A Survey on Deep Learning Resilience Assessment Methodologies

Annachiara Ruospo, Ernesto Sanchez, Lucas Matana Luza, Luigi Dilillo, Marcello Traiola, Alberto Bosio

Title: A Survey on Deep Learning Resilience Assessment Methodologies

Author(s): Annachiara Ruospo, Ernesto Sanchez, Lucas Matana Luza, Luigi Dilillo, Marcello Traiola, and Alberto Bosio

Document version: Post-print version (Final draft)

Please cite the original version:
A. Ruospo et al., ”A Survey on Deep Learning Resilience Assessment Methodologies,” in Computer, 2022.

A Survey on Deep Learning Resilience Assessment Methodologies

Annachiara Ruospo, Ernesto Sanchez
Politecnico di Torino, Dipartimento di Automatica ed Informatica, Italy

Lucas Matana Luza, Luigi Dilillo
LIRMM, Univ Montpellier, CNRS, Montpellier, France

Marcello Traiola
University of Rennes, Inria, CNRS, IRISA, UMR6074, France

Alberto Bosio
Univ Lyon, ECL, INSA Lyon, CNRS, UCBL, CPE Lyon, INL, UMR5270, France

***Abstract*—Deep Learning (DL) applications are gaining increasing interest in the industry and academia for their outstanding computational capabilities. Indeed, they have found successful applications in various areas and domains such as avionics, robotics, automotive, medical wearable devices, gaming; some have been labeled as safety-critical, as system failures can compromise human life. Consequently, DL reliability is becoming a growing concern, and efficient reliability assessment approaches are required to meet safety constraints. This paper presents a survey of the main DL reliability assessment methodologies, focusing mainly on Fault Injection (FI) techniques used to evaluate the DL resilience. The article describes some of the most representative state-of-the-art academic and industrial works describing FI methodologies at different levels of abstraction. Finally, a discussion of the advantages and disadvantages of each methodology is proposed to provide valuable guidelines for carrying out safety analyses.**

In the last few decades, Deep Learning (DL) has drastically enhanced the state of the art of applications such as computer vision, object detection, and language translation. For their outstanding computational capabilities, DL architectures, such as Deep Neural Networks (DNNs), have become attractive solutions in safety-critical areas such as avionics, robotics, medical image analysis, and automotive.

For this reason, the research community has shown increasing attention to understanding the resilience of DL models, which is defined as the capability to tolerate the presence of hardware faults.

It is commonly argued that DL models have inherent fault-tolerant properties due to their distributed and parallel structure, and their redundancy due to over-provisioning. Indeed, they can withstand the failure of a limited number of neurons and continue to function properly [1].

Unfortunately, the choice of hardware on which DL applications run has also been shown to have an impact on resilience [2].

In other words, DL resilience must be assessed by examining the entire system stack (hardware and software), making the overall process more complex and costly.

Furthermore, DL models are not 100% ac-

Published by the IEEE Computer Society

curate, which raises an important issue that needs to be considered when addressing the resilience of these systems. As an example, assume a classifier that due to a fault decreases its accuracy to 88% instead of the original 90%. Then, can the results be accepted (considering the device slightly degraded), or not? In other words, the metrics used to assess resilience need to be thoughtfully evaluated. Metrics generally vary depending on the specific DL model and the task it performs (e.g., image classification or segmentation), but they all have in common the goal of measuring the degradation introduced by software or hardware faults.

This article presents a systematic survey of existing methodologies developed for the assessment of DL resilience. Some of the most representative state-of-the-art academic and industrial works are analyzed and discussed. The article proposes a classification of evaluation techniques according to the level of abstraction and the models being evaluated. Moreover, a further contribution of the article is a comparative table indicating the costs and implementation efforts required to use any of the methodologies presented, as well as the main advantages and disadvantages. This latter analysis is intended to highlight their strengths and weaknesses, and to guide future research directions.

BACKGROUND

Deep neural network resilience assessment is intended to evaluate the impact of hardware faults on DNN inference. This is done by artificially injecting faults into the system, performing the inference, and comparing the results with a golden reference. This process is called Fault Injection (FI).

Firstly, it is important to clarify where faults can be injected and at what level of abstraction.

In a DNN-based system, two levels can be identified: the *hardware* level and the *application* level. The former comprises the hardware device that runs the DNN, while the latter includes only the DNN application. In this light, the resilience assessment can target:

1) The application level: the DNN model as a technology-independent software application.
2) The entire system: the DNN model and the hardware architecture.

In the first case (1), a designer might be interested in evaluating the resilience of the DNN, regardless of the target device on which it will be deployed. Therefore, a FI process can target only the units belonging to the DNN model, i.e., neurons and synaptic weights. According to [1], each neuron must be regarded as a single entity that can fail independently of the failure of any other. This also applies to synaptic weights. Errors in artificial neurons may occur in the following elements:

- **Communication channels**: The communication link between two neuronal cells can be disrupted due to faulty interconnections or disturbances.
- **Synaptic weights**: The weights represent the strength of the connection between two neuronal cells.
- **Neuron body**: It constitutes the nucleus of the neuronal cell. An error affecting the neuron body can be distinguished in two categories: *crash* and *byzantine*. In the first case, the neuron completely stops its activity and saturates with positive or negative values. In the second, it transmits arbitrary values.

A FI campaign can mimic the occurrence of these types of errors. Regarding the second case (2), a designer might be interested in evaluating the resilience of the entire system before deploying a given DNN on a final device. In this case, in addition to the types of errors that compromise the DNN model, it is necessary to consider also the physical faults that may affect the hardware.

As previously stated, the hardware selection impacts the resilience of DNN-based systems.

In state-of-the-art architectures, a single processing element (PE) elaborates many neuronal computations, due to the size of current DNNs. This implies that a **single** fault affecting one PE corresponds to **multiple** faulty

neurons. As a consequence, a comprehensive assessment of system resilience can be obtained by also considering the hardware platform running the DNN.

Faults affecting electronic devices can be classified, based on their temporal characteristics, as permanent or transient. The first one is stable over the time and represents irreversible physical damage. The second one, instead, is only active for a short period of time and occurs as a result of external disturbance or abnormal conditions. Based on this classification, the following fault models have been proposed over the years as an abstraction of physical defects in electronic devices:

1) **Stuck-at Faults**: Individual elements of the electronic device are tied to a logical state. In a memory array, for instance, one bit may be stuck at a logical state '1' or '0' and, regardless of the operation, the result of the read will be always the same.
2) **Bit-Flips**: Individual memory elements of the electronic device have undergone a change in their logical state. This unintentional change can be recovered by writing a new value in the affected memory element.

It is fair to state that today, these two fault models are unable to cover the newer fault mechanisms of deep-submicrometer technologies. New fault models are needed to handle delays, open-lines, bridging, and transient pulses.

Nevertheless, it has been demonstrated that stuck-at and bit-flip fault models enable good fault tolerance investigation even at the application level and, because of this, they have been widely employed for DL reliability studies. An error affecting the communication channels of a DNN, for example, can be modeled as a single or multiple stuck-at faults affecting one or more bits of the channel. Similarly, an error in a DNN synaptic weight can be represented by a stuck-at (or bit-flip) fault impacting one or more bits of the synaptic weight. The same reasoning can be applied to represent a crash or byzantine neuron. A neuron can be considered dead if it no longer transmits values: this error can be modeled as a stuck-at-0 at its output. In contrast, a byzantine neuron can be modeled as a stuck-at-*value*.

FAULT INJECTION METHODOLOGIES

Resilience evaluation of DNN models and DNN-based systems can be pursued for various purposes and at different levels: from the application level to silicon measurements on physical devices, such as Application-Specific Integrated Circuits (ASICs), Field Programmable Gate Arrays (FPGAs), and Graphics Processing Units (GPUs).

Significant efforts have been devoted in the last years to propose methodologies that facilitate this task. Most of them rely on specific frameworks that promote the execution of FI campaigns.

In line with a preliminary classification provided in [3], the state-of-the-art FI methodologies fall into one of the following categories:

- **Simulation-based:** The injection process is conducted without relying on the physical device executing the DNN. Moreover, depending on the level of abstraction, they can be further classified.
  - **Software Level:** Injections are performed on a high-level model of the DNN, without considering any details of the actual hardware architecture.
  - **Hardware Level:** Injections are performed on a more accurate model of the DNN that simulates the target hardware architecture. For example, the target can be expressed at the register transfer level (RTL) or at the gate level.
- **Platform-based:** Measurements and analyses are performed directly on a physical device that emulates the final implementation of a design using FPGAs, or on physical platforms that run DNNs, e.g., CPUs and GPUs.
- **Radiation-based:** Reliability assessment is carried out through accelerated radiation test campaigns mimicking external electromagnetic interference, such as the occurrence of ionizing particles, on the actual platform running the DNN.

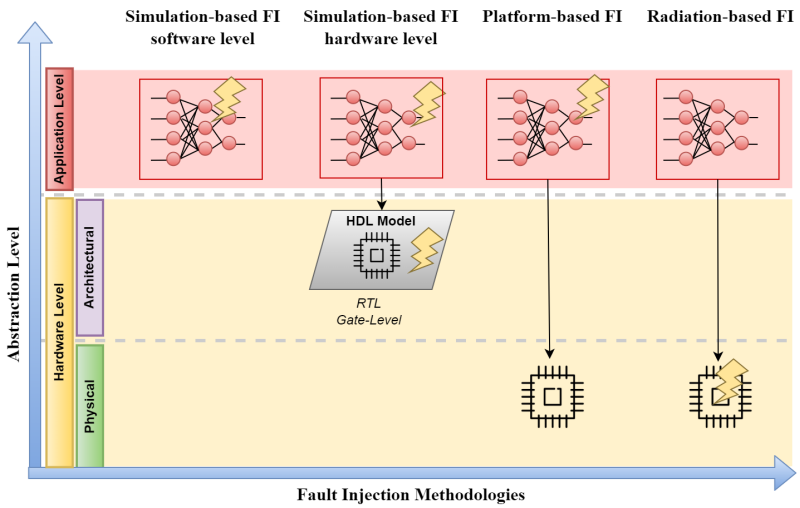

Figure 1. Taxonomy for the FI methodologies developed for the resilience assessment of DNNs and DNN-based systems.

An illustration of the proposed classification is shown in Figure 1. Specifically, for each category, a lightning bolt symbol indicates the level of abstraction and the entities on which FI campaigns can be held. For example, FIs based on hardware-level simulation offer the possibility of injecting faults on both the DNN and HDL hardware models.

Software-Level Simulation-based FIs

Nowadays, simulation-based FIs represent the most widely used techniques. Specifically, software-level approaches are the most frequently adopted as they are cost-effective, faster, more controllable, and easier to implement. A high-level model of the DNN is created which is independent of any potential hardware architecture. This model is then simulated using a FI framework for injecting faults. This allows a general characterization, unrelated to the ultimate hardware platform. A software-level FI is meant to identify weaknesses in the DNN, and to determine the most vulnerable layers, or choose the most reliable data representation. It is worth saying that DNNs are not all equal; they have different architectures, but also different pruning and quantization algorithms [4], which lead to different properties that can impact the reliability assessment. Several works have been advanced in the literature so far [5], [6], [7]. They differ mainly in the used software platforms (e.g., PyTorch or TensorFlow), the type of injected faults (e.g., permanent or transient), and the fault locations (e.g., weights, activation, and operators).

Unfortunately, software methodologies might not be complete to represent real behavior when implemented on a physical device. It is evident that software-level simulations and theoretical analyses may lack the information of the underlying hardware platform, and are relatively less accurate [8].

Hardware-Level Simulation-based FIs

Hardware-level simulations allow the injection of more realistic fault models (i.e., faults that can affect the hardware). The lower the level of abstraction, the higher the injection accuracy. Therefore, for a comprehensive assessment, it might be beneficial to run FIs on a Hardware Description Language (HDL) model of the target hardware architecture running the DNN.

In [8], the authors present a simulation-based hardware-level FI framework to perform in-depth vulnerability analyses of a hardware accelerator described at the RTL. The assessment is performed by considering both application-level specifications (the weights, inputs, and intermediate DNN values) and architectural-level specifications (the specific data representation and the amount of computational re-

sources, i.e., the PEs). Permanent and transient faults are injected during inference cycles on the subset of registers containing the DNN parameters. In [3], the authors exploit the pipeline mechanism to reduce the fault simulation time when executing DNN inferences at the RTL level. Indeed, since DNN layers work independently of each other in a sequence of steps, their parallelism can be exploited to maximize performance. This leads to reducing fault simulation time by more than 60% by mimicking the flow of a pipeline, in which layers perform many inferences in parallel.

To further reduce simulation time, many techniques take advantage of hardware knowledge to derive realistic fault models to be injected later at the software level to take the benefits of both techniques. In [9], the authors model a class of hardware errors in software with high fidelity, exploiting only the high-level design information obtained from architectural descriptions. Another approach consists of modeling the hardware architecture at the software level to expedite the process. In [10], the authors modified an open-source DNN simulator, i.e., Tiny-CNN, to map each line of code to the corresponding hardware component of the DNN accelerators. It is important to mention that the currently available commercial tools for FI, such as Z01X[1], Xcelium[2], and others, can be exploited at the hardware level of the simulation-based approaches. Unfortunately, these tools are mainly oriented to assess the quality of the end of production testing procedures, usually performed at the structural level of the devices. On the other hand, functional FI campaigns, as the ones devoted to assess DNN reliability, are extremely time-consuming and require huge efforts to configure the system, making it very difficult to resort to these tools when assessing the reliability of DNN accelerators.

Platform-based FIs

In this category, the resilience assessment is measured directly on a physical platform that emulates the final implementation using for example FPGAs, or executes the targeted DNN in GPUs and embedded CPUs.

In [11], the DNN workload is executed on a GPU. Injected faults are permanent faults that occur in DNN parameters. At the application level, faults are injected into the weights, activations, and hidden states through bit-flips.

In [12], the authors evaluate the vulnerability of DNNs to permanent and transient faults, exploiting a FI framework and accelerated neutron beam testing. The DNN is a 54-layer model used for the object detection task. The experiments were performed on a Volta GPU using TensorRT, a framework developed by NVIDIA to optimize inferences on GPU architectures. Concerning the emulation on FPGAs, several approaches have been proposed. As an example, FireNN [13] exploits the reconfigurability of FPGAs to mimic faults affecting the hardware running DNNs.

It is important to underline that the effectiveness of simulation-based and platform-based approaches depends also on the designer's configuration choices. For instance, selecting the appropriate number of faults to be injected may affect the succeeding of the experiment. Validating the safety properties by exhaustively fault simulating a DNN is typically prohibitive as the complexity and the size of newer DNN models grow. To address this problem, statistical FI approaches have been proposed over the past decades with the intent of reducing the cost of the fault simulation procedure while still achieving statistically significant results [14]. Clearly, this concern affects simulation-based and platform-based techniques, where the source of errors is not external but stems from designer's decisions.

Radiation-based FIs

The actual implementation of the system is exposed to the same external conditions as the in-field application (e.g., a flux of atmospheric-like neutrons which can induce single-event effects on electronic devices). This methodology guarantees a highly accurate reliability assessment. However, measuring the effects of radiation-induced faults is costly in terms of hardware resources and facility access.

[1] https://www.synopsys.com/verification/simulation/z01x-functional-safety.html

[2] https://www.cadence.com/en_US/home/training/all-courses/86246.html

In [12], irradiation tests were performed on GPUs based on the Volta architecture. The entire GPU was irradiated for the entire duration of the experiment. A second example of a radiation-based method is provided in [15], in which the resilience of a Convolutional Neural Network (CNN) on three different NVIDIA GPU architectures exposed to controlled neutron beams is evaluated. The objective of these works was to understand the propagation of injected errors not only in memory elements, but also in computational and control resources.

Furthermore, the authors in [16] studied the impact of neutron irradiation on the HyperRAM memory, which stored the weights of the CNN-based application. In this way, the source of error was isolated and the focus was on the CNN weights. A variety of error types were identified: single bit upsets, stuck-at faults, and block errors. A similar approach is proposed in [17], in which the authors evaluate the use of 2-D and 3-D Flash memories to store DNN weights.

DISCUSSION

The previous sections highlighted the main characteristics of FI methodologies. This section is intended to provide a qualitative comparison between them, and discuss their main advantages and disadvantages.

In Table 1, for each FI methodology, the following parameters are discussed and graded either as *Low*, *Medium*, or *High*:

- **Cost**: The total costs required to perform the reliability assessment, including time and resources;
- **Development Effort**: The effort required to develop and set up the FI methodology at that specific level of abstraction;
- **Exactness**: How close the FI procedure is to reality, i.e., the results are accurate.
- **Controllability**: The ability to control where and when a fault is injected;
- **Observability**: The ability to identify internal events within the system circuit (not only primary outputs).
- **Repeatability**: How many times the FI process can be repeated using the same framework.
- **Early Availability**: If reliability assessment is performed early in the design cycle.
- **Fault Injection Time**: The time required to perform a single cycle of injection.

In terms of costs, simulation-based methodologies are the least expensive because they do not require the development and purchase of specific electronic devices to run the tests. Hence, the cost level is *Low*, for both FIs based on hardware-level simulation and those based on software-level simulation.

Contrarily, platform-based techniques have a *Medium* cost, requiring the use of validation or emulation devices such as GPUs, CPUs, and FPGAs. Once purchased, the advantage is that they can be reused after FI campaigns. Moreover, they can be parallelized to increase performance. The most expensive techniques are undoubtedly radiation-based techniques for two main reasons: first, access to the irradiation facility; second, the setup development. Moreover, in some cases the electronics exposed to radiation can be irremediably degraded, inhibiting their reuse. The cost level is *High*.

Radiation-based techniques do not involve the creation of a specific FI environment. However, the identification/creation of observation means may not be trivial. The highest development effort is required for platform-based FI methodologies, where the developer must both build a FI environment and configure the platform on which experiments will run (e.g., reconfigure for FPGA emulations). For software-level simulation-based techniques, a medium to high development effort is required to build the FI framework. Likewise for those at the hardware level, where the HDL model of the device to be tested is required.

The exactness of the results varies and depends on how closely these FI techniques mimic the occurrence of realistic defects in the system, and how close they are to reality. In other words, realistic fault models with failure rates according to the environment and systems characteristics will have a direct impact on the exactness of the FI procedure. The highest level of exactness is achieved with radiation-based FIs, where radiation-induced

Table 1. Comparison among the different fault injection methodologies.

Metric	Simulation-based Software Level	Simulation-based Hardware Level	Platform-based	Radiation-based
Cost	Low	Low	Medium-High	High
Development Effort	Medium-High	Medium-High	High	Low-Medium
Exactness[a]	Low	Medium-High	Low-Medium	Very High
Controllability	High	High	Medium	Low
Observability	High	Medium[b]	Low[b]	Low[b]
Repeatability	High	High	High	Medium-Low
Early Availability	High	Medium	High	Medium-Low
Fault Injection Time	Low	High	Medium-Low	Low
Principal Advantages	Cheap & Fast	Good FI Exactness	Portability	Best FI Exactness Realistic
Principal Drawbacks	Low FI Exactness	Time-consuming The HDL must be available	Limited FI Exactness	Expensive

[a] Closeness to reality.
[b] The observability depends on the complexity of the hardware, which is used for the implementation of the FI process.

faults directly affect the silicon implementation of the device. This allows for a really accurate characterization of the DNN model. Right after this category, simulation-based hardware-level FIs are characterized by a good level of precision, which can be close to the final silicon implementation. Indeed, by adopting the HDL model of the hardware device (RTL or gate-level), they can be credited with a medium to high level of exactness.

In contrast, a different reasoning must be made for simulation-based software-level FIs and platform-based FIs: they present a *Low* and a *Low-Medium* level of exactness, respectively. The lower the level of programming language adopted for DNN applications, the higher the exactness. When injecting errors at the software or algorithmic level, the occurrence of realistic hardware faults is reproduced with specific software fault models. A FI framework and a DNN model developed in C/C++ can be compiled and executed directly on a physical hardware device. Therefore, the injected software errors can be close to the faults they seek to reproduce. Examples are the FI frameworks described in [5], [10]. However, this is not true for FI frameworks that inject algorithmic-level errors into high-level programming languages or tools, such as Python, PyTorch, and TensorFlow. Indeed, they are subject to a more complex compilation chain. A great number of works exploit such tools in their deployment; examples of FI frameworks that inject errors at this level are described in [6], [7], [9], [11], [12].

It is worth underlining that although the FI frameworks in [3] and [13] make use of high-level programming languages, they do not inject errors into the high-level DNN model. Rather, they corrupt the RTL register/signals or bits in the FPGAs configuration memory bitstream, respectively. As discussed previously, this leads to a higher level of exactness.

Noteworthy, one of the advantages of conducting resilience assessments based on software-level simulations is the ability to characterize the vulnerability of the DNN independently of the target hardware device and, in particular, to conduct analyses on layers, data types, and network parameters. Nevertheless, when a more comprehensive resilience assessment is needed, injection campaigns should also address the target hardware running the DNN. This is possible when the device's HDL model is available ([3], [8]) either at the RTL or gate level. In this case, hardware-level FIs can achieve better accuracy of results, closer to the implementation on silicon. The main drawback is simulation time. RTL (or gate-level) simulations are known to be time-consuming, due to the complexity of the HDL and DNN models. For example, a small CNN with only seven

layers simulated at RTL can take about 25 minutes to perform a single inference [3]. Furthermore, existing commercial simulation tools are neither tuned nor optimized to deal with the complexity of state-of-the-art DNN applications performing billions of neuronal computations. This means that a hardware-level FI is accurate but very expensive in terms of simulation time. Indeed, hardware-level resilience assessments typically consider only DNNs of limited size: a 6-layer fully connected classifier in [8], or a 7-layer CNN in [3]. Conversely, software-level simulation-based methodologies are not concerned with this non-negligible limitation. Finally, when the device's HDL model is not available, the FI framework can extract architectural details and inject software errors that closely mimic realistic physical hardware faults, as shown in [9] and [10].

In terms of controllability and observability parameters, the FI methodologies described exhibit different degrees. The highest controllability is obtained in simulation (at the software or hardware level), where the developer can create an accurate FI framework for this purpose. Conversely, the level of observability in simulation is not always high: while at the software level it is possible to observe the internal events of the system, at the hardware level this task may prove more challenging. Depending on the complexity of the HDL model of the hardware, observing all internal states can be very difficult. For platform-based FI methodologies, it may be straightforward to control where and when to inject errors, but as outlined before, the FI frameworks can be subject to a specific compilation chain, which could lead to less controllability and visibility. Observability is also reduced for this category. Observing changes in the internal states of the hardware can be very complicated, and also depends on its complexity. Usually, in these cases, the output signals are observed. The same reasoning applies to the observability parameter of radiation-based FI approaches. However, the level of controllability is drastically reduced. Indeed, since the source of errors is external, the developer is unable to control when and where faults are introduced.

Software-level simulation-based, hardware-level simulation-based, and platform-based FI methodologies are characterized by a high level of repeatability: indeed, the FI procedure can be repeated several times using the same injection framework (probably just by adjusting a few configuration parameter). For radiation-based techniques, this is unrealizable. After the irradiation test, the devices, most of the time, cannot be reused.

Another important metric to consider is early availability. Simulation-based software-level FIs can be run early in the design cycle, without relying on the availability of a HDL model. Platform-based FIs can also be performed by emulating or running the DNN on an FPGA, CPU, or GPU, which may or (in most cases) may not represent the final platform on which the DNN is to be deployed. Finally, if the radiation test targets the complete DNN-based system, the FI procedure should be executed at the end of the full design cycle on the final silicon implementation.

To conclude, it is worth noting that it was difficult to precisely compare the time required to run a single FI among all existing FI methodologies. Indeed, there are many variables that determine the FI execution time, such as the parallelization of experiments, the instruments adopted, and the specific radiation source. Another influential factor is the number of fault injections performed: as mentioned before, exhaustive simulation of DNN faults is typically beyond computational possibilities. Therefore, statistical inferences are commonly performed to reduce complexity by injecting a reduced number of faults while still obtaining statistically significant results.

Open Challenges and Future Directions

This article discusses some of the most representative works proposing FI methodologies, highlighting advantages and disadvantages. In this subsection, the main open issues that need research and innovation are underlined.

In a recent publication [18], the authors demonstrated that systematic FIs in the configuration memory of SRAM-based FPGAs could not be generalized to all devices of that type. Exper-

imental analyses conducted on sixteen Xilinx Artix-7 and ten Lattice iCE40 showed that results vary from device to device, and that temperature influences the FI results. This means that parallelization of the FI procedure may yield inaccurate results.

Moreover, the study of different fault models (e.g., delays, bridging, open-lines), covering the new fault mechanisms of deep-submicrometer technologies is an open challenge. Indeed, only the effects of transient and permanent faults have been investigated in this topic.

Finally, because all FI methodologies have advantages and disadvantages, the research community is pushing for hybrid solutions that can get the best out of each approach. For example, a very recent approach [19] proposes a simulation-based cross-layer framework for the reliability analysis of CNN-based applications against soft errors in GPUs. It combines the high exactness of hardware-level FIs with the low fault injection time of software-level FIs.

Hybrid FIs have already been used for resilience assessment of generic applications, such as the FI tool based on simulation and emulation cooperation presented in [20]. We believe that for future works, Table 1 could serve as a basis for researchers to select the best hybrid strategy that considers all the metrics examined.

CONCLUSION

In recent years, there has been a growing interest in studying the resilience of DNN-based systems. This article presents a review of the main methodologies used for this purpose. The reviewed works have been classified into four main categories, namely software-level simulation-based, hardware-level simulation based, platform-based, and radiation-based FI methodologies, depending on the level of abstraction, the source of errors (internal or external), and the overall injection procedure. Their main characteristics are highlighted and qualitatively compared with the aim of providing a guideline for all those who want to examine the resilience of their DNN-based systems. The literature study highlights challenges and open research issues. For instance, in our long-term view, there is a need for more accurate FI metrics, tools, and statistic-based methodologies able to handle the non-negligible complexity of modern DNNs. The proposed discussion is meant to serve as a basis for advanced research in this area.

ACKNOWLEDGMENT

This study has been achieved thanks to the financial support of the project ANR RETRUSTING (Contract No. ANR-21-CE24-0015).

REFERENCES

1. E. M. El Mhamdi and R. Guerraoui, “When neurons fail,” in *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. Orlando, FL, USA: IEEE, 2017, pp. 1028–1037.
2. V. Piuri, “Analysis of fault tolerance in artificial neural networks,” *Journal of Parallel and Distributed Computing*, vol. 61, no. 1, pp. 18 – 48, Jan. 2001.
3. A. Ruospo, A. Balaara, A. Bosio, and E. Sanchez, “A pipelined multi-level fault injector for deep neural networks,” in *2020 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*. Frascati, Italy: IEEE, 2020, pp. 1–6.
4. T. Liang, J. Glossner, L. Wang, S. Shi, and X. Zhang, “Pruning and quantization for deep neural network acceleration: A survey,” *Neurocomputing*, vol. 461, pp. 370–403, 2021.
5. A. Ruospo, E. Sanchez, M. Traiola, I. O’Connor, and A. Bosio, “Investigating data representation for efficient and reliable convolutional neural networks,” *Microprocessors and Microsystems*, vol. 86, p. 104318, 2021.
6. Z. Chen *et al.*, “Tensorfi: A flexible fault injection framework for tensorflow applications,” in *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*. Coimbra, Portugal: IEEE, Oct. 2020, pp. 426–435.
7. A. Mahmoud *et al.*, “PyTorchFI: A runtime perturbation tool for DNNs,” in *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*. IEEE, 2020, pp. 25–31.
8. B. Salami, O. S. Unsal, and A. C. Kestelman, “On the resilience of RTL NN accelerators: Fault characterization and mitigation,” in *2018 30th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*. Lyon, France: IEEE, 2018, pp. 322–329.

9. Y. He, P. Balaprakash, and Y. Li, “Fidelity: Efficient resilience analysis framework for deep learning accelerators,” in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. Athens, Greece: IEEE, 2020, pp. 270–281.
10. G. Li *et al.*, “Understanding error propagation in deep learning neural network (DNN) accelerators and applications,” in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 2017, pp. 1–12.
11. B. Reagen *et al.*, “Ares: A framework for quantifying the resilience of deep neural networks,” in *Proceedings of the 55th Annual Design Automation Conference*. ACM, 2018, pp. 1–6.
12. A. Lotfi *et al.*, “Resiliency of automotive object detection networks on GPU architectures,” in *2019 IEEE International Test Conference (ITC)*. IEEE, 2019, pp. 1–9.
13. C. De Sio, S. Azimi, and L. Sterpone, “An emulation platform for evaluating the reliability of deep neural networks,” in *2020 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*. IEEE, 2020, pp. 1–4.
14. R. Leveugle, A. Calvez, P. Maistri, and P. Vanhauwaert, “Statistical fault injection: Quantified error and confidence,” in *2009 Design, Automation Test in Europe Conference Exhibition*, 2009, pp. 502–506.
15. F. Fernandes dos Santos *et al.*, “Evaluation and mitigation of soft-errors in neural network-based object detection in three GPU architectures,” in *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*. IEEE, 2017, pp. 169–176.
16. L. Matana Luza *et al.*, “Emulating the effects of radiation-induced soft-errors for the reliability assessment of neural networks,” *IEEE Transactions on Emerging Topics in Computing*, pp. 1–1, 2021.
17. M. M. Hasan, M. Raquibuzzaman, I. Chatterjee, and B. Ray, “Radiation tolerance of 3-d nand flash based neuromorphic computing system,” in *2020 IEEE International Reliability Physics Symposium (IRPS)*. IEEE, 2020, pp. 1–4.
18. C. Fibich, R. Obermaisser, and M. Horauer, “Device- and temperature dependency of systematic fault injection results in Artix-7 and iCE40 FPGAs,” in *2021 Design, Automation Test in Europe Conference Exhibition (DATE)*. IEEE, 2021, pp. 1600–1605.
19. C. Bolchini, L. Cassano, A. Miele, and A. Toschi, “Fast and accurate error simulation for cnns against soft errors,” *IEEE Transactions on Computers*, pp. 1–14, 2022.
20. A. Ejlali *et al.*, “A hybrid fault injection approach based on simulation and emulation co-operation,” in *2003 International Conference on Dependable Systems and Networks, DSN 2003. Proceedings*. IEEE, 2003, pp. 479–488.