Review article

Resilience of deep learning applications: A systematic literature review of analysis and hardening techniques

Cristiana Bolchini *, Luca Cassano, Antonio Miele

Politecnico di Milano, Dip. Elettronica, Informazione e Bioingegneria, P.zza L. da Vinci, 32, Milan, 20133, Italy

ARTICLE INFO

ABSTRACT

Machine Learning (ML) is currently being exploited in numerous applications, being one of the most effective Artificial Intelligence (AI) technologies used in diverse fields, such as vision, autonomous systems, and the like. The trend motivated a significant amount of contributions to the analysis and design of ML applications against faults affecting the underlying hardware. The authors investigate the existing body of knowledge on Deep Learning (among ML techniques) resilience against hardware faults systematically through a thoughtful review in which the strengths and weaknesses of this literature stream are presented clearly and then future avenues of research are set out. The review reports 85 scientific articles published between January 2019 and March 2024, after carefully analysing 222 contributions (from an initial screening of eligible 244 publications). The authors adopt a classifying framework to interpret and highlight research similarities and peculiarities, based on several parameters, starting from the main scope of the work, the adopted fault and error models, to their reproducibility. This framework allows for a comparison of the different solutions and the identification of possible synergies. Furthermore, suggestions concerning the future direction of research are proposed in the form of open challenges to be addressed.

Contents

---

* Corresponding author.
E-mail addresses: cristiana.bolchini@polimi.it (C. Bolchini), luca.cassano@polimi.it (L. Cassano), antonio.miele@polimi.it (A. Miele).

1. Introduction

The widespread adoption of Machine Learning (ML) in safety/mission-critical systems motivated a great attention towards the resilience of such complex systems against the occurrence of faults in the underlying hardware. Among all ML techniques, Deep Learning (DL) is the one that the research community is mainly focusing its attention on, also in terms of reliability issues. In fact, DL is widely used for vision and perception functionalities, which are particularly relevant for implementing human-assisting tasks (e.g., advanced driver-assistance systems), and it represents the enabling technology for autonomous behaviours (e.g., unmanned aerial vehicles or rovers). DL consists of a set of specific Artificial Neural Network (ANN) models where multiple layers of processing are used to extract progressively higher level features and information from raw data, such as images taken from cameras [1]. Adopting the classification proposed in [2], faults can occur in (i) input data, (ii) software, and (iii) hardware , possibly causing the application to behave differently from what is expected. Faults on *input data* may derive from defective/broken sensors and devices, noise, as well as from adversarial attacks. Faults in *software* usually originate from bugs or aggressive implementations. Finally, faults in *hardware* may be caused by radiation, voltage over-scaling, and ageing or in-field permanent stuck-at failures [3]. When addressing hardware faults, the underlying assumption is that the DL application has been designed and implemented to achieve the best performance (in terms of accuracy of the prediction tasks) with respect to requirements and constraints, and the input data is genuine. In this work we focus on *hardware faults* and we investigate analysis and design methods and tools to evaluate and possibly improve the *reliability* of DL algorithms and applications against this source of failure. We adopt the taxonomy of dependability attributes defined in [4], focusing on internal hardware faults that impact system functionality. We use the term *resilience* (a synonym of *fault tolerance*) because most analysis and hardening techniques target the ability to mitigate the effects of the faults. To avoid confusion, we do not adopt the term *reliability*, which, in the context of DL and Artificial Intelligence (AI) in general often means the ability to produce correct results, rather than ensuring correct service even when faults occur. Additionally, we also include approaches dealing with *robustness*, when they address resilience against internal hardware faults and not to the algorithm correctness or the ability to function correctly even when facing external faults, such as adversarial attacks in a security context. Moreover, although the design and training processes have an impact on the performance of the final implementation resilience, such facets are here considered only when they are associated with the possibility to mitigate hardware fault effects.

On this topic the body of knowledge is quite rich, and a very detailed analysis has been presented in [5], where the author introduces a comprehensive and extensive synthesis of analysis and hardening methods against faults affecting hardware platforms running ANN applications. The contribution details the various adopted fault models, the fault simulation/injection and emulation strategies presented in literature at that time, as well as the proposed solutions to make the AI resilient against the analysed faults/errors. A similar contribution is given by [6], where the authors analyse how faults in Deep Neural Network (DNN) accelerators such as Graphic Processing Units (GPUs) and Field Programmable Gate Arrays (FPGAs), affect the executed application. The analysis framework takes into account the different sources of faults and possible fault locations, and a few final considerations mention hardening solutions. The most recent contribution reporting part of the body of work on DL resilience is [7], analysing some recent research and results focused on resilience assessment, covering contributions before 2023. The authors introduce the context and detail the fault analysis strategies and methods adopted when dealing with DL applications, reporting some novel solutions. These papers serve not only as a reference to prominent research up to that
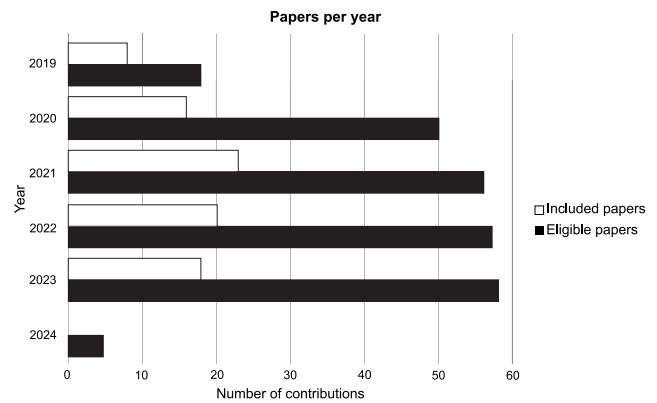

Fig. 1. Number of contributions on the domain of interest per year, in the considered time frame.

time instant, but also provide a concise explanation of the various existing techniques. To complete the scenario overview, three recent contributions that briefly discuss the state of the art and focus also on possible research challenges and opportunities are the works presented in [2,8,9], sometimes embracing also security-related considerations.

As Fig. 1 shows, the community is very active and the contributions of the last four years introduce new relevant elements and insights, motivating, in our opinion, a new review. Differently from previous surveys in this field, though, this contribution aims at analysing also the scientific research community active on the topic to gather insights, challenges and observed trends that go beyond the technical aspects, with a different perspective.

Given the breadth of the domain and the many different facets, we define a boundary based on (i) the time window of the publication, selecting only those included in the Jan. 2019 - Mar. 2024 window, to better frame the discussion; (ii) the *adopted fault model*, by including only contributions that cover transient and permanent faults; (iii) the *DL algorithm*, by excluding works that strictly depend on the specific ANN architecture (e.g., spiking neural networks, vision transformers), so that the presented solutions can be broadly adopted; and (iv) the hardware platform running the application, by including CPUs and hardware accelerators, such as GPUs and FPGAs.

The rest of the paper is organised as follows (see Fig. 2). Section 2 introduces the adopted search methodology aligned with the boundary of the domain previously mentioned, and the classification framework defined to analyse the available contributions. Section 3 reports the various contributions, characterised according to the defined analysis framework, briefly summarising the most relevant aspects. Section 4 draws some considerations on the overall state of the art, highlighting open challenges and opportunities, while Section 5 concludes the paper.

2. Methodology

Before presenting the proposed classification framework and the selected contributions, we here introduce the adopted search and selection process.

2.1. Research design

This study aims at conducting a systematic literature review to explore the current state of the art in the design and analysis of resilient DL applications against hardware faults and to observe the present research trends in this context. The purpose is to get an up-to-date overview of the available solutions, also identifying the open challenges and possible opportunities in the field. To this end we performed a thorough search and designed an analysis framework to classify the numerous contributions.
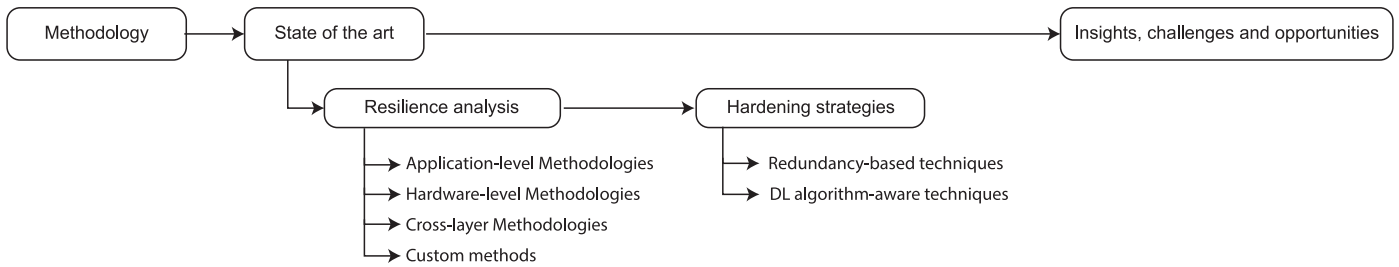
Fig. 2. Paper organisation.

Table 1
The implemented search strings.

1	(''soft error'' OR ''resilien*'' OR ''dependab*'' OR ''fault toleran*'' OR ''reliab*'' OR robust OR ''harden*'')
2	(''Deep Learning'' OR DL OR ''Machine Learning'' OR ML)
3	(''Convolutional Neural Network'' OR ''Convolutional Neural Network'' OR CNN OR ''Deep Neural Network'' OR DNN)
4	(''soft error'' OR fault OR ''Single Event Upset'' OR SEU)

Table 2
The selected databases and formulated search strings.

Database	Search string
Scopus	TITLE-ABS-KEY ((''Resilien*'' OR ''Fault toleran*'' OR ''Robust*'' OR ''Dependab*'' OR ''Reliab*'') AND (''CNN'' OR ''DNN'' OR ml OR ''Convolutional Neural Network'' OR ''Deep Neural Network'') AND (''Soft error'' OR seu OR fault)) AND PUBYEAR > 2018 AND (EXCLUDE (SUBJAREA,''PHYS'') OR EXCLUDE (SUBJAREA,''MATH'') OR EXCLUDE (SUBJAREA,''ENER'') OR EXCLUDE (SUBJAREA,''MATE'') OR EXCLUDE (SUBJAREA,''DECI'') OR EXCLUDE (SUBJAREA,''CHEM'') OR EXCLUDE (SUBJAREA,''EART'') OR EXCLUDE (SUBJAREA,''BIOC'') OR EXCLUDE (SUBJAREA,''CENG'') OR EXCLUDE (SUBJAREA,''ENVI'') OR EXCLUDE (SUBJAREA,''MULT'') OR EXCLUDE (SUBJAREA,''SOCI'') OR EXCLUDE (SUBJAREA,''NEUR'') OR EXCLUDE (SUBJAREA,''MEDI'') OR EXCLUDE (SUBJAREA,''BUSI'') OR EXCLUDE (SUBJAREA,''HEAL'') OR EXCLUDE (SUBJAREA,''AGRI'')) AND (EXCLUDE (LANGUAGE,''Chinese'') OR EXCLUDE (LANGUAGE,''French'')) AND (EXCLUDE (EXACTKEYWORD,''Diagnos''))
WOS	((TI=(''Resilien*'' OR ''Fault toleran*'' OR ''Robust*'' OR ''Dependab*'' OR ''Reliab*'') OR AK=(''Resilien*'' OR ''Fault toleran*'' OR ''Robust*'' OR ''Dependab*'' OR ''Reliab*'')) AND (TI=(''CNN'' OR ''DNN'' OR ML OR ''Convolutional Neural Network'' OR ''Deep Neural Network'' OR ML OR ''Machine Learning'' OR DL OR ''Deep Learning'') OR AK=(''CNN'' OR ''DNN'' OR ML OR ''Convolutional Neural Network'' OR ''Deep Neural Network'' OR ML OR ''Machine Learning'' OR DL OR ''Deep Learning'')) AND (TI=(''Soft error'' OR SEU OR fault) OR AK=(''Soft error'' OR SEU OR fault)))

2.2. Research method

To gather the contributions within the area of interest, we started from Scopus and Web of Science to collect papers that appeared in renowned venues (both journals and conferences), delimiting the time span between January 2019 and March 2024, and excluding all topic areas and keywords that would surely lead to not relevant publications. Tables 1 and 2 report the desired search strings and the actual ones in the mentioned repositories.

The searches returned a very high number of contributions (2163) and we adopted the process reported in Fig. 3 to filter out clearly unrelated contributions and to include other ones through reference mining and snowballing also on other search engines. More precisely, we initially excluded contributions (filter ①) based on the title, the abstract and the keywords. Indeed many results referred to the use of ML/DL for resilience and diagnosis, sometimes applied to out-of-scope contexts (e.g., power/transmission lines or not ML/DL applications). Through snowballing and reference mining we added new contributions, leading to a batch of 244 papers we read. Further filtering took place (filter ②) based on the strength of the contribution (paper length equal or greater than 4 pages and/or venue) and the existence of a subsequent more mature/complete publication (222 papers, dubbed *eligible*). Finally, we selected a set of 85 papers considered as the review sources (filter ③) to have contributions presenting solutions of general validity, possibly excluding too specific scenarios or narrow case study.
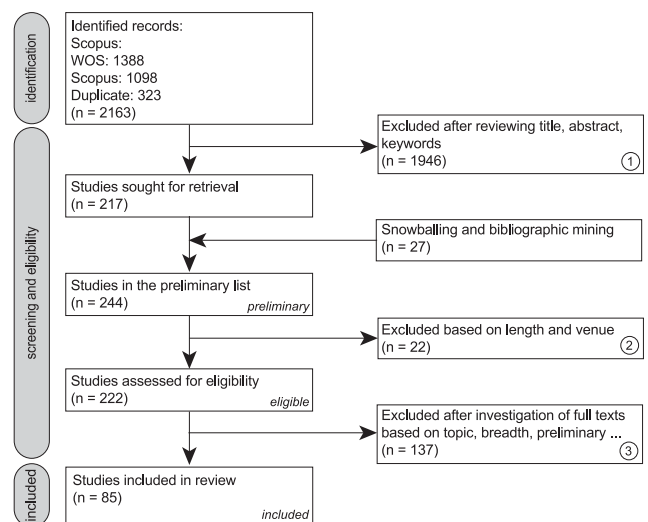

Fig. 3. Flow diagram presenting the retrieval and screening process of the literature following the Preferred Reporting Items for Systematic Reviews and Meta-Analyses (PRISMA) process.

Table 3
Search methodology details.

Keywords:	soft error, resilience, dependable, fault tolerance, reliable, robust AND DL, CNNs, DNNs
Repositories:	IEEE, ACM, Elsevier, Springer
Search engines:	Google scholar, Semantic scholar, Scopus, Web of Science, lens.org, DBLP
Publication years:	January 2019 – March 2024
Search outcome:	2163
analysed contributions:	222
Reported contributions:	85
Novel technical contributions:	76

scope	abstraction level	hardware platform	fault model	error model	ML framework	tool support	reproducibility
▪ analysis ▪ hardening	▪ device ▪ logic ▪ RTL ▪ microarchitecture ▪ algorithm ▪ application	▪ GPU ▪ CPU ▪ custom hardware ▪ platform-agnostic	▪ permanent / stuck-at ▪ transient / SEU ▪ permanent / functional ▪ transient / functional	▪ register/memory ▪ parameters ▪ data value ▪ neuron output ▪ layer output	▪ TensorFlow ▪ PyTorch ▪ Keras ▪ Darknet ▪ Caffee ▪ TensorRT ▪ N2D2 ▪ FINN	▪ yes ▪ no	▪ yes ▪ no

Fig. 4. The primary axes of the adopted classification framework, with a few sample values.

6 out of the 85 documents are surveys or position papers, 2 are tools not specific to resilience analysis/hardening, thus we actually analyse and classify 76 papers, presenting novel contributions on the topic of interest. The characteristics of the search method as well as the outcomes are summarised in Table 3. The spreadsheet file with all the raw bibliographic data analysed during this systematic literature review process can be downloaded from https://github.com/D4De/dl_resilience_survey.

2.3. Classification framework

We have defined an analysis framework to carry out a rigorous classification of the selected papers. Fig. 4 sketches the primary axes of this analysis framework, being a set of relevant aspects for the considered topic, i.e., system's *resilience*, and the referred application scenario, i.e., *DL applications*. A brief description of all the considered aspects, further synthesised in Table 4, is given in the following paragraphs.

Scope. The primary element adopted to organise the contributions is the main goal of the presented solutions, broadly aggregated into two main classes; analysis and hardening methods. Contributions devoted to the development of techniques and tools to evaluate the resilience of the application against hardware faults belong to the *analysis methods* class, those that present new approaches to enhance the capabilities of the system to detect and mitigate the effects of hardware faults are included in the *hardening methods* class. Indeed some contributions introduce innovative strategies to evaluate resilience and exploit such information to tailor a hardening method; these methods have been included in the category they provide the strongest contribution. Finally, some publications explore the application of either traditional or recent methods to specific study cases, reporting outcomes and limitations, and experiences others might benefit from; we classified them in the *custom methods* group.

Abstraction level. Common to many fields of the digital systems' design area, approaches work at different levels of abstraction, within the entire hardware/software stack from the technological level to the application one. Moreover, multiple other aspects are highly dependent on the adopted abstraction level, therefore we prioritised it and identified the following six values, based on the main system element the proposed methods work on:

- *Device* – physical device,

- *Logic* – logic netlist.
- *Register-Transfer Level (RTL)* – architectural description at RTL level,
- *Microarchitecture* – hardware schema described in the Instruction Set Architecture (ISA),
- *Algorithm* – software elements within the implementation of the single DL operators,
- *Application* – software elements in the dataflow graph of the DL model.

Hardware platform. The type of misbehaviour caused by faults affecting the hardware in the application execution is highly dependent on the underlying platform. Therefore, another key aspect in the analysis framework is the hardware platform where the DL application is executed. Frequently adopted platforms are the GPUs and custom hardware accelerators, implemented on FPGA or ASIC; the CPU is used only in a few contributions, while the Tensor Processing Unit (TPU) is increasingly receiving interest (e.g., the NVDLA platform [10]). As we will see, some contributions, especially when acting at the application abstraction level, will not consider any specific hardware, thus being platform independent or *platform-agnostic*.

Fault model. Every reliability study has a fundamental element driving the discussion, that is the source of the anomalous behaviour the proposed approach is addressing. The reference abstraction level for the definition of the fault model is the logic/architecture one, where literature defines permanent models, such as the stuck-at faults, and transient ones, such as Single Event Upset (SEU). Some of the proposed methods work at the application level, not referring to a specific hardware platform; it is therefore not possible to identify the mechanisms causing the anomaly in the expected values/behaviour. For these contributions we added a *functional* fault model, transient and/or permanent, according to the authors' specification.

Since many of the analysed works act at a higher abstraction level, fault models are generally abstracted to derive the corresponding error models.

Error model. An error model describes the effects of the considered fault model at the selected abstraction level, and it affects one of the elements of the abstraction level. When working at device level or RTL, the relationship between fault and error are quite straightforward, when moving to higher abstraction levels, such a relationship is sometimes part of the contribution (for resilience analysis methods),

sometimes omitted. Indeed, when adopting a functional fault model as previously discussed, fault and error models tend to be a unique element. Nevertheless, the error model is characterised by the specific corrupted *location* which, once more, depends on the abstraction level. At device, RTL and microarchitectural levels, fault locations typically include registers and memory elements storing processed data and the DL model weights. At a higher levels of abstraction, error locations may also include parameters as single weights and bias constants, or data values, and even more complex data structures such as the outputs of the various neurons or the intermediate tensors produced by the layers in the DL model. Therefore, we identify the following corrupted values: (i) register/memory element, (ii) parameter, (iii) data value, (iv) neuron output, (v) layer output.

ML framework. The design of DL applications is generally performed in specific ML frameworks guiding and easing this type of activity by providing ML operators already implemented, and algorithms to automate the training and testing of the models. TensorFlow and PyTorch are examples of such frameworks. Several reliability studies and tools are developed and tailored for the specific ML framework, to enable the integration of the resilience activity in the design flow and to exploit the elements it provides. This axis of the classification collects this aspect when specific to the proposed solution.

Tool support. The availability of open-source tools is indeed beneficial to the entire scientific community, to foster further developments as well as fair comparisons. Our framework includes also this aspect, to indicate whether the authors make available the developed software to perform the presented analysis/hardening solutions. The list of urls of the available software is reported in the last part of the paper.

Reproducibility. Similarly to the previous aspect, we deemed relevant to be able to reproduce the outcomes of the study, in the future, to present a comparative analysis for supporting new solutions. To this end, we marked entries with a positive answer when the software is available or the adopted method is discussed in details allowing for it to be replicated.

Analysis and hardening approaches can be further characterised with respect to the specific proposed solutions, namely the dependability attribute, injection method and analysis output in the former approaches, target outcome, hardening technique and hardening strategy in the latter. They are discussed in the following and summarised in Tables 5 and 6, respectively.

Dependability attribute. The various analysis approaches may focus on the evaluation of different attributes falling under the umbrella of the dependability; generally, works use to quantitatively analyse a reliability metric. In the considered scenario, further works analyse the vulnerability to faults of the various layers, operators or parameters composing the DL model. Thus, the considered *dependability attribute* is another characterising aspect for the reviewed papers, that includes in our work the following values[1]:

- reliability - the continuity of correct service [4],
- safety - the absence of catastrophic consequences on the user(s) and the environment [4], and
- vulnerability factor - the measure the likelihood that a fault in a hardware component will lead to an observable error at the system level [11].

Injection method. The vast majority of the analysed contributions rely on fault/error injection methods to perform the resilience analysis, and the specific one depends on the abstraction level of the work. Here we list the following values to include all included studies:

- Radiation tests – the final system is irradiated with nuclear particles.
- Fault emulation – faults are emulated on the target hardware platform.
- Error simulation – processed data are corrupted during the execution of the software running non-necessarily on the target platform.

Analysis output. When performing a resilience analysis, two main types of outcomes are typically reported: (i) a quantitative measure adopted as a figure of merit, or (ii) a qualitative evaluation of the solution, based on various considerations. Sometimes, based on the analysis results, also guidelines for hardening the system are provided, often targeting the mitigation of the most susceptible elements in the analysed DL model. In the set of selected papers, all contributions on analysis methods report a quantitative output and eventually some hardening guidelines, that is what we report in the final synthesis.

Reliability property. Hardening approaches can be classified w.r.t. the reliability property the final system will exhibit, that in the present set of studies is either fault detection or fault tolerance.

Hardening technique. In the DL scenario, as in other contexts, often the hardening process relies on redundancy-based techniques. Some approaches adopt the classical techniques, such as Duplication with Comparison (DWC) possibly coupled with re-execution, Triple Modular Redundancy (TMR), N-Modular Redundancy (NMR) and Error Correcting Code (ECC). Other works apply Algorithm-Based Fault Tolerance (ABFT) or Algorithm-Based Error Detection (ABED) techniques within the single DL operator, being the algorithm generally based on matrix multiplications. Finally, a last class of works exploits specific characteristics of the DL models, such as the adoption of fault-aware training strategies to exploit the intrinsic information redundancy in DL models to deal with the effects of a fault.

Hardening strategy. Finally, various strategies can be adopted aimed at reducing the overhead of hardening redundancies. In particular, apart from the application of a technique to the entire application, *selective* hardening is used to protect only the most critical portion of the system and *approximation* strategies can be used to limit the overheads of redundant application replicas. Finally, some solutions design *specific* versions of DL operators to obtain at their output a resilient result.

A detailed list of the collected values for each one of the framework axes is reported in Tables 4 and 5. Indeed the framework can be extended in the future to include new relevant axes, and the values can always be incremented to cover newly reviewed solutions.

3. The state of the art

We classified the reviewed papers primarily based on their main contribution, organising them into *analysis* methods and *hardening* ones; studies tackling both aspects have been included in the group associated with the predominant contribution.

3.1. Resilience analysis

This first class of works includes approaches for the analysis of the resilience of digital systems running DL applications w.r.t. the occurrence of faults. To further characterise them, we consider the abstraction level they work at, namely *application-level*, *hardware-level* or *cross-layer*.

Application-level methodologies aim at analysing the resilience of the DL engine ignoring the underlying hardware platform. Therefore,

---

[1] We referred to the term used by the authors of the contribution, merging including fault tolerance in the reliability class, as motivated in the introduction.

Table 4

Taxonomy axes.

Classification Axis	Description	Values
Scope	The focus of the approach	Analysis (A), Hardening (H) or both (B)
Abstraction level	The abstraction level methodologies/solutions work at	Device (DEV), Logic (LOG), RTL, Microarchitectural (ISA), Algorithm (ALG), Application (APP)
Architectural platform	The hardware where the application is executed. Affects the fault/error model, the abstraction level, etc.	CPU, GPU, TPU, FPGA, or any (in case of high abstraction-level methodologies)
Fault model	The source of the anomalous behaviour	Stuck-at (SA), Single Event Upset (SEU), permanent functional (PFunc), transient functional (TFunc)
Error model	The effects of the fault at the selected abstraction level, identifying the corrupted element	register/memory element (REG), parameter (P), data value (DV), neuron output (NO), layer output (LO)
ML Framework	The exploited software ML framework, if specified	TensorFlow (TF, [12]), PyTorch (PT, [13]), Keras (KE, [14]), Darknet (DK, [15]), Caffe (CA, [16]), TensorRT (TR, [17]), cuDNN (cu [18]), N2D2 (ND [19]), FINN (FI [20]), CMSIS-NN (CM [21]), CMix-NN (CN [22])
Tool support	Tools released	Yes/No
Reproducibility	The possibility to replicate/compare against	Yes/No

Table 5

Analysis studies: further classification.

Classification Axis	Description	Values
Dependability attribute	Attribute of interest	Reliability (Re), Safety (Sa), Vulnerability factor (VF)
Injection method	Fault injection method	Radiation (Ra), Emulation (Em), Simulation (Si), Analytical (An)
Output	Kind/type of output of the analysis	Quantitative metrics (QM), Hardening guidelines (HG)

Table 6

Hardening studies: further classification.

Classification Axis	Description	Values
Reliability property	Aim of the hardening	Fault detection (FD), Fault tolerance (FT)
Strategy	Type of action	Full, Selective (Sel), Specific (Spec), Approximated (Ax)
Technique	Adopted technique	Duplication with Comparison (DWC), Triple Modular Redundancy (TMR), N-Modular Redundancy (NMR), DWC + Re-Execution (D+R) Algorithm-Based Fault Tolerance (ABFT), Algorithm-Based Error Detection (ABED), Error Correcting Code (ECC), Checkpointing (CHK), DL algorithm-aware (DL)

such works consider the engine at the dataflow graph-level and study the impact of errors corrupting the weights of the model, the output of the operators, or the variables within the operators' execution. The advantages of these methodologies are (i) the possibility to apply them early in the design process, as soon as the DL engine has been designed and trained; (ii) easiness of the deployment (no hardware prototypes and/or instrumentation is required); and (iii) the opportunity to work directly on the actual DL engine that will then be used. On the other hand, the solutions may suffer from poor accuracy because of the abstract adopted error models. It is vital for application-level analyses to properly work that the adopted error models actually capture the effects that the faults in the hardware platform cause in the executed application otherwise inconsistent and only partially useful results are obtained.

Hardware-level methodologies exploit hardware-level fault injection platforms (mainly by emulating SEUs in the configuration memory of FPGAs or in the registers of GPUs) to accurately emulate the effects of faults in the hardware where the DL model will be executed. These approaches are highly accurate because of the ability of reproducing the faulty behaviour, and are time-wise more sustainable than simulation solutions, since fault injection can be executed at speed. On the other hand, these approaches are generally hard to be deployed, demanding specific hardware-level skills that a design team specialised in DL may lack. Moreover, the application of resilience analyses belonging to this class are typically carried out late in the design process, thus making modifications expensive.

Finally, cross-layer methodologies try to bring together the advantages of the previous methodologies by splitting the analysis into two steps. First, a hardware platform-specific fault injection or radiation testing activity is performed on a portion of the DL engine under analysis or on the single operators. In this way the actual effects of the faults occurring into an FPGA, a GPU or a CPU while accelerating/executing a DL engine are captured. Then, the observed effects are used to feed a higher-level analysis/simulation engine to observe how these effects propagate through the subsequent layers of the model and if and how they affect the final output.

An additional group gathers a number of works that serve as custom solutions, because they apply to a specific DLs model, or actually report application case studies, presenting interesting results that are though specifically tailored for the discussed context. A brief description of contributions that belong to this class and to the above mentioned groups follows.

3.1.1. Application-level methodologies

The paper in [23] presents one of the first tools for the resilience analysis of Convolutional Neural Networks (CNNs) by performing error injection at application level. The tool, developed within the Darknet ML framework, allows to corrupt the weights in the CNN model and to carry out error simulation campaigns. The goal of the tool is to analyse the safety of the DL applications; in particular, single experiments are classified as *masked, observed safe* and *observed unsafe*; a threshold set to $+/-5\%$ is used to analyse the difference between the top ranked

percentage in the erroneous result and the golden counterpart, and to determine the safe/unsafe class the corrupted output belongs to. The paper considers permanent faults affecting the CNN weights, not whatsoever relating these permanent functional faults to realistic faults in the underlying hardware running the application.

BinFI and TensorFI, presented in [24,25] respectively, are two subsequent contributions from the same research group, who, among other works, designed, developed and distributed two fault injection frameworks to evaluate ML systems resilience. BinFI identifies safety-critical bits in ML applications, while TensorFI analyses the effects of hardware and software faults that occur during the execution of TensorFlow programs. The paper in [26] presents TensorFI+, an extension of the TensorFI environment. In particular, TensorFI+ supports TensorFlow 2 models, allowing to analyse also non-sequential models by corrupting the output of the layers. An interesting feature of the framework is the possibility to inject faults during the training phase of the CNN.

PyTorchFI (presented in [27]) is an error simulation engine for DNNs that exploits the PyTorch framework. The tool allows to emulate faults by injecting *perturbations* in the weights and neurons of the convolutional layers of DNNs; the injected perturbations are functional errors, therefore no specific hardware architecture is considered. The analysis can be run on either CPUs or GPUs. A similar approach is implemented by Ares [28], an application-level error simulator for DNNs.[2] Again, the tool supports the simulation of perturbations modelling faults affecting the weights, the activation functions and the state of the neurons. Several observations and guidelines are also drawn in the paper: (i) the resilience of DNNs is strongly influenced by the data type and quantisation of the weights; (ii) some classes are more likely to be mispredicted than others; (iii) faults in the weights are more likely to cause a misprediction than those in the activation functions; and, (iv) the more weights are reused the higher the failure probability.

An analytical model called SERN is proposed in [29] for the resilience analysis of CNNs w.r.t. soft errors affecting the weights. The results obtained by SERN are then validated against a set of fault injection experiments. In particular, by exploiting the proposed framework, the authors analyse the impact of the occurring faults w.r.t. (i) the position of the affected bit within the stored value and (ii) the size of the stored value itself. The authors further propose to harden the CNN by protecting the most significant bits of the weights via ECC and by selectively duplicating the first convolutions layers of the network.

The work in [30] addresses the problem of how to define a significant fault injection campaign. In particular, the paper presents a methodology for statistical fault injection aimed at sizing the fault injection campaign and selecting the most appropriate fault locations to achieve statistically significant results. The proposed method is specifically tailored to evaluate the weights of the CNN models.

When working at this abstraction level, the attention is focused on the performance and behaviour of the DNN with respect to different implementation strategies, when a fault corrupts its elements. Studies [31–33] explore the effects of quantisation, compression and pruning on resilience. In particular, [31] explores the impact of transient faults on compressed DNNs with respect to different pruning rates and data precision. The adopted fault model is the single bit flip on random live values stored in latches or registers, and the authors develop a fault injection framework dubbed *TorchFI* to emulate such effects. The main outcomes of this work are: (i) 16-bit integer quantisation can mitigate the overall error propagation w.r.t. the 32-bit floating-point baseline; (ii) while 16-bit quantisation increases resilience, the more aggressive 8-bit quantisation can produce a resilience drop; and (iii) pruned networks being smaller and faster will be less prone to faults,

therefore possibly achieving a better resilience. Similar quantisation strategies are explored in [32], proposing a simulator for evaluating the resilience of DNNs based on the frameworks of Keras and Tensorflow. The targeted fault model includes SEUs in the inputs, in the weights and in the output of the operators. Finally, the work presented in [33] discusses a simulation analysis for understanding the fault resilience of compressed DNN models as compared to uncompressed ones. Simulation is then used to study the resilience of pruned and quantised DNNs w.r.t. not pruned and not quantised ones. The results presented in the paper demonstrate that, on the one hand, pruning does not impact the resilience of the DNN while, on the other hand, data quantisation largely increases it.

Another neural network element being tailored during the design and implementation of a system is the type of data, similarly to quantisation. Approximation can be adopted to leverage model accuracy and implementation costs (e.g., execution time, hardware resource demand and power consumption). Since such representation choice has an impact on resilience, some studies investigate this aspect. The authors in [34] exploit the application-level error simulator presented in [23] to analyse the safety w.r.t. the occurrence of permanent faults in the weights of two different CNNs when varying the data type. Both floating point and fixed point data types at different precision levels are considered. The conclusions are that the most resilient data type and precision level depend on the specific model; moreover, the paper suggests to select the most suited solution by trading safety and memory footprint of the various alternatives. Finally, the same authors have also analysed in [35] the resilience of the novel POSIT data types, specifically defined for AI computations, by means of the fault injection approach presented in [30]. Experimental results demonstrate how POSIT data types are less resilient than fixed point integer data types using a reduced precision.

The authors of [36] use an ad-hoc designed ML algorithm to build a *vulnerability model* of the parameters of the DNN. To reduce the number of required fault injection experiments to analyse the effects of bit flips, empirical considerations are introduced on the importance of the various bits within the value representation, both in the floating point and in the fixed point cases. The authors evaluate the benefits/loss of accuracy with respect to injecting faults in all locations showing that the outcome offers good opportunities.

3.1.2. Hardware-level methodologies

Libano and others investigates in various studies the resilience of CNNs accelerated onto FPGAs by means of both radiations tests and fault emulation. In particular, in [37] radiation testing experiments are performed to analyse the impact of data precision and degree of parallelism on the resilience of the network. The conclusions of the study are: (i) lower precision means less hardware resources and consequently lower fault probability; and (ii) more parallelism means more hardware resources but also faster execution thus, the best performance-resilience trade-off is reached with the highest achievable degree of parallelism.

An analysis of the effects of SEUs in Binarised Neural Networks (BNNs) accelerated onto SRAM-based FPGAs is presented in [38]. The authors exploit the Xilinx FINN framework to build the BNN and the FPGA Reliability Evaluation through JTAG (FREtZ) framework for the fault injection activity. The outcome of such logic-level fault injection experiment is subsequently exploited to carry out an in-depth layer-per-layer analysis of the effects of the faults on the accuracy of the network. The results of this study show that BNNs are inherently resilient to soft errors.

Additional examples of fault resilience analysis of CNNs accelerated onto FPGA devices are presented in [39,40]. In the former the authors explore alternative quantised designs and compare them against a classical TMR to evaluate costs and benefits. In the latter, the authors consider permanent stuck-at faults and explore their effects, investigating four popular CNNs, including Yolo. The analysis shows that hardware faults can cause both system exceptions, such as system

---

[2] The tool is dated 2018, outside the boundary of this investigation. However, we included it, because it is adopted in several of the analysed studies.

stall and abnormal runtime, and prediction accuracy loss. A custom evaluation metric based on accuracy loss is exploited, also taking into account system exception probability; the nominal and TMR-protected versions are analysed and compared against.

Another analysis for CNNs accelerated onto FPGA devices is presented in [41], where the focus is on investigating the impact of various pruning techniques on the resilience of the network. Several interesting considerations are drawn: (i) removing filters that marginally contribute to the final classification increases the resilience of the CNN w.r.t. fault in the configuration memory; (ii) networks with higher pruning rates are more resilient to errors affecting the weights; and (iii) only a small percentage of weights (about 30%) can (when corrupted) actually modify the behaviour of the network and the percentage is even smaller if we consider the ability of causing an accuracy loss (about 14%). The work in [42] extends previous analyses by considering also BRAM to better focus on the various elements' susceptibility, to later apply a TMR-based selective hardening strategy.

A broad contribution to this class of solutions comes from Rech's team, analysing the resilience to SEUs when executing DL applications on GPUs. In particular, in [43], radiation tests are used to cause realistic SEUs in the target device; then, they complement the first set of experiments with microarchitectural-level fault injection by means of the SASSIFI tool, to collect a more extensive set of results. In the experiments, various versions of the same CNN applications are analysed, including the nominal versions and the versions hardened by means of ECCs and ABFT strategies applied to the convolutional layer. In a subsequent work [44], the same research team evaluates with a similar approach the resilience of Google's TPU by means of radiation testing. The most interesting aspect of this work is the definition of a set of error models in terms of the spatial patterns of the erroneous values in the output tensor of the convolution operator.

The work in [45] presents a strategy to estimate the criticality of Processing Elements (PEs) in a systolic array with respect to faults that may permanently affect one of them, by building and training a *neural twin*. The aim is to simplify the complexity (in terms of time) to analyse faults' effects with respect to solutions based on fault injection (as the authors did in the past) by using a trained model of the PE. The analysis on the single element offers the expected advantages and coherence with the PE real fault/error behaviour, however the possibility to generalise and transfer the model to the rest of the PEs is still to be investigated. Finally, also the work in [46] focuses on the resilience of systolic arrays. The approach designs a RTL simulator to inject stuck-at faults both in the weights and in the processed data and uses it to evaluate various architectural configurations w.r.t. the achieved performance and resilience. Experiments are executed on a very simple LeNet CNN.

3.1.3. Cross-layer methodologies

Fidelity [47] is an accurate logic-level error simulator for DNNs accelerated via custom circuits. By exploiting a deep knowledge of the regular structure of DNN hardware accelerators, Fidelity is able to reproduce and track in software the effects of SEUs occurring in the underlying hardware platform and affecting both the weights and the neurons. Moreover, based on the application of Fidelity to a set of large networks the authors draw the following considerations: (i) not only the weights but also neurons and neuron scheduling highly affects the resilience of the network; (ii) the adopted data precision has an impact on the resilience; and, (iii) the larger the perturbation in the output of the neuron, the more likely the network suffers from a mis-classification. An evolution of the same method and tool has been later presented in [48], carried out on the NVDLA architecture [10], to investigate the effects of hardware faults (namely, SEUs but applicable to other models) on training performance and accuracy. A detailed investigation is carried out, leading to valuable insights that can also be generalised to different platforms. Based on the outcomes, the authors

propose a hardening solution exploiting a tailored partial re-execution of training runs when a problem is detected.

The work in [49] presents an analysis framework aimed at predicting the propagation of SEUs affecting the registers of a CPU executing a CNN. The SIMICS system simulator is employed to simulate the CPU and the executed CNN; corruptions in the CPU registers are introduced to simulate SEUs. A small set of fault simulation experiments are first performed to extract data that are later used to train a Generative Adversarial Network (GAN). The GAN represents the actual core of the methodology since, after its training, it is used to predict, layer by layer, the percentage of faults that will be masked, those that will cause a crash and the ones that will lead to a Silent Data Corruption (SDC).

The work in [50] presents another cross-layer error simulation framework; the approach is developed for a specific working scenario considering a microprocessor-based system running CNNs, focusing on faults affecting the RAM chip. The proposed approach is based on radiation experiments aimed at systematically analysing the effects of the faults to build application-level error models, defined in terms of data corruption patters and occurrence frequencies; such models are specifically devoted to corrupt CNN parameters, such as weights and bias constants. These models are integrated into an in-house error simulator offering the possibility to run CNN resilience analysis at the application level, and, therefore, on any platform, without the need of actually deploying the CNN on the target architecture. The framework is used to evaluate the resilience of various implementations of the LeNet-5 CNN obtained by using different data types, with different precisions. A three-level resilience analysis environment is proposed in [51]. The first step is a profiling where each instruction of the DL model under analysis is associated with information such as input values, output result and opcode by means of NVBit [52]. As a second step, the microarchitectural fault injection for GPUs (called FlexGripPlus [53]) is employed to characterise the effects of SEUs affecting the microarchitectural resources of the GPU cores while they are executing a single layer of the CNN. Finally, the observed erroneous behaviours are fed into a software-level fault simulation environment to analyse how faults propagate among the layers of the CNN. This enables a detailed analysis of the vulnerability factor of every layer in the considered CNN.

The work in [54] presents a cross-layer framework for the analysis of CNN sensitivity against faults. The framework consists of a CPU executing the CNN and an FPGA-based accelerator implementing the operator where faults are injected; the actual fault injection is realised by bit-flipping the content of the configuration memory of the FPGA device.

CLASSES [55] is a cross-layer error simulation framework developed in the TensorFlow ML framework. The tool is provided with a methodological approach to define error models starting from microarchitecture-level fault injection. More precisely, the method runs a preliminary fault injection campaign for each type of ML operator on the target architectural platform; then, corrupted output tensors are analysed to identify recurrent spatial patterns of erroneous values and their frequency. Thus, error models are defined for each one of these ML operators in terms of an algorithmic description of how the output tensor of the operator should be modified according to the observed spatial patterns. Error models are stored in a repository used by the application-level error simulator that runs the entire CNN model and injects errors on selected intermediate tensors produced by any operator. Since the error model captures the effects of the fault corrupting the target architecture, error simulation is performed at application level, on any machine, without the need to deploy the application on the target final hardware. The paper demonstrates the effectiveness of the tool and the companion approach in the scenario of Yolo CNN executed on a GPU, however, the approach is general and can be employed for any architecture and CNN model. The susceptibility to SEUs of the General Matrix Multiplication (GEMM), Fast Fourier Transform (FFT) and Winograd's convolution implementations has been studied in [56]

by exploiting CLASSES. The authors first characterise the effects of the SEUs affecting the GPU while executing the convolution operators; then, they analyse how the occurred faults impact on the overall CNN accuracy. The remarkable outcome of the analysis is that the GEMM-based convolution is the most resilient one against SEUs.

Similarly, SiFI-AI, presented in [57], is a hybrid simulation environment that combines PyTorch inference with a cycle-accurate RTL simulator of SEUs in the registers of TPUs. saca-Fi [58] works at the same abstraction level and consists of an execution simulator, a fault injection module, and a resilience analysis framework to analyse both transient and permanent faults in the registers, providing an Architecture Vulnerability Factor (AVF) evaluation. Based on the outcome of the analysis, as case studies, the authors propose to harden the most sensitive registers and data parts by means of ECC codes.

[59] presents FireNN; it is a cross-layer resilience analysis engine for CNNs accelerated onto FPGAs. The tool allows to study how SEUs occurring either in the CNN weights or in the layers output affect the CNN output. More precisely, the entire CNN is executed in software by means of the PyTorch framework while the CNN operator cons the target for the fault injection in transferred onto the FPGA device. Once the operator has been configured in the FPGA, the fault is injected and the (possibly corrupted) operator output is collected and then fed to the subsequent operators that, again, are executed in software.

LLTFI [60] supports framework-agnostic fault injection in both C/C++ programs and ML applications written using any high-level ML framework. It uses LLVM to compile the DNN model in the Intermediate Representation (IR) targeted for CPU platform, that is used for fault injection activities. In this way, the tool supports injection at the granularity of single IR instructions, allowing also to observe at a fine-grain level the error propagation among the various parts of the DNN. Based on these capabilities, LLTFI provides guidelines and metrics to drive the selective instruction hardening, as demonstrated by the experimental activities discussed in the paper.

A framework, called DeepAxe, for the analysis and the design space exploration of the effects of approximation and the trade-off between area occupation and reliability is presented in [61]. DeepAxe targets DNN accelerators implemented onto FPGAs. The framework starts with a Keras description of the DNN that is used to measure the *ground-truth* accuracy. Then, the Keras model is translated into C and fault simulation is performed to measure the reliability of the model. Finally, high-level synthesis is applied to obtain the hardware description, and approximation is applied thus allowing to evaluate the area occupation of the final circuit.

3.1.4. Custom methods

The paper in [62] from NVIDIA analyses the reliability and safety of a CNN (executed on a GPU) for object detection in the automotive application domain. Both fault simulation and radiation testing are carried out. It is one of the few papers where safety issues (Failure in Time in particular) are taken into account. The paper highlights how the use of ECCs for the protection of the content of the memory of the GPU increases the reliability of the system. On the other hand, the paper also states that ECC protection is not enough and that periodic structural tests are recommended to mitigate risks due to SEUs.

The impact of SEUs occurring in the weights on the accuracy of CNNs is analysed in [63] via an ad-hoc designed fault simulation framework. GoogleNet, Alexnet, VGG16, and SqueezeNet are considered in the analysis and the target hardware platform is a GPU. The analysis is carried out targeting three aspects: (i) data representation (fixed point versus floating point values), (ii) position of the corrupted bit within the value, and (iii) position of the corrupted layer within the network. The outcome of the analysis refers to (i) CNNs using fixed point values are much more resilient than the ones using floating point values, (ii) faults occurring in the exponent of floating point CNNs have the biggest impact on resilience (as expected), and (iii) the last layer of the network are the ones having the biggest impact on its resilience.

The works in [64,65] deal with two different case studies, analysing and improving the resilience of ResNet and GoogLeNet implemented on GPUs, respectively. In both cases the context is very specific such that, as the authors state, it is not possible to generalise the outcomes that, thus, can actually be exploited only in similar application contexts. Layer and kernel vulnerability is analysed by performing a fault injection campaign via SASSIFI, to identify the most vulnerable aspects of the implemented model. In [64] the authors also selectively harden some of the kernels that exhibited high vulnerability, by triplicating them and voting the output.

The paper in [66] presents an analysis of the resilience against SEUs affecting the weights of the LeNet5 CNN applied to the MNIST dataset. Based on the results of this analysis the authors draw several considerations: (i) faults affecting the convolutional layers are more likely to cause a significant accuracy drop than faults affecting the fully connected layers; (ii) the faults affecting the exponent of the floating point values used to represent the weights have the largest effect on the accuracy of the CNN; (iii) the use of Sigmoid operators instead of ReLU ones decreases the resilience of the CNN; and (iv) average pooling is more capable of preventing the propagation of faults compared to max pooling.

In [67] the reliability and safety analysis of a systolic array against stuck-at faults occurring in the datapath, i.e., weights, bias, multiplier, and accumulator units, is presented. The analysed faults are classified based on the severity of the effects they cause on the output of the systolic array. Based on such analysis, the paper additionally presents two algorithms for test pattern generation meant to detect the most critical faults. Similarly, a simulation analysis of the effects of permanent faults in the datapath occurring during the training of TPUs has been presented in [68]. The authors observe three possible effects: faults that are totally masked, faults that cause NaN/Inf values and faults that cause a sharp accuracy degradation. Finally a simple fault detection and reaction scheme is proposed: a training iteration is discovered to have suffered from a fault as soon as the training loss value exceeds a pre-configured bound; then, the two most recent training iterations are re-executed to recover from the fault.

3.2. Hardening strategies

The second class of reviewed works includes approaches for the hardening of systems running DL applications w.r.t. effects of faults corrupting the underlying hardware. These works focus on handling and mitigating SDCs, constituting the most dangerous effect of faults, because it is not detected by the system; a few contributions deal also with the recovery from Detected Unrecoverable Errors (DUEs). This class of works can be further partitioned into (i) approaches applying classical redundancy-based hardening strategies, and (ii) design strategies exploiting peculiar characteristics of DL models. One of the main challenges in the hardening process is the fact that DL applications are compute intensive; therefore selective or approximated techniques are generally defined when considering redundancy-based strategies, to limit overheads. Moreover, DL models are internally redundant and presents specific peculiarities that can be exploited to introduce a degree of intrinsic resilience to faults in the designed applications. This second group of works exploits these properties to define resilience-driven design methods.

3.2.1. Redundancy-based techniques

The work in [69] proposes two complementary selective hardening techniques for introducing fault tolerance in DL systems acting at application level, without targeting any specific architecture. The first technique works at design time to identify the most vulnerable feature maps. This vulnerability analysis is performed by means of metrics to estimate (i) the probability of activation of a fault while processing a feature map, and (ii) the probability of propagation of the generated error to the primary outputs of the CNN. Then, most vulnerable feature

maps are hardened by means of DWC, and, in the case of mismatch, re-execution is performed at run-time. The second proposed technique works at run-time and monitors with an ABED approach the outputs of each CNN inference. In particular, two metrics are used to classify the outputs as *suspicious*, and if needed, a re-execution is triggered. These two metrics are defined based on empirical observations showing that, when considering a CNN for classification activities, the difference between the top two confidence classes exhibits a strong inverse relationship with the occurrence of a mis-classification. The extensive experimental evaluation of the proposed techniques is performed in PyTorchFI, by the same authors, and is architecture agnostic.

The work in [70] proposes a hardening approach based on selective application of classical redundancy-based techniques against both transient faults in computations and permanent faults in the memory storing the weights. The approach exploits techniques for explainable AI to identify the most susceptible locations in the CNN at the granularity of the single weight, and neurons in the feature map whose corruption will possibly cause a mis-classifications with a high probability. Then, ECCs and TMR are selectively applied to the most critical weights and neurons, respectively. Even if the approach works at the application level and is prototyped in the PyTorch framework, it is particularly tailored for DNNs designed by using a low data precision, generally accelerated in hardware.

The authors in [71] develop a so-called *Resilient TensorFlow* framework, obtained by adding to TensorFlow a set of fault-aware implementations of its base operators, to address SEUs occurring in the underlying GPU device. Each new operator is implemented to execute a thread-level TMRed version of the nominal counterpart. Then, thread blocks are opportunistically scheduled and distributed on the GPU cores to avoid a single fault to corrupt multiple redundant threads. The proposed approach is validated by means of both application-level fault simulation, by means of TensorFI, and microarchitectural-level fault emulation, by means of NVBitFI [72]. An interesting further contribution is the introduction of the Operation Vulnerability Factor, a metric used to evaluate the resilience of operations, to validate the proposed solution. In our opinion, the metric could be adopted to compare different solutions focused on hardening the single operator.

The work in [73] puts together various preliminary contributions by the same research group on hardening CNNs executed on ARM CPUs. In particular, they evaluate through simulated fault injection at microarchitectural level, by means of the SOFIA tool [74], the resilience of various implementations of the same CNN with different data precision models (integers at 2, 4, and 8 bits). Based on the results, they harden the CNNs via two different techniques: (i) a partial TMR applied at instruction level on sub-parts of the application, or (ii) an ad-hoc allocation of variables to registers. The idea at the basis of the second technique is that minimising the number of used memory elements reduces the area exposed to radiations and therefore system resilience improves, here measured in terms of Mean Work To Failure (MWTF). The experimental analysis is performed on the MobileNet CNN. An evaluation of the proposed lightweight technique under neutron radiation is presented in [75].[3]

The work in [76] proposes a selective hardening approach for CNNs. First, the approach uses the CLASSES error simulator [55] to characterise the vulnerability against SEUs of each layer in the CNN. This metric is defined as the percentage of faults corrupting the single layer causing the final CNN output to be functionally different from the golden one, i.e., *unusable* as defined in [77]. As an example, when considering an image classification task, the output of the CNN is *usable* when the input image is correctly classified, even if the actual output percentage values are slightly different from the golden ones; on the other hand, the output is *unusable* when the output percentage values are highly corrupted thus causing a mis-classification of the

input image. Then, the overall resilience of the CNN is computed by combining the layers' vulnerability factors. The approach performs an optimisation of the hardening based on a selective layer duplication to co-optimise the overall resilience of the CNN and its overall execution time. The approach is applied to a set of 4 different CNN applications targeting a GPU device.

Another example of application-level selective hardening approach is the strategy in [78], that exploits a resilience score previously defined in [79] to rank neurons in the model; then, the approach prunes neurons classified as non-critical to reduce memory footprint, and triplicates neurons classified as critical to improve model resilience. The strategy is implemented in the PyTorch framework without targeting a specific hardware platform, and the resilience of the system is evaluated against errors randomly modifying or setting to zero the output values of the single neurons.

SHIELDeNN [80] and STMR [81] are two similar approaches, targeting BNNs implemented on FPGAs. Both tools perform a preliminary vulnerability analysis of the parameters of the BNN (in particular, weights and activation functions) to identify the most critical ones; this analysis is based on in-house fault simulators. Then, selective TMR is applied to the most critical parameters, at the granularity of entire layers in [80] and individual channels in [81]. Although both works target FPGA devices, they only harden against faults affecting the data memory storing BNN parameters, neglecting faults affecting the device configuration memory, whose corruption actually leads to a modified functionality.

Still targeting FPGAs, [82] presents a methodology for achieving a lightweight fault tolerance for CNNs. The idea is to avoid the classical TMR scheme by adopting an approximated NMR-based approach; instead of having three exact replicas of the CNN plus a voter, the proposed methodology exploits the so-called *ensemble learning*, an approach used in DL for increasing model accuracy. In particular, the technique introduces a number of redundant CNNs, that are simpler and smaller than the original one. During the training phase each CNN learns a *subset* of the problem; then, during testing/deployment all CNN output responses are *merged* by a *combiner* module that produces the final output as the original CNN would have computed. The methodology is applied to various versions of the ResNet CNN and the resilience evaluation is performed by means of a fault injector corrupting the FPGA configuration memory.

The work in [83] targets a hardware accelerator organised as a dataflow architecture for ML acceleration. The strategy exploits computing elements in the architecture currently having as activation value a zero or an identical value of a neighbour computing element; the aim is to duplicate the same computation of the neighbour element. Additional logic is introduced into the architecture to manage on-the-fly duplication of the computations, to check results and, if needed, to re-execute faulty elaborations. The advantage of the approach is to benefit from the massively parallel nature of the considered ML accelerator to introduce computation replicas at execution level without extending the architecture with additional computing elements. The architecture is experimentally validated onto an FPGA device by performing emulated fault injection in the registers of the RTL description.

[84] introduces several ABFT schemes to detect and correct errors in the convolutional layers during the inference process; to this end the authors develop in the Caffe framework a *soft error detection library for CNNs*, *FT-Caffe*. The approach is based on the adoption of checksum schemes and layer-wise optimisations, opportunely calibrated by means of a workflow that provides error detection and then error correction. Being it a runtime method, performance degradation is traded against fault resilience. Application-level error simulation is used by means of an in-house tool to evaluate the approach.

Two ABED techniques are proposed in [85,86] for linear layers, i.e., convolutional and fully-connected layers, targeting GPU devices. Both works are based on computation and checksum validation in

---

[3] In Table 8 we only report the latest, inclusive contribution.

matrix multiplication algorithms. The approach in [85] considers quantised models and is implemented in CUDA, using also the cuDNN library; the other layers are protected by traditional DWC. The experimental evaluation is performed through microarchitectural-level fault emulation by injecting single bit-flips in the layer inputs and outputs and weights, and through radiation testing. The approach in [86] defines two different checksum strategies: (i) a global one, being a refined version of the classical hardening scheme for matrix multiplication, and (ii) a thread-level one, where the classical scheme is redesigned to aggressively use the GPU tensor cores. A design-time profiling approach, called *intensity-guided ABFT*, is used to decide, for each CNN layer, which strategy is the most efficient one in terms of execution time. The paper presents only an experimental evaluation of the performance of the proposed approach, neglecting reliability measures.

It is worth mentioning another similar ABED strategy based on checksums [87], applicable to convolutional and fully-connected layers. As for the previous contributions, the authors propose a hardware module to accelerate computation and checksum validation. The evaluation is again performed at the application level within a custom error simulation environment developed in Keras and Tensorflow.

A hardening methodology based on a selective ECC application, dubbed harDNNing, is presented in [88]. The framework first performs fault injection experiments in the parameters of the various layers of the DNN model. As a second step, based on the results of these fault injections, ML models are trained to predict the criticality of all the parameters and of all the bits within a single parameter. Finally, ECC are selectively inserted to protect the previously identified critical bits and critical parameters, thus achieving low-overhead fault tolerance.

An analytical model to study the propagation of SEUs affecting the weights of a CNN is proposed in [89]. The authors define the concept of *SEU-Induced Parameter Perturbation (SIPP)* as the modification of the value of a CNN weight caused by an SEU. Once the possible SIPPs have been identified, the authors study if and how they propagate to the output of the CNN and, based on this analysis, the most critical weights are identified. As a final step, TMR or ECC are applied to the most critical weights to increase the resilience of the CNN.

3.2.2. DL algorithm-aware techniques

Paper [90] introduces *Ranger*, a fault correction technique identifying and modifying values presenting a deviation from the nominal ones, presumably due to the occurrence of transient faults in the processed data. The intuition at the basis of this technique, previously discussed in the paper presenting BinFI [24], is that each layer in a DNN model produces in output tensors containing elements included in a specific value range. Moreover, if a SEU generates a corrupted value in the output tensor sensibly different from its nominal range, there is a high probability that this will cause the DNN to generate an erroneous output, an event that does not occur when the corrupted values is anyway within the nominal range. Thus, the proposed low-cost technique consists in introducing on the output of selected DNN layers a new operator that clips those output values that are outside identified restriction bounds. The proposed idea is implemented in TensorFlow and evaluated by means of TensorFI.

Paper [91] presents a technique very similar to Ranger. The paper considers permanent faults in the weights of the DNN and defines a novel clipped version of the ReLU activation function, replacing output values larger than a given threshold with a 0. A methodology is proposed to identify a proper threshold capable of identifying possible faults causing out-of-range corrupted values and at the same time limiting the negative impact of this new operator on the accuracy of the overall DNN. The experimental evaluation is carried out by means of an in-house error simulator developed in PyTorch.

The work in [92] proposes yet another value range limiting strategy, implemented by modifying the activation function to perform a clipping against a threshold. Based on the limitations of previous

efforts in the same direction, the authors employ a fine-grained neuron-wise activation function, to be determined in a supplementary training phase, that follows the traditional accuracy training. To this end, the work proposes a two-steps framework that supports the design and implementation of a resilient DNN. The authors analyse the final implementation against memory faults, that is weights and biases of different layers, as well as parameters of activation functions. An in-house error simulator is developed in PyTorch for running an experimental evaluation. Results are compared against hardening solutions proposed in [90,91], showing an improvement.

Few other papers present alternative strategies to address faults causing high-magnitude errors. For instance, the work in [93] combines quantisation tailored on the parameter distribution at each DNN layer and a training method considering a specific loss function, optimistically exploiting the selected quantisation scheme not to decrease the accuracy while pursuing a high resilience. This approach, validated in an ad-hoc application level error simulation framework developed in PyTorch, outperforms two different strategies proposed by the same authors and a state-of-the-art approach based on explicit value range clipping [91]. Another work exploiting the statistical distribution of the tensor values is proposed in [94]; it defines thresholds for localising and suppressing errors. The technique is coupled with state-of-the-art checksum strategies for error detection. The authors in [95] also exploit the statistical distribution of the values in the output of the DNN, before applying the final softmax normalisation, to detect outliers, which represent a suspicious symptom of a fault corrupting the system.

In this class of papers, we found papers that optimise the memory overhead introduced by the application of ECC to the DNN weights by exploiting peculiar properties and characteristics of DNN models. As an example, the study in [96] proposes a novel training scheme, namely Weight Distribution Oriented Training (WOT), to regularise the weight distribution of CNNs so that they become more amenable for protection by encoding without incurring in overheads. The idea is to exploit the fact that weights in a well-trained CNN are small number, requiring a few bits to be represented with respect to the available ones. Therefore, part of the bits are used to hold the ECC, effectively using a 8-bit quantisation strategy for the weights, to use the remaining bits for the checksum. The evaluations is performed at application level by means of a custom fault simulation method in PyTorch. Another similar work is presented in [97] where a Double Error Correcting code based on parity is adopted to protect weights against stuck-at faults. The proposed approach, prototyped in Keras, outperforms the one in [96].

Finally, other papers follow the same path, also broadening the field of analysis. As an example the authors in [98] continue the analysis of the resilience of the various data types by considering the recently introduced *Brain-Float 16 (bf16)* format; since this data type is obtained by removing 16 bits from the mantissa of the standard 32 bit floating point, it presents a higher vulnerability to faults. Based on the resilience analysis, the authors define another similar coding scheme for the weights of the model. In particular, to avoid any memory overhead, a parity code is applied by using the Least Significant Bit (LSB) of each word as the checking bit; the intuition is that a change in the LSB marginally affects the model accuracy. Then, when a parity error is detected, the entire weight is set to zero; in fact, as studied in [99], a change of a single weight to zero generally does not affect the DNN result.

A novel hardening paradigm, dubbed *fault-aware training* is proposed in [100,101]. The idea behind this technique is to inject faults during the training process to force the CNN to learn how to deal with the occurrence of faults at inference time. This promising technique, on the one hand, enables a *low-cost* hardening, but, on the other hand, it poses new challenges to the designer. Indeed, it is vital to identify the proper amount of faults to be presented to the CNN during the training phase; a high number could increase resilience, introducing the side-effect of preventing training convergence and producing an excessively large CNN. A reduced number of faults will result in a quick

Table 7
Contributions according to their type.

Resilience analysis	
Application-level methodologies	
Error simulation	[23–36]
Hardware-level methodologies	
Radiation testing	[37,42–44]
Fault injection	[38–43]
Error simulation	[45,46]
Cross-level methodologies	
Radiation testing	[50]
Fault injection	[54–56,58,59]
Error simulation	[47–51,55–57,60,61]
Case studies	
Radiation testing	[62]
Fault injection	[64,65]
Error simulation	[62,63,66–68]
Hardening strategies	
Redundancy-based techniques	
DWC	[76,85]
TMR	[70,71,75,78,80,81,89]
NMR	[82]
D+R	[69,105]
ABFT	[84,94,106]
ABED	[69,84–87,107]
ECC	[70,88,89,96–98]
CHK	[83,105]
DL algorithm-aware techniques	
Value type/distribution analysis	[90–92,94,95,97] [105,106]
Fault-aware training	[96,100,101,103,104,107]

but possibly ineffective training. In the paper the newly proposed fault-aware training is coupled with two additional CNN model modifications aimed at mitigating high-magnitude errors: (i) replacing the standard ReLU activation with its clipped counterpart, ReLU6 (originally proposed in [102]); and (ii) re-ordering the layers in the CNN such that ReLU6 is always executed before batch normalisation. The paper evaluates the proposed approach by considering a GPU target device and by using both microarchitectural fault injection (via NVbitFI) and application level error simulation (via a Python-based in-house tool). Fault-aware training is also investigated in [103], where the authors introduce specific loss functions and training algorithm to deal with multiple bit errors. The evaluation is carried out at the application level by not considering any specific hardware platform.

Fault-aware weight re-tuning for fault mitigation is proposed in [104]. The authors first analyse the resilience against permanent faults of a Multiply and Accumulate (MAC) structure generally used in GPUs and TPUs. In particular, the authors analyse how the structure is sensitive to SA faults a CNN is w.r.t. (i) the degree of approximation adopted in the employed multipliers; (ii) the position of the faulty bit in the corrupted value; and (iii) the position of the layer affected by the fault in the whole CNN. The authors propose to prune the weights that are mapped on the corrupted bits and that are thus going to be affected by the SA faults (previously identified through post-production test procedures). Once such pruning has been carried out, re-training of the CNN is performed. The experimental evaluation is performed by designing a systolic array architecture based on the considered MAC structure. Fault injection campaigns are run with an in-house error simulator in TensorFlow.

The work in [105] first performs a systematic analysis of the Program Vulnerability Factor (PVF) of the various instructions of an ARM CPU executing DL applications. Experiments are performed by means of a fault emulation tool corrupting the ISA registers by means of the on-chip debugging interface. Then, it defines two techniques to harden the considered system against SDCs: (i) selective kernel-level DWC with re-execution, and (ii) a *symptom-based* technique checking all values of the intermediate results against a given threshold to trigger a re-execution when a value is above it. This second technique is based on the same intuition of the range restriction strategies discussed above (e.g., [90,91]). Finally, the paper considers the adoption of kernel-level

check-pointing to recover from crashes or other DUE. In a subsequent work [97], the same authors note that output values of a DNN layer present a regular data distribution that can be analysed at runtime to compute, during the inference process, the two thresholds to be used for the range restriction technique.

[107] focuses on a different perspective with respect to all previous contributions: the impact of faults during model training. An in-house error simulator is defined within the Caffe framework to inject bit-flips in the variables to simulate SEUs affecting the High Performance Computing system running the training procedure. Outcomes of such an analysis are that (as already emerged in other works for errors affecting floating point values and layers) (i) most training failures result from higher order bit flipping in the exponents, and (ii) convolutional layers are more failure prone. Moreover, the authors highlight how monitoring the value of the loss function among the various training iterations is an effective signal to detect most of the SDCs causing a training failure. Based on this observation, an ad-hoc error detection strategy is defined for training against failures due to SEUs.

A mitigation methodology without redundant hardware and without model retraining for permanent faults is systolic arrays is presented in [106]. The method exploits fault maps generated during post-fabrication testing to arrange significant data to MACs with fewer faults. Moreover, the authors propose to compensate the effect of a fault by *correcting* the faulty value substituting it with the value of the sign (they call this technique *sign compensation*).

The adoption of the two identified main classes, namely *resilience analysis* and *hardening strategies*, to partition the reviewed contributions allows us to organise them based on the main focus of the novelty of the presented solution. Table 7 offers a bird's-eye view of this classification and summarises the outcome.

As mentioned, the classification framework we define allows us to capture the elements we deem more relevant emerging from the reviewed contribution, thus providing a guide in identifying pertinent state-of-the-art proposals to build upon or to compare against. Table 8 collects the 76 entries of the analysed papers for an easy access to the information.

Another classification perspective that may be of interest maps the reviewed analysis or/and hardening techniques to the DL task and the DL model they are applied to. While some contributions compare their results against other solutions, this is not the most common case. Furthermore, results are significantly related to the adopted application context (e.g., DL task, considered dataset) and should be re-evaluated for different ones. However, it can be of interest to be able to identify what solutions have been already applied to a specific application context. Table 9 synthesises the application context per contribution, reporting the DL task and specific model; considering also the dataset selected for each model resulted in too a fragmented and heterogeneous map to be useful. For the same reason, we collapsed all alternative designs of the same model in a single item (e.g., ResNet-18, ResNet-50, ResNet-101, etc. have been collapsed in a single item named ResNet). Moreover, the table includes a column for each DL model adopted at least in two papers, all remaining models considered only once have been listed in the *others* column. Finally, for the sake of completeness, for each paper we report all used models including those not analysed in this work (e.g., transformers).

As we can notice, most contributions focus on DL models performing image classification. Some models, as ResNet or VGG, represent very popular case studies. A few works consider less-known, very specific models and frequently, authors define simple custom DL models to evaluate the proposed technique (see the *custom* column); inevitably, this reduces the generality and the reproducibility of the performed experiments.

Finally, Table 10 collects the information on the available software presented in the works listed in Table 8 and commented in the "Tool support" column.

Table 8
Contribution classification.

Paper	Scope	Abs. Lev.	HW Plat.	Fault Model	Error Model	ML Fram.	Tool	Rep.	Analysis			Hardening		
									Dep. Attr.	Inject Meth.	Out	Rel. Pro.	Tech.	Str.
[28]	A	APP	any	PFunc/TFunc	P/DV/NO	KE	Yes	Yes	Re	Si	[a]	–	–	–
[23]	A	APP	any	PFunc	P	DK	No	Yes	Sa	Si	HG	–	–	–
[24]	A	APP	any	TFunc	LO	TF	Yes	Yes	Re	Si		–	–	–
[25]	A	APP	any	TFunc	LO/P	TF	Yes	Yes	Re	Si		–	–	–
[26]	A	APP	any	TFunc	LO	KE	Yes	Yes	Re	Si		–	–	–
[27]	A	APP	any	PFunc/TFunc	P/NO	PT	Yes	Yes	Re	Si		–	–	–
[29]	A	APP	any	TFunc	P	KE/TF	No	No	VF/Re	Si		FD/FT	Sel	ECC+DWC
[30]	A	APP	any	SA	P	–	No	No	Re	Si		–	–	–
[31]	A	APP	any	PFunc/TFunc	LO	PT	No	No	Re	Si	HG	–	–	–
[32]	A	APP	any	PFunc/TFunc	LO/P/NO	KE/TF	No	No	Re	Si		–	–	–
[33]	A	APP	any	TFunc	DV	PT	No	No	Re	Si		–	–	–
[34]	A	APP	any	SA	P	DK	No	No	Re	Si		–	–	–
[35]	A	APP	any	SA	P	–	No	No	Re	Si		–	–	–
[36]	A	APP	any	PFunc/TFunc	P/LO	–	No	No	Re	Si		–	–	–
[37]	A	DEV	FPGA	SEU/SET	REG	TF	No	No	Re	Ra	HG	–	–	–
[38]	A	RTL	FPGA	SEU	REG	FI	No	No	VF	Em		–	–	–
[39]	A	DEV/RTL	FPGA	SEU	REG	–	No	No	Re	Em		–	–	–
[40]	A	RTL	FPGA	SA	REG	–	Yes	Yes	Re	Em		–	–	–
[41]	A	RTL	FPGA	SEU	REG	–	No	No	Re	Em	HG	–	–	–
[42]	A	RTL	FPGA	SEU	REG	–	No	No	Re	Ra/Em	HG	–	–	–
[43]	A	DEV/ISA/ALG	GPU	SEU	REG	DK	No	No	Re	Ra/Em		FT	Spec	ECC/ABFT
[44]	A	DEV	TPU	SEU	REG	TF	No	No	Re	Ra		–	–	–
[45]	A	RTL	TPU	SA	REG	PT	No	No	Re	Si		–	–	–
[46]	A	ISA	TPU	SA	P/DV	ND	No	No	Re	Si		–	–	–
[47]	A	RTL/APP	TPU	SEU	REG	TF	Yes	Yes	Re	Si	HG	–	–	–
[48]	A	RTL/APP	TPU	SEU	REG	TF	Yes	Yes	Re	Si	HG	FT	Spec	D+R
[49]	A	ISA/APP	CPU	SEU	REG	–	No	No	VF	Si		–	–	–
[50]	A	DEV/APP	CPU	SEU/SA	REG	ND	No	No	Re	Ra/Si		–	–	–
[51]	A	ISA/APP	GPU	SA	REG	–	No	No	Re	Si		–	–	–
[54]	A	RTL/APP	FPGA	SEU	REG	–	No	No	Re	Em		–	–	–
[55]	A	ISA/APP	GPU	SEU	REG	TF	Yes	Yes	VF/Re	Em/Si		–	–	–
[56]	A	ISA/APP	GPU	SEU	REG	TF	No	No	Re/VF	Em/Si	HG	–	–	–
[57]	A	RTL/APP	TPU	SEU	REG	PT	No	No	VF	Si	HG	–	–	–
[58]	A	RTL/APP	TPU	SEU/SA	REG	KE	Yes	Yes	VF/Re	Em	HG	–	–	–
[59]	A	RTL/APP	FPGA	SEU	REG	PT	No	No	Re	Em		–	–	–
[60]	A	ISA/APP	CPU	SEU	REG	any	Yes	Yes	Re/VF	Si	HG	–	–	–
[61]	A	RTL/APP	FPGA	SEU	REG	KE	No	No	Re	Si		–	–	–
[62]	A	DEV/RTL/APP	GPU	SEU	REG	TR	No	No	Re/Sa	Ra/Si		–	–	–
[63]	A	APP	any	TFunc	P	CA	Yes	Yes	Re	Si	HG	–	–	–
[64]	A	ISA/APP	GPU	SEU	REG	DK	No	Yes	VF/Re	Em		FT	Sel	TMR
[65]	A	ISA/APP	GPU	SEU	REG	DK	No	Yes	VF/Re	Em		–	–	–
[66]	A	APP	any	TFunc	P	–	No	No	Re	Si	HG	–	–	–
[67]	A	RTL/APP	TPU	SA	REG	KE	No	No	Re/VF/Sa	Si		–	–	–
[68]	A	RTL/APP	TPU	SA	REG	–	No	No	VF	Si	HG	FD/FT	Full	DL
[69]	H	APP	any	SEU	NO	PT	No	No	VF	Si		FT	Sel	D+R/ABED
[70]	H	APP	any	SEU/SA	P/DV	PT	Yes	Yes	–	–	–	FT	Sel	TMR/ECC
[71]	H	ALG	GPU	SEU	REG/LO	TF	No	No	–	–	–	FT	Ex	TMR
[75]	H	ISA/ALG	CPU	SEU	REG	CN	No	No	Rel	Si		FT	Sel	TMR/DL
[76]	H	ISA/APP	GPU	SEU	LO	TF	No	Yes	VF/Re	Em/Si		FD	Sel	DWC
[78]	H	APP	any	TFunc	NO	PT	No	No	–	–	–	FT	Sel	TMR
[80]	H	APP	FPGA	SEU	P	FI	No	No	VF	Em		FT	Sel	TMR
[81]	H	APP	FPGA	SA	P	FI	No	No	VF	Em		FT	Sel	TMR
[82]	H	APP	FPGA	SEU	REG	–	No	No	–	–	–	FT	Ax	NMR
[83]	H	RTL	TPU	SEU	REG	–	No	No	–	–	–	FT	Sel	D+R
[84]	H	ALG	any	SEU	LO	CF	No	Yes	–	–	–	FD/FT	Spec	ABED/ABFT
[85]	H	ALG	GPU	SEU	REG/LO/P	cu	No	No	–	–	–	FD	Spec	ABED/DWC
[86]	H	ALG	GPU	SEU	DV	–	No	No	–	–	–	FD	Spec	ABED
[87]	H	ALG	any	SEU	P/LO	KE/TF	No	No	–	–	–	FD	Spec	ABED
[88]	H	APP	any	SEU	P	–	No	No	VF	Em	HG	FT	Sel	ECC
[89]	H	APP	any	SEU	P	–	No	No	VF	An	HG	FT	Sel	ECC/TMR
[90]	H	APP	any	SEU	DV	TF	Yes	Yes	–	–	–	FT	Full	DL
[91]	H	APP	any	Pfunc	P	PT	No	Yes	–	–	–	FT	Full	DL
[92]	H	APP	any	SEU	P	PT	No	No	–	–	–	FT	Full	DL
[93]	H	APP	any	Pfunc	P/LO	PT	No	No	–	–	–	FT	Full	DL
[94]	H	APP	any	Pfunc	P/LO	PT	No	No	–	–	–	FT	Full	ABEF/DL
[95]	H	APP	any	SA	P	–	No	No	–	–	–	FT	Full	DL
[96]	H	APP	any	SEU	P	PT	No	No	–	–	–	FT	Spec	ECC/DL
[97]	H	APP	any	SA	P	KE	No	No	–	–	–	FT	Spec	ECC/DL
[98]	H	APP	any	SEU	P	ND	No	Yes	Re	Si		FD/FT	Spec	ECC/DL
[100]	H	APP	GPU	SA	P	PT	No	No	–	–	–	FT	Full	DL
[101]	H	APP	GPU	SEU	REG/LO	–	No	No	–	–	–	FT	Full	DL

Table 8 (*continued*).

Paper	Scope	Abs. Lev.	HW Plat.	Fault Model	Error Model	ML Fram.	Tool	Rep.	Analysis			Hardening		
									Dep. Attr.	Inject Meth.	Out	Rel. Pro.	Tech.	Str.
[103]	H	APP	any	SEU	P/NO	–	No	No	–	–	–	FT	Full	DL
[104]	H	RTL/APP	TPU	SA	REG	TF	No	No	VF/Re	Si		FT	Full	DL
[105]	H	ISA/APP	CPU	SEU	REG	CM	No	No	Re/VF	Em		FT	Sel	D+R/DL/CHK
[107]	H	APP	any	SEU	P/DV	CF	No	No	Re	Si		FD	Spec	ABED/DL
[106]	H	ALG	TPU	SA	P/LO	–	No	No	–	–	–	FT	Spec	ABFT/DL

a QM is always present and therefore omitted, we only report HG when applicable.

4. Insights, challenges and opportunities

The high number of pertinent contributions in the last four years (i.e., 244 authored by more than 400 scientists) shows a dynamic context, that in this decade has been fostering interesting and relevant outcomes, characterised by some common aspects, that we summarise in the following, together with open challenges and opportunities (beyond the ones highlighted by [2]).

Trend The number of contributions in the years has been increasing (as Fig. 1 shows) if we consider that the spectrum of analysis and design targets has grown and the works reported in the chart cover only a limited research area (the one included in this survey) with respect to the total.

DL design impact on resilience Numerous are the studies that explore how different DL design choices – from data type, to data quantisation, from pruning to compression – affect the resulting network resilience to faults corrupting both stored data (e.g., weights, neuron output) and manipulation (e.g., convolution output). Such impact, though, is heavily and strictly related to the specific adopted DL solution, and although some general considerations are drawn, there is no "one ground truth that applies to every case" so that, in our opinion, every time a DL application has to be deployed in a safety/mission-critical application domain, analysis and hardening solutions need to be specifically tailored. To this end, approaches providing usable tools and methods to analyse and harden a DL application seem to be of great interest.

Global techniques' comparison New hardening techniques are usually presented and applied to a selected application "context" defined by the task the DL application targets, the used input dataset, the adopted representation. The effectiveness of the technique and the quality of the outcome could vary when applied to a different application context, therefore it is not straightforward to compare different techniques and make a global ranking; given an application "context" all compatible techniques should be applied to be able to make an educated selection. **Challenge:** application of a new technique to a differentiated set of application contexts, highlighting whether the context has a significant impact on the outcomes and thus supporting possible adopters in the selection of the most appropriate solution.

Metrics For both the analysis and hardening strategies, most contributions can be partitioned into two classes, those evaluating resilience with respect to conventional reliability metrics, such as Mean Time To Failure, Failures in Time , Architecture Vulnerability Factor, Program Vulnerability Factor, Kernel Vulnerability Factor or the Silent Data Corruption rate (e.g., [43]) and those who adopt an *application-aware* metric, more closely related to the specific and special context, such as usable/not usable (e.g., [73,77]). Both classical and innovative figures of merit are adopted or defined, leading to numerous alternative visions. Some of the best contributions report comparative results that allow the reader to identify benefits and potentials of the new discussed solutions, but the rich set of different quantitative metrics makes the task not an easy one. **Challenge:** although the choice of the adopted metric depends on the application context, future efforts could go in the direction of reporting always the results also with respect to a commonly adopted metric, to enable fair comparisons.

Cross-layer strategies The complexity of the hardware platforms able to efficiently execute heavy ML/DL applications, and that of the applications themselves, initially led to contributions that worked either at the architecture level (working on faults), or at the application level (working on errors). However, the gap between these levels and the necessity to maintain a correspondence between faults and errors to provide a reliable susceptibility/resilience evaluation are spurring cross-layer approaches that explore and support such a fault-error relation.

Fault injection tools and their availability Considering the application context and the involved elements, fault injection is a critical task with respect to (i) the experiment time, (ii) the controllability/observability aspects, and (iii) the adherence of the injected errors to the underlying realistic faults. Specifically targeting the domain of interest, several fault injection tools have been recently proposed, working from the architectural level [24,47,60,72] to the application one [25,27,28], or cross-layer [55,59]. Although several of them are available (see Table 10 for the available open-source software packages), when developing hardening techniques and strategies, proprietary fault injection solutions are devised, sometimes to drive a selective hardening policy based on the analysis outcomes. **Challenge:** an ecosystem of available tools working at different abstraction levels, on different platforms could indeed allow for a systemic effort to tackle DL resilience for present and future challenges.

Reproducible research One of the critical activities when developing new methods is the evaluation of their performance with respect to existing ones, to motivate the introduction of yet another approach. Often, the comparison is carried out against the vanilla solution, the baseline implementation without any sort of hardening. Indeed, only a few contributions (besides the ones proposing a new tool) share and make public their software/data. **Challenge:** encourage reproducible research to foster stronger contributions, as well as the possibility to move towards an integrated ecosystem of solutions for the different hardware/software/application variants. As an example, an available benchmark suite that offers for the various hardware/software/application contexts a reference to (i) compare solutions, and (ii) support the integration of complementary approaches , could be a valuable asset for the community.

Community There are a number of very active research groups on the topic, that are steadily contributing to the discussion. To visually get an overview of such a community, the awareness and

Table 9
Contributions classification with respect to the application context: DL task and net.

	Paper	Image Classification											Regression		Object detection	Others
		ResNet	VGG	LeNet	AlexNet	MobileNet	SqueezeNet	DenseNet	GoogleNet	Inception	ShuffleNet	Custom	Dave	Comma.ai	Yolo	
Analysis	[28]	✓	✓	✓												TiGRU (Speech Classification)
	[23]														✓	
	[24]		✓	✓	✓							✓	✓	✓		kNN (Image Classification)
	[25]	✓	✓	✓	✓			✓				✓				RNN (Image Classification) – U-net (Image Segmentation)
	[26]	✓	✓			✓			✓	✓						Xception (Image Classification)
	[27]	✓	✓		✓			✓	✓	✓	✓					
	[29]	✓	✓	✓								✓				
	[30]	✓				✓										
	[31]	✓			✓											
	[32]	✓		✓		✓						✓				
	[33]			✓	✓											
	[34]		✓												✓	
	[35]		✓													
	[36]	✓													✓	
	[37]											✓				
	[38]											✓				
	[39]															ZynqNet (Image Classification)
	[40]	✓													✓	LSTM (Voice Processing) – DCGAN (Image Generation)
	[41]	✓	✓			✓										
	[42]			✓												
	[43]	✓													✓	R-CNN (Object Detection)
	[44]	✓			✓				✓							SSD (Object Detection)
	[45]		✓													
	[46]		✓													
	[47]	✓			✓					✓					✓	Transformer (Language Processing)
	[48]	✓						✓							✓	EfficientNet, NFNet (Image Classification) – Transformer (Language Processing)
	[49]															RNN (Control)
	[50]		✓													
	[51]		✓													
	[54]														✓	
	[55]		✓									✓			✓	
	[56]	✓		✓	✓											
	[57]	✓							✓							
	[58]		✓	✓								✓				
	[59]			✓												
	[60]		✓	✓	✓		✓					✓	✓			
	[61]		✓	✓								✓				
	[62]															NVIDIA-DriveWorks (Object Detection)
	[63]		✓		✓		✓		✓							
	[64]	✓														
	[65]								✓							
	[66]		✓													
	[67]											✓				
	[68]	✓													✓	Transformer (Language Processing)
Hardening	[69]	✓	✓		✓	✓	✓		✓		✓					
	[70]	✓														
	[71]		✓		✓	✓										
	[75]				✓											
	[76]		✓									✓		✓		PilotNet (Image Classification)
	[78]	✓						✓								
	[80]		✓									✓				
	[81]											✓				
	[82]	✓														
	[83]	✓	✓												✓	
	[84]	✓	✓		✓										✓	
	[85]	✓	✓	✓												
	[86]	✓	✓		✓			✓	✓		✓					Coral, Roundabout, Taipei, Amsterdam (Video Processing) – DLRM (Recommendation)
	[87]			✓												
	[88]	✓		✓												
	[89]	✓	✓													NIN (Image Classification)
	[90]	✓	✓	✓	✓		✓						✓	✓		
	[91]		✓		✓											
	[92]	✓	✓		✓											
	[93]	✓	✓	✓			✓									
	[94]	✓		✓												
	[95]	✓														
	[96]	✓	✓				✓									
	[97]	✓			✓			✓		✓						

Table 9 (*continued*).

Paper	Image Classification											Regression		Object detection	Others
	ResNet	VGG	LeNet	AlexNet	MobileNet	SqueezeNet	DenseNet	GoogleNet	Inception	ShuffleNet	Custom	Dave	Comma.ai	Yolo	
[98]	✓				✓	✓									
[100]											✓				
[101]	✓														
[103]	✓				✓						✓				
[104]		✓	✓	✓							✓				
[105]		✓	✓	✓											
[107]	✓														ConvNet, LRCN (Image Classification)
[106]	✓														

Table 10

Open-source software made available from the works presented in Table 8 (Tool support).

Ref.	Name	url
[28]	Ares	github.com/alugupta/ares
[24]	BinFI	github.com/DependableSystemsLab/TensorFI-BinaryFI
[25]	TensorFI2	github.com/DependableSystemsLab/TensorFI2
[26]	TensorFI+	github.com/sabuj7177/TensorFIPlus
[27]	PyTorchFI	github.com/PyTorchfi/PyTorchfi
[40]		github.com/ICT-CHASE/fault-analysis-of-FPGA-based-NN-accelerator
[47]	FIdelity	github.com/silvaurus/FIdelityFramework
[48]	FIdelityTraining	https://github.com/YLab-UChicago/ISCA_AE
[55]	CLASSES	github.com/D4De/classes
[58]	saca-FI	github.com/One-B-Tree/Saca-FI-A-microarchitecture-level-fault-injection-framework-for-CNN-accelerator
[60]	LLTFI	github.com/DependableSystemsLab/LLTFI
[63]		github.com/cypox/CNN-Fault-Injector
[70]		github.com/Msabih/FaultTolerantDnnXai
[90]	Ranger	github.com/DependableSystemsLab/Ranger

relationships among the research groups, as well as the typical venues where the topic is presented and discussed, we exploited VOSviewer [108]. On the 222 papers considered eligible we explored co-authorship, shown in Fig. 5. The analysis identifies 86 authors having authored at least 3 papers on the topic, belonging to 14 clusters (research groups). Links between nodes represent a co-authorship. Bibliographic coupling on the same data set is reported in Fig. 6, highlighting similar specific interests based on citations. We also explore the publication venues, shown in Fig. 7, reporting the venues where the included contributions have been published, highlighting the number of documents and the cross-references through links. Finally, we analyse the set of 85 included papers highlighting cross-citations and the number of citations per document to get insights on other scientists' awareness. The emerging view is reported in Fig. 8.

Synergy opportunity This work, as well as past literature review analyses, shows that ML resilience, and DL in the specific, against faults affecting the underlying hardware is a research area exhibiting many challenges and facets, setting an opportunity for creating a synergy in the research community towards the development of an ecosystem of methods and tools that can tackle the different facets of DL resilience against hardware faults.

5. Concluding remarks

This paper collects and reviews the most recent literature (since 2019) on the analysis and design of resilient DL algorithms and applications against faults in the underlying hardware. The analysis includes 85 studies focused on methods and tools dealing with the occurrence of transient and permanent faults possibly causing the DL application

to misbehave. Through a detailed search and selection process we reviewed the contributions and analysed them with respect to a classification framework supporting the reader in the identification of the most promising works based on the area of interest (e.g., with respect to the adopted fault model, error model or DL framework). The aim is twofold; (i) mapping the active research landscape on the matter, and (ii) classifying the contributions based on various parameters deemed of interest to support the interested reader in finding the relevant information they might be looking for (e.g., similar studies, solutions that might be applied, etc.). The study emphasises the breadth of the research and actually defines some boundaries to limit the included contributions, focusing on DL applications and the most commonly adopted fault models, leaving other facets (e.g., spiking neural networks, vision transformers, manufacturing and process-variation faults) to future studies. Some insights and overall considerations are also drawn; the vibrant research on this topic and the broad spectrum of challenges calls, in our opinion, towards the development of an ecosystem of solutions that offer a support in the implementation of resilient DL applications.

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.

Data availability

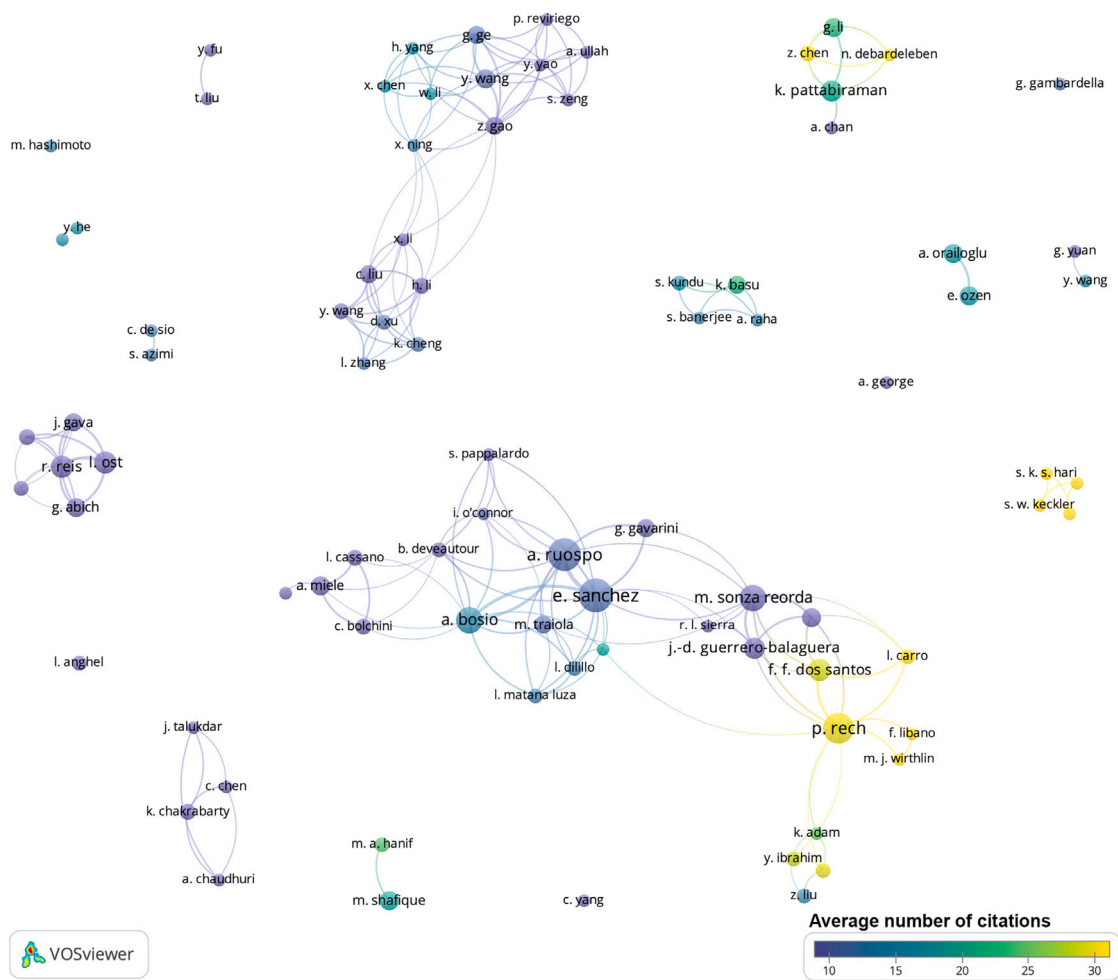No data was used for the research described in the article.

Fig. 5. Co-authorship analysis with "authors" as the unit of analysis. In this analysis, the minimum number of documents for each author is 3, and the number of resulting authors is 86, grouped in 14 clusters, accordingly. Node size depends on the number of documents and the connecting lines between them indicate the collaboration between authors. The colour spectrum represents the average number of citations.
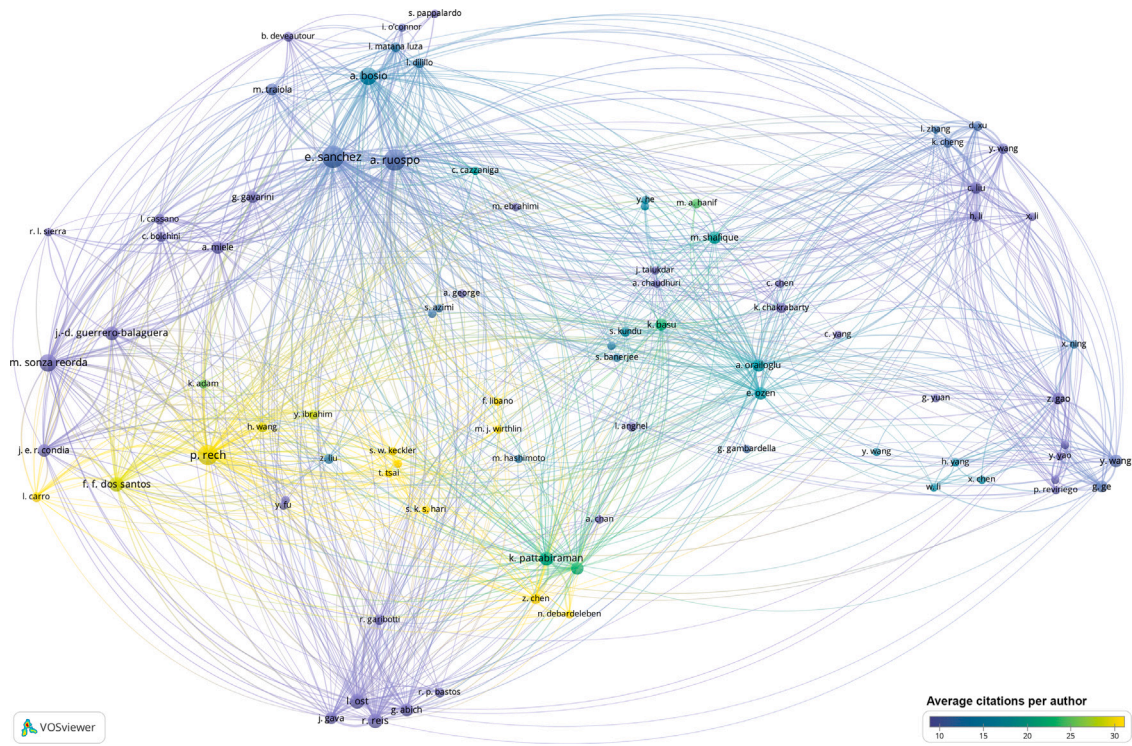
Fig. 6. Bibliographic coupling using "authors" as the unit of analysis. In this analysis, the minimum number of documents for each author is 3, and the number of selected authors is 86, grouped in 14 clusters, accordingly. Node size depends on the number of documents and the connecting lines between them indicate cross-referencing of the authored papers, mapping both topic similarity and awareness. The colour spectrum represents the average number of citations.
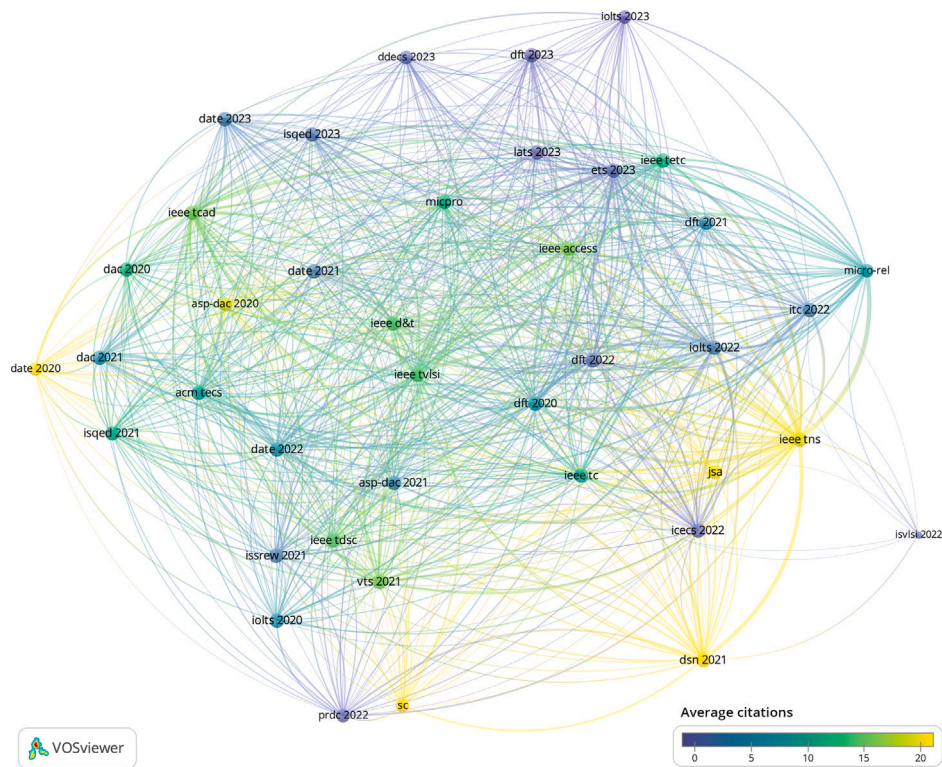
Fig. 7. Eligible studies: analysis of the publication venues with respect to the number of papers at such a venue. A link between two items means that one of them cites the other and the colour spectrum represents the average number of citations.
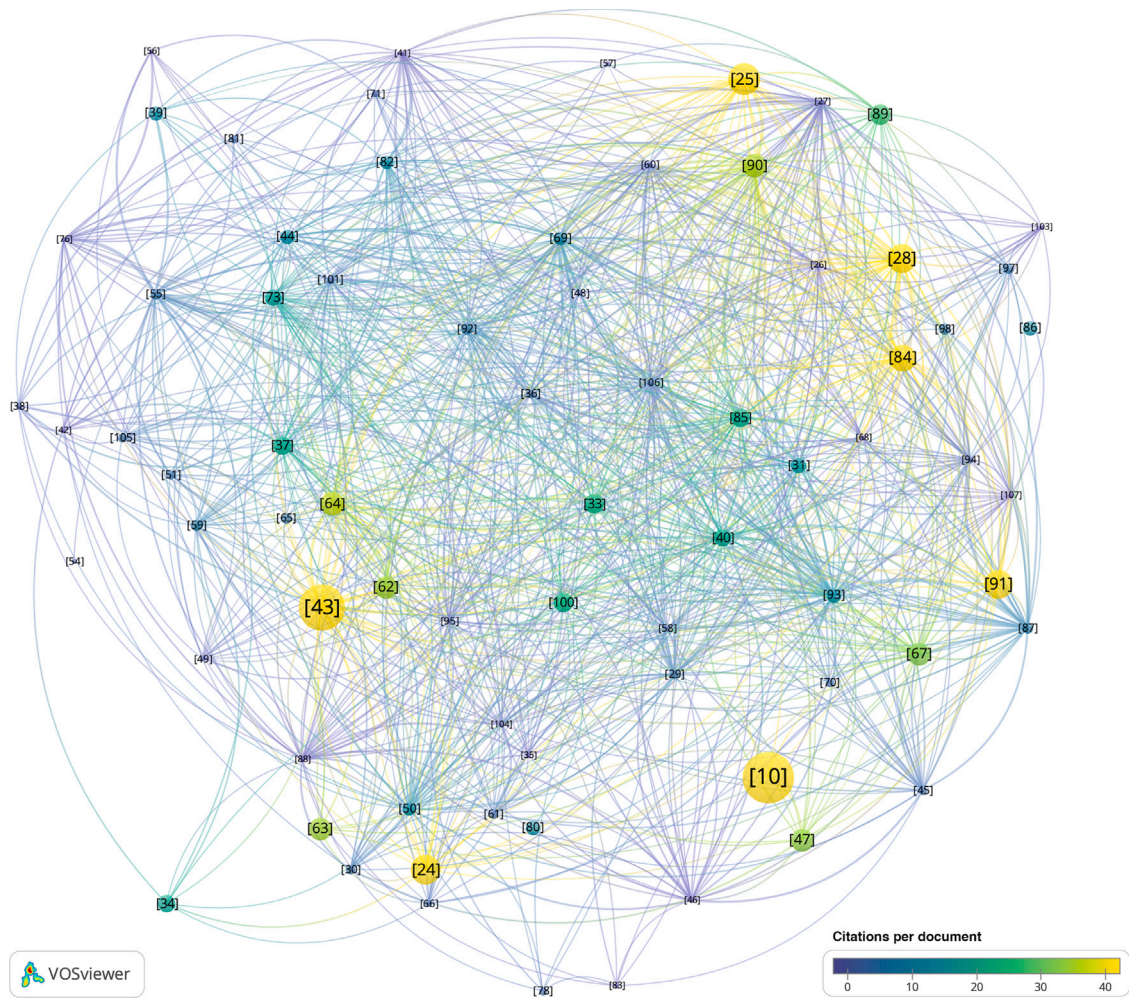
Fig. 8. Included studies (reported in Table 8): citations counts indicating the most cited literature and the cross-reference among them. Node size depends on the number of citations and the connecting lines between them indicate the reference in the bibliography.

References

[1] Y. LeCun, Y. Bengio, G. Hinton, Deep learning, Nature (521) (2015) 436–444, http://dx.doi.org/10.1038/nature14539.

[2] Z. Wan, K. Swaminathan, P.-Y. Chen, N. Chandramoorthy, A. Raychowdhury, Analyzing and improving resilience and robustness of autonomous systems, in: Proc. Int. Conf. Computer-Aided Design, 2022, pp. 1–9, http://dx.doi.org/10.1145/3508352.3561111.

[3] E. Cheng, Daniel-Mueller-Gritschneder, J. Abraham, P. Bose, A. Buyuktosunoglu, D. Chen, H. Cho, Y. Li, U. Sharif, K. Skadron, M. Stan, U. Schlichtmann, S. Mitra, INVITED: Cross-layer resilience: Challenges, insights, and the road ahead, in: Proc. Design Automation Conference, 2019, pp. 1–4, http://dx.doi.org/10.1145/3316781.3323474.

[4] A. Avizienis, J.-C. Laprie, B. Randell, C. Landwehr, Basic concepts and taxonomy of dependable and secure computing, IEEE Trans. Dependable Secure Comput. 1 (1) (2004) 11–33, http://dx.doi.org/10.1109/TDSC.2004.2.

[5] S. Mittal, A survey on modeling and improving reliability of DNN algorithms and accelerators, J. Syst. Archit. 104 (2020) 101689, http://dx.doi.org/10.1016/j.sysarc.2019.101689.

[6] Y. Ibrahim, H. Wang, J. Liu, J. Wei, L. Chen, P. Rech, K. Adam, G. Guo, Soft errors in DNN accelerators: A comprehensive review, Microelectron. Reliab. 115 (2020) 113969, http://dx.doi.org/10.1016/j.microrel.2020.113969.

[7] A. Ruospo, E. Sanchez, L. Matana Luza, L. Dilillo, M. Traiola, A. Bosio, A survey on deep learning resilience assessment methodologies, Computer 56 (2) (2023) 57–66, http://dx.doi.org/10.1109/MC.2022.3217841.

[8] J.J. Zhang, K. Liu, F. Khalid, M.A. Hanif, S. Rehman, T. Theocharides, A. Artussi, M. Shafique, S. Garg, INVITED: Building robust machine learning systems: Current progress, research challenges, and opportunities, in: Proc. Design Automation Conf., 2019, pp. 1–4, http://dx.doi.org/10.1145/3316781.3323472.

[9] M.A. Hanif, M. Shafique, Dependable deep learning: Towards cost-efficient resilience of deep neural network accelerators against soft errors and permanent faults, in: Proc. Int. Symp. on-Line Testing and Robust System Design, 2020, pp. 1–4, http://dx.doi.org/10.1109/IOLTS50870.2020.9159734.

[10] NVDLA open source project, 2018, URL http://nvdla.org/primer.html, (Accessed: 2024-04-15).

[11] S.S. Mukherjee, C.T. Weaver, J. Emer, S.K. Reinhardt, T. Austin, Measuring architectural vulnerability factors, IEEE Micro 23 (6) (2003) 70–75, http://dx.doi.org/10.1109/MM.2003.1261389.

[12] TensorFlow, https://www.tensorflow.org, (Accessed: 2023-05-05).

[13] S. Imambi, K.B. Prakash, G.R. Kanagachidambaresan, PyTorch, Springer International Publishing, Cham, 2021, pp. 87–104, http://dx.doi.org/10.1007/978-3-030-57077-4_10.

[14] A. Gulli, S. Pal, Deep Learning with Keras, Packt Publishing Ltd, 2017.

[15] DarkNet, https://pjreddie.com/darknet, Accessed: 2023-05-05.

[16] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, T. Darrell, Caffe: Convolutional architecture for fast feature embedding, in: Proc. Int. Conf. Multimedia, 2014, pp. 675–678, http://dx.doi.org/10.1145/2647868.2654889.

[17] NVIDIA, TensorRT, https://developer.nvidia.com/tensorrt, Accessed: 2023-03-10.

[18] NVIDIA, cuDNN, https://developer.nvidia.com/cudnn, Accessed: 2023-05-05.

[19] CEA List, N2D2, 2019, https://github.com/CEA-LIST/N2D2, (Accessed: 2023-03-10).

[20] Y. Umuroglu, N.J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre, K. Vissers, FINN: A framework for fast, scalable binarized neural network inference, in: Proc. Int. Symp. Field-Programmable Gate Arrays, 2017, pp. 65–74, http://dx.doi.org/10.1145/3020078.3021744.

[21] CMSIS-NN, https://www.keil.com/pack/doc/CMSIS/NN/html/index.html, (Accessed: 2023-05-05).

[22] A. Capotondi, M. Rusci, M. Fariselli, L. Benini, CMix-NN: Mixed low-precision CNN library for memory-constrained edge devices, IEEE Trans. Circuits Syst. II: Express Briefs 67 (5) (2020) 871–875, http://dx.doi.org/10.1109/TCSII.2020.2983648.

[23] A. Bosio, P. Bernardi, A. Ruospo, E. Sánchez, A reliability analysis of a deep neural network, in: Proc. Latin American Test Symp., 2019, pp. 1–6, http://dx.doi.org/10.1109/LATW.2019.8704548.

[24] Z. Chen, G. Li, K. Pattabiraman, N. DeBardeleben, BinFI: An efficient fault injector for safety-critical machine learning systems, in: Proc. Int. Conf. High Performance Computing, Networking, Storage and Analysis, 2019, pp. 1–23, http://dx.doi.org/10.1145/3295500.3356177.

[25] N. Narayanan, Z. Chen, B. Fang, G. Li, K. Pattabiraman, N. Debardeleben, Fault injection for TensorFlow applications, IEEE Trans. Dependable Secure Comput. 20 (4) (2023) 2677–2695, http://dx.doi.org/10.1109/TDSC.2022.3175930.

[26] S. Laskar, M.H. Rahman, B. Zhang, G. Li, Characterizing deep learning neural network failures between algorithmic inaccuracy and transient hardware faults, in: Proc. Pacific Rim Int. Symp. Dependable Computing, 2022, pp. 54–67, http://dx.doi.org/10.1109/PRDC55274.2022.00020.

[27] A. Mahmoud, N. Aggarwal, A. Nobbe, J.R.S. Vicarte, S.V. Adve, C.W. Fletcher, I. Frosio, S.K.S. Hari, PyTorchFI: A runtime perturbation tool for DNNs, in: Proc. Int. Conf. Dependable Systems and Networks Workshops, 2020, pp. 284–291, http://dx.doi.org/10.1109/DSN-W50199.2020.00014.

[28] B. Reagen, U. Gupta, L. Pentecost, P. Whatmough, S.K. Lee, N. Mulholland, D. Brooks, G. Wei, Ares: A framework for quantifying the resilience of Deep Neural Networks, in: Proc. Design Automation Conf., 2018, pp. 1–6, http://dx.doi.org/10.1145/3195970.3195997.

[29] L. Ping, J. Tan, K. Yan, SERN: Modeling and analyzing the soft error reliability of convolutional neural networks, in: Proc. Great Lakes Symp. VLSI, 2020, pp. 445–450, http://dx.doi.org/10.1145/3386263.3406938.

[30] A. Ruospo, G. Gavarini, C. de Sio, J. Guerrero, L. Sterpone, M.S. Reorda, E. Sanchez, R. Mariani, J. Aribido, J. Athavale, Assessing convolutional neural networks reliability through statistical fault injections, in: Proc. Design, Automation & Test in Europe Conference & Exhibition, 2023, pp. 1–6, http://dx.doi.org/10.23919/DATE56975.2023.10136998.

[31] B.F. Goldstein, S. Srinivasan, D. Das, K. Banerjee, L. Santiago, V.C. Ferreira, A.S. Nery, S. Kundu, F.M. França, Reliability evaluation of compressed deep learning models, in: Proc. Latin American Symp. Circuits & Systems, 2020, pp. 1–5, http://dx.doi.org/10.1109/LASCAS45839.2020.9069026.

[32] Y.-Y. Tsai, J.-F. Li, Evaluating the impact of fault-tolerance capability of deep neural networks caused by faults, in: Proc. Int. System-on-Chip Conf., 2021, pp. 272–277, http://dx.doi.org/10.1109/SOCC52499.2021.9739383.

[33] M. Sabbagh, C. Gongyex, Y. Fei, Y. Wang, Evaluating fault resiliency of compressed deep neural networks, in: Proc. Int. Conf. Embedded Software and Systems, 2019, pp. 1–7, http://dx.doi.org/10.1109/ICESS.2019.8782505.

[34] A. Ruospo, E. Sanchez, M. Traiola, I. O'Connor, A. Bosio, Investigating data representation for efficient and reliable Convolutional Neural Networks, Microprocess. Microsyst. 86 (2021) 104318, http://dx.doi.org/10.1016/j.micpro.2021.104318.

[35] G. Gavarini, A. Ruospo, E. Sanchez, On the resilience of representative and novel data formats in CNNs, in: Proc. Int. Symp. Defect and Fault Tolerance in VLSI and Nanotechnology Systems, 2023, pp. 1–6, http://dx.doi.org/10.1109/DFT59622.2023.10313551.

[36] Y. Zhang, H. Itsuji, T. Uezono, T. Toba, M. Hashimoto, Estimating vulnerability of all model parameters in DNN with a small number of fault injections, in: Proc. Design, Automation & Test in Europe Conference & Exhibition, 2022, pp. 60–63, http://dx.doi.org/10.23919/DATE54114.2022.9774569.

[37] F. Libano, P. Rech, B. Neuman, J. Leavitt, M. Wirthlin, J. Brunhaver, How reduced data precision and degree of parallelism impact the reliability of convolutional neural networks on FPGAs, IEEE Trans. Nucl. Sci. 68 (5) (2021) 865–872, http://dx.doi.org/10.1109/TNS.2021.3050707.

[38] I. Souvatzoglou, A. Papadimitriou, A. Sari, V. Vlagkoulis, M. Psarakis, Analyzing the single event upset vulnerability of binarized neural networks on SRAM FPGAs, in: Proc. Int. Symp. Defect and Fault Tolerance in VLSI and Nanotechnology Systems, 2021, pp. 1–6, http://dx.doi.org/10.1109/DFT52944.2021.9568280.

[39] H.-B. Wang, Y.-S. Wang, J.-H. Xiao, S.-L. Wang, T.-J. Liang, Impact of single-event upsets on convolutional neural networks in xilinx zynq FPGAs, IEEE Trans. Nucl. Sci. 68 (4) (2021) 394–401, http://dx.doi.org/10.1109/TNS.2021.3062014.

[40] D. Xu, Z. Zhu, C. Liu, Y. Wang, S. Zhao, L. Zhang, H. Liang, H. Li, K.-T. Cheng, Reliability evaluation and analysis of FPGA-based neural network acceleration system, IEEE Trans. Very Large Scale Integr. Syst. 29 (3) (2021) 472–484, http://dx.doi.org/10.1109/TVLSI.2020.3046075.

[41] Z. Gao, S. Gao, Y. Yao, Q. Liu, S. Zeng, G. Ge, Y. Wang, A. Ullah, P. Reviriego, Systematic reliability evaluation of FPGA implemented CNN accelerators, IEEE Trans. Dev. Mater. Reliabil. 23 (1) (2023) 116–126, http://dx.doi.org/10.1109/TDMR.2023.3235767.

[42] H. Tian, Y. Ibrahim, R. Chen, C. Jin, S. Shi, J. Xing, J. Li, L. Chen, Evaluation of SEU impact on convolutional neural networks based on BRAM and CRAM in FPGAs, Microelectron. Reliabil. 144 (2023) 114974, http://dx.doi.org/10.1016/j.microrel.2023.114974.

[43] F. F. dos Santos, P.F. Pimenta, C.B. Lunardi, L. Draghetti, L. Carro, D.R. Kaeli, P. Rech, Analyzing and increasing the reliability of convolutional neural networks on GPUs, IEEE Trans. Reliabil. 68 (2) (2019) 663–677, http://dx.doi.org/10.1109/TR.2018.2878387.

[44] R.L. Rech Junior, S. Malde, C. Cazzaniga, M. Kastriotou, M. Letiche, C. Frost, P. Rech, High energy and thermal neutron sensitivity of google tensor processing units, IEEE Trans. Nucl. Sci. 69 (3) (2022) 567–575, http://dx.doi.org/10.1109/TNS.2022.3142092.

[45] A. Chaudhuri, C.-Y. Chen, J. Talukdar, S. Madala, A.K. Dubey, K. Chakrabarty, Efficient fault-criticality analysis for AI accelerators using a neural twin, in: Proc. Int. Test Conf., 2021, pp. 73–82, http://dx.doi.org/10.1109/ITC50571.2021.00015.

[46] S. Pappalardo, A. Ruospo, I. O'Connor, B. Deveautour, E. Sanchez, A. Bosio, Resilience-performance tradeoff analysis of a deep neural network accelerator, in: Proc Int. Symp. Design and Diagnostics of Electronic Circuits and Systems, 2023, pp. 181–186, http://dx.doi.org/10.1109/DDECS57882.2023.10139704.

[47] Y. He, P. Balaprakash, Y. Li, Fidelity: Efficient resilience analysis framework for deep learning accelerators, in: Proc. Int. Symp. Microarchitecture, 2020, pp. 270–281, http://dx.doi.org/10.1109/MICRO50266.2020.00033.

[48] Y. He, M. Hutton, S. Chan, R. De Gruijl, R. Govindaraju, N. Patil, Y. Li, Understanding and mitigating hardware failures in deep learning training systems, in: Proc. Int. Symp. Computer Architecture, 2023, http://dx.doi.org/10.1145/3579371.3589105.

[49] T. Liu, Y. Fu, X. Xu, W. Yan, A cross-layer fault propagation analysis method for edge intelligence systems deployed with DNNs, J. Syst. Archit. 116 (2021) 102057, http://dx.doi.org/10.1016/j.sysarc.2021.102057.

[50] L. Matana Luza, A. Ruospo, D. Soderstrom, C. Cazzaniga, M. Kastriotou, E. Sanchez, A. Bosio, L. Dilillo, Emulating the effects of radiation-induced soft-errors for the reliability assessment of neural networks, IEEE Trans. Emerg. Top. Comput. 10 (4) (2022) 1867–1882, http://dx.doi.org/10.1109/TETC.2021.3116999.

[51] J.E. Rodriguez Condia, J.-D. Guerrero-Balaguera, F.F. Dos Santos, M.S. Reorda, P. Rech, A multi-level approach to evaluate the impact of GPU permanent faults on CNN's reliability, in: Proc. Int. Test Conf., 2022, pp. 278–287, http://dx.doi.org/10.1109/ITC50671.2022.00036.

[52] O. Villa, M. Stephenson, D. Nellans, S.W. Keckler, NVBit: A dynamic binary instrumentation framework for NVIDIA GPUs, in: Proc. Int. Symp. Microarchitecture, 2019, pp. 372–383, http://dx.doi.org/10.1145/3352460.3358307.

[53] J.E.R. Condia, B. Du, M. Sonza Reorda, L. Sterpone, FlexGripPlus: An improved GPGPU model to support reliability analysis, Microelectron. Reliabil. 109 (2020) 113660, http://dx.doi.org/10.1016/j.microrel.2020.113660.

[54] K. Chen, X. Chen, Y. Zhang, Z. Zhang, A rapid evaluation technology for SEU in convolutional neural network circuits, in: Proc. Int. Conf. Circuits and Systems, 2021, pp. 19–23, http://dx.doi.org/10.1109/ICCS52645.2021.9697197.

[55] C. Bolchini, L. Cassano, A. Miele, A. Toschi, Fast and accurate error simulation for CNNs against soft errors, IEEE Trans. Comput. 72 (4) (2023) 984–997, http://dx.doi.org/10.1109/TC.2022.3184274.

[56] C. Bolchini, L. Cassano, A. Miele, A. Nazzari, D. Passarello, Analyzing the reliability of alternative convolution implementations for deep learning applications, in: Proc. Int. Symp. Defect and Fault Tolerance in VLSI and Nanotechnology Systems, 2023, pp. 1–6, http://dx.doi.org/10.1109/DFT59622.2023.10313558.

[57] J. Hoefer, F. Kempf, T. Hotfilter, F. Kreß, T. Harbaum, J. Becker, SiFI-AI: A fast and flexible RTL fault simulation framework tailored for AI models and accelerators, in: Proc. Great Lakes Symp. VLSI, 2023, pp. 287–292, http://dx.doi.org/10.1145/3583781.3590226.

[58] J. Tan, Q. Wang, K. Yan, X. Wei, X. Fu, Saca-FI: A microarchitecture-level fault injection framework for reliability analysis of systolic array based CNN accelerator, Future Gener. Comput. Syst. 147 (2023) 251–264, http://dx.doi.org/10.1016/j.future.2023.05.009.

[59] C. De Sio, S. Azimi, L. Sterpone, FireNN: Neural networks reliability evaluation on hybrid platforms, IEEE Trans. Emerg. Top. Comput. 10 (2) (2022) 549–563, http://dx.doi.org/10.1109/TETC.2022.3152668.

[60] U.K. Agarwal, A. Chan, K. Pattabiraman, LLTFI: Framework agnostic fault injection for machine learning applications (tools and artifact track), in: Proc. Int. Symp. Software Reliability Engineering, 2022, pp. 286–296, http://dx.doi.org/10.1109/ISSRE55969.2022.00036.

[61] M. Taheri, M. Riazati, M.H. Ahmadilivani, M. Jenihhin, M. Daneshtalab, J. Raik, M. Sjödin, B. Lisper, DeepAxe: A framework for exploration of approximation and reliability trade-offs in DNN accelerators, in: Proc. Int. Symp. Quality Electronic Design, 2023, pp. 1–8, http://dx.doi.org/10.1109/ISQED57927.2023.10129353.

[62] A. Lotfi, S. Hukerikar, K. Balasubramanian, P. Racunas, N. Saxena, R. Bramley, Y. Huang, Resiliency of automotive object detection networks on GPU architectures, in: Proc. Int. Test Conf., 2019, pp. 1–9, http://dx.doi.org/10.1109/ITC44170.2019.9000150.

[63] M.A. Neggaz, I. Alouani, S. Niar, F. Kurdahi, Are CNNs reliable enough for critical applications? an exploratory study, IEEE Des. Test 37 (2) (2019) 76–83, http://dx.doi.org/10.1109/MDAT.2019.2952336.

[64] Y. Ibrahim, H. Wang, M. Bai, Z. Liu, J. Wang, Z. Yang, Z. Chen, Soft error resilience of deep residual networks for object recognition, IEEE Access 8 (2020) 19490–19503.

[65] Y. Ibrahim, H. Wang, K. Adam, Analyzing the reliability of convolutional neural networks on GPUs: GoogLeNet as a case study, in: Proc. Int. Conf. Computing and Information Technology, 2020, pp. 1–6, http://dx.doi.org/10.1109/ICCIT-144147971.2020.9213804.

[66] E. Malekzadeh, N. Rohbani, Z. Lu, M. Ebrahimi, The impact of faults on DNNs: A case study, in: Proc. Int. Symp. Defect and Fault Tolerance in VLSI and Nanotechnology Systems, 2021, pp. 1–6, http://dx.doi.org/10.1109/DFT52944.2021.9568340.

[67] S. Kundu, S. Banerjee, A. Raha, S. Natarajan, K. Basu, Toward functional safety of systolic array-based deep learning hardware accelerators, IEEE Trans. Very Large Scale Integr. (VLSI) Syst. 29 (3) (2021) 485–498.

[68] Y. He, Y. Li, Understanding permanent hardware failures in deep learning training accelerator systems, in: Proc. European Test Symp., 2023, pp. 1–6, http://dx.doi.org/10.1109/ETS56758.2023.10173972.

[69] A. Mahmoud, S.K.S. Hari, C.W. Fletcher, S.V. Adve, C. Sakr, N. Shanbhag, P. Molchanov, M.B. Sullivan, T. Tsai, S.W. Keckler, Optimizing selective protection for CNN resilience, in: Proc. Int. Symp. Software Reliability Engineering, 2021, pp. 127–138, http://dx.doi.org/10.1109/ISSRE52982.2021.00025.

[70] M. Sabih, F. Hannig, J. Teich, Fault-tolerant low-precision DNNs using explainable AI, in: Proc. Int. Conf. Dependable Systems and Networks Workshops, 2021, pp. 166–174, http://dx.doi.org/10.1109/DSN-W52860.2021.00036.

[71] T. Garrett, A.D. George, Improving dependability of onboard deep learning with resilient TensorFlow, in: Proc. Space Computing Conf., 2021, pp. 134–142, http://dx.doi.org/10.1109/SCC49971.2021.00021.

[72] T. Tsai, S.K.S. Hari, M. Sullivan, O. Villa, S.W. Keckler, NVBitFI: Dynamic fault injection for GPUs, in: Proc. Int. Conf. Dependable Systems and Networks, 2021, pp. 25–31, http://dx.doi.org/10.1109/DSN48987.2021.00041.

[73] G. Abich, J. Gava, R. Garibotti, R. Reis, L. Ost, Applying lightweight soft error mitigation techniques to embedded mixed precision Deep Neural Networks, IEEE Trans. Circuits Syst. I: Regular Pap. 68 (11) (2021) 4772–4782.

[74] J. Gava, V. Bandeira, F. Rosa, R. Garibotti, R. Reis, L. Ost, SOFIA: An automated framework for early soft error assessment, identification, and mitigation, J. Syst. Archit. 131 (2022) 102710.

[75] J. Gava, A. Hanneman, G. Abich, R. Garibotti, S. Cuenca-Asensi, R.P. Bastos, R. Reis, L. Ost, A lightweight mitigation technique for resource-constrained devices executing DNN inference models under neutron radiation, IEEE Trans. Nucl. Sci. 70 (8) (2023) 1625–1633.

[76] C. Bolchini, L. Cassano, A. Miele, A. Nazzari, Selective hardening of CNNs based on layer vulnerability estimation, in: Proc. Int. Symp. Defect and Fault Tolerance in VLSI and Nanotechnology Systems, 2022, pp. 1–6, http://dx.doi.org/10.1109/DFT56152.2022.9962339.

[77] M. Biasielli, C. Bolchini, L. Cassano, A. Mazzeo, A. Miele, Approximation-based fault tolerance in image processing applications, IEEE Trans. Emerg. Top. Comput. 10 (2) (2022) 648–661.

[78] A. Ruospo, G. Gavarini, I. Bragaglia, M. Traiola, A. Bosio, E. Sanchez, Selective hardening of critical neurons in deep neural networks, in: Proc. Int. Symp. Design and Diagnostics of Electronic Circuits and Systems, 2022, pp. 136–141, http://dx.doi.org/10.1109/DDECS54261.2022.9770168.

[79] A. Ruospo, E. Sanchez, On the reliability assessment of artificial neural networks running on AI-oriented MPSoCs, Appl. Sci. 11 (14) (2021) 6455.

[80] N. Khoshavi, A. Roohi, C. Broyles, S. Sargolzaei, Y. Bi, D.Z. Pan, SHIELDeNN: Online accelerated framework for fault-tolerant deep neural network architectures, in: Proc. Design Automation Conf., 2020, pp. 1–6, http://dx.doi.org/10.1109/DAC18072.2020.9218697.

[81] T.G. Bertoa, G. Gambardella, N.J. Fraser, M. Blott, J. McAllister, Fault tolerant neural network accelerators with selective TMR, IEEE Des. Test 40 (2) (2023) 67–74, http://dx.doi.org/10.1109/MDAT.2022.3174181.

[82] Z. Gao, H. Zhang, Y. Yao, J. Xiao, S. Zeng, G. Ge, Y. Wang, A. Ullah, P. Reviriego, Soft error tolerant convolutional neural networks on FPGAs with ensemble learning, IEEE Trans. Very Large Scale Integr. Syst. 30 (3) (2022) 291–302.

[83] B. Dong, Z. Wang, W. Chen, C. Chen, Y. Yang, Z. Yu, OR-ML: Enhancing reliability for machine learning accelerator with opportunistic redundancy, in: Proc. Design, Automation & Test in Europe Conference & Exhibition, 2021, pp. 739–742, http://dx.doi.org/10.23919/DATE51398.2021.9474016.

[84] K. Zhao, S. Di, S. Li, X. Liang, Y. Zhai, J. Chen, K. Ouyang, F. Cappello, Z. Chen, FT-CNN: Algorithm-based fault tolerance for convolutional neural networks, IEEE Trans. Parallel Distrib. Syst. 32 (7) (2021) 1677–1689.

[85] S.K.S. Hari, S.W. Sullivan, T. Tsai, S.W. Keckler, Making convolutions resilient via algorithm-based error detection techniques, IEEE Trans. Dependable Secure Comput. 19 (4) (2022) 2546–2558.

[86] J. Kosaian, K. Rashmi, Arithmetic-intensity-guided fault tolerance for neural network inference on GPUs, in: Proc. Int. Conf. High Performance Computing, Networking, Storage and Analysis, 2021, pp. 1–15, http://dx.doi.org/10.1145/3458817.3476184.

[87] E. Ozen, A. Orailoglu, Low-cost error detection in deep neural network accelerators with linear algorithmic checksums, J. Electron. Test. 36 (6) (2020) 703–718.

[88] M. Traiola, A. Kritikakou, O. Sentieys, harDNNing: a machine-learning-based framework for fault tolerance assessment and protection of DNNs, in: Proc. European Test Symp., 2023, pp. 1–6, http://dx.doi.org/10.1109/ETS56758.2023.10174178.

[89] Z. Yan, Y. Shi, W. Liao, M. Hashimoto, X. Zhou, C. Zhuo, When single event upset meets deep neural networks: Observations, explorations, and remedies, in: Proc. Asia and South Pacific Design Automation Conf., 2020, pp. 163–168, http://dx.doi.org/10.1109/ASP-DAC47756.2020.9045134.

[90] Z. Chen, G. Li, K. Pattabiraman, A low-cost fault corrector for Deep Neural Networks through range restriction, in: Proc. Int. Conf. Dependable Systems and Networks, 2021, pp. 1–13, http://dx.doi.org/10.1109/DSN48987.2021.00018.

[91] L.-H. Hoang, M.A. Hanif, M. Shafique, FT-ClipAct: Resilience analysis of deep neural networks and improving their fault tolerance using clipped activation, in: Proc. Design, Automation and Test in Europe Conference & Exhibition, 2020, pp. 1241–1246.

[92] B. Ghavami, M. Sadati, Z. Fang, L. Shannon, FitAct: Error resilient deep neural networks via fine-grained post-trainable ActIvation functions, in: Proc. Design, Automation & Test in Europe Conference & Exhibitione, 2022, pp. 1239–1244, http://dx.doi.org/10.48550/arXiv.2112.13544.

[93] E. Ozen, A. Orailoglu, SNR: Squeezing numerical range defuses bit error vulnerability surface in deep neural networks, ACM Trans. Embed. Comput. Syst. 20 (5s) (2021).

[94] C. Amarnath, M. Mejri, K. Ma, A. Chatterjee, Soft error resilient deep learning systems using neuron gradient statistics, in: Proc. Intl. Symp. on-Line Testing and Robust System Design, 2022, pp. 1–7, http://dx.doi.org/10.1109/IOLTS56730.2022.9897815.

[95] G. Gavarini, D. Stucchi, A. Ruospo, G. Boracchi, E. Sanchez, Open-set recognition: an inexpensive strategy to increase DNN reliability, in: Proc. Intl. Symp. on-Line Testing and Robust System Design, 2022, pp. 1–7, http://dx.doi.org/10.1109/IOLTS56730.2022.9897805.

[96] H. Guan, L. Ning, Z. Lin, X. Shen, H. Zhou, S.-H. Lim, In-place zero-space memory protection for CNN, in: Proc. Int. Conf. Neural Information Processing Systems, 2019, pp. 1–10, http://dx.doi.org/10.5555/3454287.3454802.

[97] S.-S. Lee, J.-S. Yang, Value-aware parity insertion ECC for fault-tolerant deep neural network, in: Proc. Design, Automation & Test in Europe Conference & Exhibition, 2022, pp. 724–729, http://dx.doi.org/10.23919/DATE54114.2022.9774543.

[98] S. Burel, A. Evans, L. Anghel, Zero-overhead protection for CNN weights, in: Proc. Int. Symp. Defect and Fault Tolerance in VLSI and Nanotechnology Systems, 2021, pp. 1–6, http://dx.doi.org/10.1109/DFT52944.2021.9568363.

[99] J.J. Zhang, T. Gu, K. Basu, S. Garg, Analyzing and mitigating the impact of permanent faults on a systolic array based neural network accelerator, in: Proc. VLSI Test Symp., 2018, pp. 1–6, http://dx.doi.org/10.1109/VTS.2018.8368656.

[100] U. Zahid, G. Gambardella, N.J. Fraser, M. Blott, K. Vissers, FAT: Training neural networks for reliable inference under hardware faults, in: Proc. Int. Test Conf., 2020, pp. 1–10, http://dx.doi.org/10.1109/ITC44778.2020.9325249.

[101] N. Cavagnero, F. F. dos Santos, M. Ciccone, G. Averta, T. Tommasi, P. Rech, Transient-fault-aware design and training to enhance DNNs reliability with zero-overhead, in: Proc. Symp. on-Line Testing and Robust System Design, 2022, pp. 1–7, http://dx.doi.org/10.1109/IOLTS56730.2022.9897813.

[102] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, L.-C. Chen, Mobilenetv2: Inverted residuals and linear bottlenecks, in: Proc. Conf. Computer Vision and Pattern Recognition, 2018, pp. 4510–4520.

[103] A.M. Buldu, A. Sen, K. Swaminathan, B. Kahne, MBET: Resilience improvement method for DNNs, in: Proc. Int. Conf. Artificial Intelligence Testing, 2022, pp. 72–78, http://dx.doi.org/10.1109/AITest55621.2022.00019.

[104] A. Siddique, K.A. Hoque, Exposing reliability degradation and mitigation in approximate DNNs under permanent faults, IEEE Trans. Very Large Scale Integr. (VLSI) Syst. 31 (4) (2023) 555–566.

[105] Z. Liu, Y. Liu, Z. Chen, G. Guo, H. Wang, Analyzing and increasing soft error resilience of Deep Neural Networks on ARM processors, Microelectron. Reliabil. 124 (2021) 114331, 1:11.

[106] N.-C. Huang, M.-S. Yang, Y.-C. Chang, K.-C. Wu, Decomposable architecture and fault mitigation methodology for deep learning accelerators, in: Proc. Int. Symp. Quality Electronic Design, 2023, pp. 1–8, http://dx.doi.org/10.1109/ISQED57927.2023.10129283.

[107] Z. Zhang, L. Huang, R. Huang, W. Xu, D.S. Katz, Quantifying the impact of memory errors in deep learning, in: Proc. Int. Conf. Cluster Computing, 2019, pp. 1–12, http://dx.doi.org/10.1109/CLUSTER.2019.8890989.

[108] N. van Eck, L. Waltman, Software survey: VOSviewer, a computer program for bibliometric mapping, Scientometrics 84 (2010) 523–538.