem of 3D stacked memory, IEEE Transactions on Parallel and Distributed Systems 30 (1) (2019) 146–160.

[40] A. Azizimazreah, Y. Gu, X. Gu, L. Chen, Tolerating soft errors in deep learning accelerators with reliable on-chip memory designs, IEEE Int. Con. Net, Archit. Storage (NAS) (2018) 1–10.

[41] B. Pourbabaee, M.J. Roshtkhari, K. Khorasani, Deep convolutional neural networks and learning ecg features for screening paroxysmal atrial fibrillation patients, IEEE Trans. Syst., Man, Cybern., Syst. 48 (12) (2017) 2095–2104.

[42] J. Gu, H. Liu, Y. Zhou, X. Wang, DeepProf: performance analysis for deep learning applications via mining GPU execution patterns, in: CoRR, 2017 vol. abs/170703750. [online]. Available: http://arxiv.org/abs/170703750.

[43] M. Shafique, et al., An overview of next-generation architectures for machine learning: Roadmap, opportunities and challenges in the IoT era, proc. Design, Autom. Test in Europe Conf. Exhibi. (DATE) (2018) pp. ans827-832.

[44] Y.-H. Chen, T. Krishna, J.S. Emer, V. Sze, Eyeriss: an energy-efficient reconfigurable accelerator for deep convolutional neural networks, IEEE J. Solid State Circuits 52 (1) (2017) 127–138.

[45] M. Nazemi, G. Pasandi, M. Pedram, NullaNet: training deep neural networks for reduced-memory-access inference, CoRR (2018) vol. abs/1807.08716. [online]. Available: https://arxiv.org/abs/1807.08716.

[46] R. Zhao, W. Luk, X. Niu, H. Shi, H. Wang, Hardware acceleration for machine learning, in: IEEE Comput. Soc. Ann. Symp. VLSI (ISVLSI), 2017, pp. 645–650.

[47] S. Mittal, A survey of FPGA-based accelerators for convolutional neural networks, Neural Comput. & Applic. (2018) 1–31 [online], https://doi.org/10.1007/s00521-018-3761-1.

[48] M.A. Hanif, F. Khalid, R.V.W. Putra, S. Rehman, M. Shafique, Robust Machine learning systems: reliability and security for deep neural networks, in: 2018 IEEE 24th Int. Symp. On-Line Test. Syst. (IOLTS), 2018, pp. 257–260.

[49] B. Fang, K. Pattabiraman, M. Ripeanu, S. Gurumurthi, GPU-Qin: a methodology for evaluating the error resilience of GPGPU applications, in: IEEE Int. Symp. Performance Anal. Syst. Softw. (ISPASS), March 2014, pp. 221–230.

[50] C. Claeys, E. Simoen, Radiation effects in advanced semiconductor materials and devices, in: Springer Sci. Mater. Sci., 1st ed., Springer-Verlag, Berlin, Heidelberg, 2013, pp. 9–52.

[51] S. Rehman, et al., Cross-layer software dependability on unreliable hardware, IEEE Trans. Comput. 65 (1) (2016) 80–94.

[52] P.F. Ramos Vargas, Evaluation of the SEE sensitivity and methodology for error rate prediction of applications implemented in multi-core and many-core processors, in: M.S. Thesis, Dept. Microelectronics, Univ. Greno. Alpes (UGA), Grenoble, France, 2017.

[53] J. Noh, et al., Study of neutron soft error rate (SER) sensitivity: investigation of upset mechanisms by comparative simulation of FinFET and planar MOSFET SRAMs, IEEE Trans. Nucl. Sci. 62 (4) (2015) 1642–1649.

[54] S. Rehman, M. Shafique, J. Henkel, Software program-level reliability optimization for dependable code generation, in: Reliable software for Unreliable Hardware: A Cross Layer Perspective, 1st ed., Springer Int. Publish., Karlsruhe, Germany, 2016, pp. 105–146. Ch5.

[55] S. Mittal, A survey of architectural techniques for managing process variation, ACM Comput. Surv. 48 (4) (2016) 1–29.

[56] D. Wolpert, P. Ampadu, Temperature effects in semiconductors, in: Managing Temperature Effects in Nanoscale Adaptive Systems, Springer New York, USA, New York, NY, 2012, pp. 15–33.

[57] N.N. Mahatme, et al., Comparison of combinational and sequential error rates for a deep submicron process, IEEE Trans. Nucl. Sci. 58 (6) (2011) 2719–2725.

[58] T.S. Nidhin, A. Bhattacharyya, R.P. Behera, T. Jayanthi, K. Velusamy, Understanding radiation effects in SRAM-based field programmable gate arrays for implementing instrumentation and control systems of nuclear power plants, Nucl. Eng. Technol. 49 (8) (2017) 1589–1599.

[59] Y. Ko, R. Jeyapaul, Y. Kim, K. Lee, A. Shrivastava, Protecting caches from soft errors: a microarchitect’s perspective, ACM Trans. Embed. Comput. Syst. 16 (4) (2017) 1–28.

[60] P. Rech, et al., Neutron-induced soft errors in graphic processing units, in: IEEE Radiation Effects Data Workshop, 2012, pp. 1–6.

[61] F. Restrepo-Calle, A. Martínez-Álvarez, S. Cuenca-Asensi, A. Jimeno-Morenilla, Selective SWIFT-R, Electron. Test 29 (6) (2013) 825–838.

[62] P. Rech, et al., Neutron sensitivity and hardening strategies for fast fourier transform on gpus, in: 2013 14th European Conf. Radiation and Its Effects Compon. Syst. (RADECS), 2013, pp. 1–5.

[63] P. Reach, Assessing Neutron-induced Effects on NVIDIA Graphics Processing Units, Science and Technology Facilities Council, 2014 [online]. Available: https://www.isis.stfc.ac.uk/Pages/Assessing-neutr oninduced-effects-on-NVIDIA-Graphics-Processing-Units.aspx.

[64] F. Kastensmidt, P. Rech, Radiation Effects and Fault Tolerance Techniques for FPGAs and GPUs, Springer Int. Publication, Cham, 2016, pp. 3–17. FPGAs Parallel Archit. Aerosp. Appl.

[65] N. Seifert, Radiation-induced soft errors: a chip-level modeling perspective, Found. Trends Electron. Design Autom. 4 (2–3) (2010) 99–221.

[66] D.D.A.G.d. Oliveira, L.L. Pilla, T. Santini, P. Rech, Evaluation and mitigation of radiation-induced soft errors in graphics processing units, IEEE Trans. Comput. 65 (3) (2016) 791–804.

[67] J.J. Zhang, et al., Building Robust Machine Learning Systems: Current Progress, Research Challenges, and Opportunities, Presented at the Proceedings of the 56th Annual Design Automation Conference 2019, Las Vegas, NV, USA, 2019.

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

[69] E. Strohmaier, J. Dongarra, H. Simon, and H. Meuer. (Nov. 2019). The top 500 list, [online]. Available: https://www.top500. Org/lists/2019/11/.

[70] K. Xing, Training for 'Unstable' CNN accelerator: a case study on FPGA, in: CoRR, 2018 vol. abs/181201689.

[71] A. Fonseca, B. Cabral, Prototyping a GPGPU neural network for deep-learning big data analysis, Big Data Research 8 (2017) 50–56.

[72] J. Wu, W. Zhao, Design and realization of WInternet: from net of things to internet of things, ACM Trans. Cyber-Phys. Syst. 1 (1) (Feb. 2017) 1–12. New York, NY, USA.

[73] J.A. Stankovic, Research directions for the internet of things, IEEE Internet Things J. 1 (1) (2014) 3–9.

[74] J. Lin, W. Yu, N. Zhang, X. Yang, H. Zhang, W. Zhao, A survey on internet of things: architecture, enabling technologies, security and privacy, and applications, IEEE Internet Things J. 4 (5) (2017) 1125–1142.

[75] J. Lin, W. Yu, X. Yang, Q. Yang, X. Fu, W. Zhao, A real-time en-route route guidance decision scheme for transportation-based Cyberphysical systems, IEEE Trans. Veh. Technol. 66 (3) (2017) 2551–2566.

[76] N.D. Lane, et al., DeepX: a software accelerator for low-power deep learning inference on mobile devices, in: Proc. of the 15th Int. Conf. Info. Processing in Sensor Networks (IPSN), 2016. Vienna, Austria.

[77] Z. Li, Y. Wang, T. Zhi, T. Chen, A survey of neural network accelerators, Frontiers of Computer Science 11 (5) (2017) 746–761.

[78] Y. Chen, Y. Xie, L. Song, F. Chen, T. Tang, A survey of accelerator architectures for deep neural networks, Engineering 6 (3) (2020) 264–274.

[79] T. Chen, et al., DianNao: a small-footprint high-throughput accelerator for ubiquitous machine-learning, in Proc. 19th int. conf. Archit. support Program. Lang. operat. syst. (S0IGPLAN) 49 (4) (2014) 269–284.

[80] N.P. Jouppi, et al., In-datacenter performance analysis of a tensor processing unit, in: Proc. 44th Ann. Int. Symp. Comput. Archit., (ISCA) 45, 2017, pp. 1–12. New York, NY, USA. no. 2.

[81] https://cloud.google.com/tpu/docs/tpus.

[82] Y. Chen, T. Chen, Z. Xu, N. Sun, O. Temam, DianNao family: energy-efficient hardware accelerators for machine learning, Commun. ACM 59 (11) (2016) 105–112.

[83] S. Zhang, et al., Cambricon-X: an accelerator for sparse neural networks, in: 2016 49th Ann. IEEE/ACM Int. Symp. Microarchitecture (MICRO), 2016, pp. 1–12.

[84] H. Kwon, A. Samajdar, T. Krishna, MAERI: enabling flexible dataflow mapping over DNN accelerators via reconfigurable interconnect, in: Proc. 23rd Int. Conf. Archit. Support Program. Lang. and Op. Syst. 53, 2018, pp. 461–475, no. 2.

[85] J. Zhang, K. Rangineni, Z. Ghodsi, S. Garg, Thundervolt: enabling aggressive voltage underscaling and timing error resilience for energy efficient deep learning accelerators, in: Proc. 55th Ann. Design Autom. Conf., San Francisco, California, 2018.

[86] P. Pandey, P. Basu, K. Chakraborty, S. Roy, GreenTPU: improving timing error resilience of a near-threshold tensor processing unit, in: 2019 56th ACM/IEEE Design Automation Conference (DAC), 2019, pp. 1–6.

[87] T.V. Huynh, Deep neural network accelerator based on FPGA, in: 2017 4th NAFOSTED Conference Info. Comput. Sci., 2017, pp. 254–257.

[88] C. Farabet, B. Martini, P. Akselrod, S. Talay, Y. LeCun, E. Culurciello, Hardware accelerated convolutional neural networks for synthetic vision systems, in: 2010 Proc. IEEE Int. Symp. Circuits Syst. (ISCAS), 2010, pp. 257–260.

[89] L. Du, et al., A reconfigurable streaming deep convolutional neural network accelerator for internet of things, IEEE Trans. Circuits Syst. 65 (1) (2018) 198–208.

[90] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, J. Cong, Optimizing FPGA-Based Accelerator Design for Deep Convolutional Neural Networks, ACM/SIGDA Int. Symp, Field-Programmable Gate Arrays, Monterey, California, USA, 2015.

[91] L. Lu, Y. Liang, SpWA: an efficient sparse winograd convolutional neural networks accelerator on FPGAs, in: Proc. 55th Ann. Design Autom. Conf., San Francisco, California, 2018.

[92] J.H. Ko, B. Mudassar, T. Na, S. Mukhopadhyay, Design of an energy-efficient accelerator for training of convolutional neural networks using frequency-domain computation, in: Proc. 54th Ann. Design Autom. Conf., Austin, TX, USA, 2017.

[93] I.d.C. Lopes, Convolutional neural network reliability on an apsoc platform – a traffic sign recognition case study, in: M.S. Thesis, Instit. Info. Technol., UFRGS, Porto Alegre, Brazil, 2017.

[94] P.A. Penz, R. Wiggins, Digital signal processor accelerators for neural network simulations, in: AIP Conference Proceedings 151, AIP, 1986, pp. 345–355, no. 1.

[95] Ignatov A., et al., AI benchmark: running deep neural networks on android smartphones, CoRR, vol. abs/181001109, 2018, [online]. Available: http://arxiv.org/abs/181001109.

[96] L. Codrescu, et al., Hexagon DSP: an architecture optimized for Mobile multimedia and communications, IEEE Micro 34 (2) (2014) 34–43.

[97] P. Desai, Enabling embedded vision neural network DSPs, in: Cadence Design Systems, Jan. 2017 [online]. Available: https://ip.cadence.com/uploads/1208/TIP_WP_8611_Vision_C5_FINAL-pdf.

[98] W.M.W. Hwu, Chapter 1 - introduction, in: W.-m.W. Hwu, 2nd ed. (Eds.), Heterogeneous System Architecture, Morgan Kaufmann, Boston, 2016, pp. 1–5.

[99] H. Kim, et al., A configurable and low-power mixed signal SoC for portable ECG monitoring applications, IEEE Trans. Biomed. Circuits Syst. 8 (2) (April 2014) 257–267.

[100] M.A. Neggaz, I. Alouani, P.R. Lorenzo, S. Niar, A reliability study on CNNs for critical embedded systems, in: 2018 IEEE 36th International Conference on Computer Design (ICCD), 2018, pp. 476–479.

[101] A.P. Arechiga, A.J. Michaels, The effect of weight errors on neural networks, in: in 2018 IEEE 8th Annual Computing and Communication Workshop and Conference (CCWC), 2018, pp. 190–196.

[102] Y. Ibrahim, et al., Soft error resilience of deep residual networks for object recognition, IEEE Access 8 (2020) 19490–19503.

[103] P. Rech, C. Aguiar, C. Frost, L. Carro, "Neutron-Induced Multiple Output Errors: A Reality on GPUs," 1st MEDIAN Workshop, median-project.eu, 2012.

[104] P. Rech, C. Aguiar, C. Frost, L. Carro, An efficient and experimentally tuned software-based hardening strategy for matrix multiplication on GPUs, IEEE Trans. Nucl. Sci. 60 (4) (2013) 2797–2804.

[105] C. Braun, S. Halder, H. Wunderlich, A-ABFT: Autonomous Algorithm-Based Fault Tolerance for Matrix Multiplications on Graphics Processing Units, Proc. 44th Ann. IEEE/IFIP Int. Conf. Dependable Syst. Netw. (2014) 443–454.

[106] F.F.d. Santos, L. Draghetti, L. Weigel, L. Carro, P. Navaux, P. Rech, Evaluation and mitigation of soft-errors in neural network-based object detection in three GPU architectures, in: 2017 47th Ann. IEEE/IFIP Int. Conf. Dependable Sys. and Netw. (DSN-W), IEEE, June 2017, pp. 169–176.

[107] K-H. Huang, I.A. Abraham, Algorithm-based fault tolerance for matrix operations, IEEE Trans. Comput. C-33 (6) (1984) 518–528.

[108] Y. Ibrahim, H. Wang, K. Adam, Analyzing the reliability of convolutional neural networks on GPUs: GoogLeNet as a case study, in: 2020-Intern. Conf. Comp. Info. Tech. (ICCIT-1441), University of Tabuk, Tabuk, KSA, Sep 2020 (in press).

[109] C. Lunardi, F. Previlon, D. Kaeli, P. Rech, On the efficacy of ECC and the benefits of FinFET transistor layout for GPU reliability, IEEE Trans. Nucl. Sci. 65 (8) (2018) 1843–1850.

[110] A.P. Arechiga, A.J. Michaels, The robustness of modern deep learning architectures against single event upset errors, in: 2018 IEEE High Performance Extreme Computing Conf. (HPEC), 2018, pp. 1–6.

[111] J. Wei, et al., Analyzing the impact of soft errors in VGG networks implemented on GPUs, Microelectron. Reliab. 110 (2020) 113648.

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

[113] X. Jiao, M. Luo, J. Lin, R.K. Gupta, An assessment of vulnerability of hardware neural networks to dynamic voltage and temperature variations, in: 2017 IEEE/ACM Int. Conf. Computer-Aided Design (ICCAD), 2017, pp. 945–950.

[114] Choi W., Shin D., Park J. , Ghosh S., Sensitivity based error resilient techniques for energy efficient deep neural network accelerators, Proceedings of the 56th Annual Design Automation Conference 2019, Las Vegas, NV, USA.

[115] M. Lee, K. Hwang, W. Sung, Fault tolerance analysis of digital feed-forward deep neural networks, in: Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE Int. Conf. on, 2014, pp. 5031–5035.

[116] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, Dropout: a simple way to prevent neural networks from overfitting, J. Mach. Learn. Res. 15 (1) (1929-1958) 2014.

[117] J.A. Clemente, et al., Hardware implementation of a fault-tolerant Hopfield neural network on FPGAs, Neurocomputing 171 (2016) 1606–1609.

[118] I.C. Lopes, F.L. Kastensmidt, A.A. Susin, SEU susceptibility analysis of a feedforward neural network implemented in a SRAM-based FPGA, in: 2017 18th IEEE Latin American Test Symp. (LATS), 2017, pp. 1–6.

[119] I.C. Lopes, F. Benevenuti, F.L. Kastensmidt, A.A. Susin, P. Rech, Reliability analysis on case-study traffic sign convolutional neural network on APSoC, in: 2018 IEEE 19th Latin-American Test Symposium (LATS), 2018, pp. 1–6.

[120] Q. Wang, W. Shi, P.M. Atkinson, Z. Li, Land cover change detection at subpixel resolution with a Hopfield neural network, IEEE J. Sel. Topics Appl. Earth Observ. 8 (3) (2015) 1339–1352.

[121] N. Khoshavi, C. Broyles, Y. Bi, A Survey on Impact of Transient Faults on BNNInference Accelerators, 2020 abs/2020arXiv200405915K, CoRR.

[122] F. Libano, et al., Selective hardening for neural networks in FPGAs, in: IEEE Trans. Nucl. Sci., 2018, pp. 1–1.

[123] C. Schorn, A. Guntoro, G. Ascheid, Accurate neuron resilience prediction for a flexible reliability management in neural network accelerators, in: 2018 Design, Automation & Test in Europe Conference & Exhibition (DATE), 2018, pp. 979–984.

[124] J. Marques, J. Andrade, G. Falcao, Unreliable memory operation on a convolutional neural network processor, in: 2017 IEEE Int. Workshop Signal. Process. Syst. (SiPS), 2017, pp. 1–6.

[125] L.-H. Hoang, M.A. Hanif, M. Shafique, FT-ClipAct: Resilience Analysis of DeepNeural Networks and Improving their Fault Tolerance using Clipped Activation, The 23rd Design,Automation and Test in Europe (DATE 2020), in: CoRR, 2019.

[126] Y. Liu, L. Wei, B. Luo, Q. Xu, Fault injection attack on deep neural network, in: 2017 IEEE/ACM Int. Conf. Computer-Aided Design (ICCAD), 2017, pp. 131–138.

[127] A. Bosio, P. Bernardi, A. Ruospo, E. Sanchez, A reliability analysis of a deep neural network, in: 2019 IEEE Latin American Test Symposium (LATS), 2019, pp. 1–6.
[128] J. Breier, X. Hou, D. Jap, L. Ma, S. Bhasin, Y. Liu, Practical fault attack on deep neural networks, in: Proc. of the ACM SIGSAC Conf. Comput. Comm. Security,, ACM, Toronto, Canada, October 2018, pp. 2204–2206.
[129] M. Capra, B. Bussolino, A. Marchisio, M. Shafique, G. Masera, M. Martina, An updated survey of efficient hardware architectures for accelerating deep convolutional neural networks, Future Internet 12 (7) (Jul. 2020) 113.