Dependability of Alternative Computing Paradigms for Machine Learning: hype or hope?

Cristiana Bolchini[1], Alberto Bosio[2], Luca Cassano[1], Bastien Deveautour[2]
Giorgio Di Natale[3], Antonio Miele[1], Ian O'Connor[2], Ioana Vatajelu[3]
[1]*Politecnico di Milano, Dip. di Elettronica, Informazione e Bioingegneria, Milano, Italy*
[2]*Univ Lyon, ECL, INSA Lyon, CNRS, UCBL, CPE Lyon, INL, UMR5270, 69130 Ecully, France*
[3]*TIMA - CNRS / Université Grenoble Alpes, UMR 5159, Grenoble, France*
Email: cristiana.bolchini@polimi.it, alberto.bosio@ec-lyon.fr, luca.cassano@polimi.it,
bastien.deveautour@cpe.fr, giorgio.di-natale@univ-grenoble-alpes.fr,
antonio.miele@polimi.it, ian.oconnor@ec-lyon.fr, ioana.vatajelu@univ-grenoble-alpes.fr

***Abstract*—Today we observe amazing performance achieved by Machine Learning (ML); for specific tasks it even surpasses human capabilities. Unfortunately, nothing comes for free: the hidden cost behind ML performance stems from its high complexity in terms of operations to be computed and the involved amount of data. For this reasons, custom Artificial Intelligence hardware accelerators based on alternative computing paradigms are attracting large interest. Such dedicated devices support the energy-hungry data movement, speed of computation, and memory resources that MLs require to realize their full potential. However, when ML is deployed on safety-/mission-critical applications, dependability becomes a concern. This paper presents the state of the art of custom Artificial Intelligence hardware architectures for ML, here Spiking and Convolutional Neural Networks, and shows the best practices to evaluate their dependability.**

***Index Terms*—Machine Learning (ML), ML-specific HW Architectures, Neuromorphic Computing, ML Dependability**

I. Introduction

Machine Learning (ML) has proven to give very good results for many complex tasks and applications, such as object recognition in images/videos, natural language processing, satellite image recognition, robotics, aerospace, smart healthcare, and autonomous driving [1]–[4]. Unfortunately, nothing came for free. Indeed, the hidden cost behind ML performance stems from their high complexity in terms of operations to be computed and the involved amount of data. The direct consequence is the need of high performance computers, long execution time (especially during training) and high power consumption. For example, the famous AlphaGo [4] required 4 to 6 weeks of training executed on a machine composed of 2,000 CPUs and 250 GPUs consuming about 600kW.

The direct consequence is that the computational workload is limited, in particular for embedded platforms, by the two well-known walls of computing architectures: (1) The memory wall due to the increasing gap between processor and memory speeds, and the limited memory bandwidth making the memory access the killer of performance and power for memory access dominated applications; (2) The power wall as the practical power limit for cooling is reached, meaning no further increase in CPU clock speed.

Nowadays, there is intense activity in designing custom Artificial Intelligence hardware accelerators based on alternative computing paradigms to support the energy-hungry data movement, speed of computation, and memory resources that MLs require to realize their full potential [5]–[7]. A promising solution leverages on the Computation in Memory (CiM) paradigm that moves the computation directly inside the memory, reducing thus the need for data transfer between memory and processor [8]. Other solutions leverage on the Neuromorphic computing aiming at reproducing the biological neurons/synapse structure directly in hardware [9], [10].

Independently on the adopted computing architecture and technology, specialized hardware for machine learning is subject to design and fabrication issues, such as: variations in fabrication process parameters, fabrication process defects, latent defects, i.e., defects undetectable at time-zero post-fabrication testing that manifest themselves later in the field of application, silicon ageing, e.g., time-dependent dielectric breakdown, or even environmental stress, such as heat, humidity, vibration, and Single Event Upsets (SEUs) stemming from ionization.

All these problems can lead to performance degradation or operational failures, which in turn can have important consequences, especially for safety-critical systems [11]–[14]. It is thus crucial to determine the reliability of ML applications implemented leveraging on emerging computing paradigms, especially when they are deployed in safety-critical and mission-critical applications, such as robotics, aerospace, smart healthcare, and autonomous driving.

This paper presents the state of the art of custom hardware architectures for ML. In particular, we will focus on two well known Deep Neural Network (DNN) types, the Spiking and the Convolutional Neural Network. For each type of DNN, we analyze the best practice related to their dependability assessment. Section II presents the main concepts of Artificial Intelligence focusing on Spiking and Convolutional Neural Networks, while the Section III is devoted to the state of

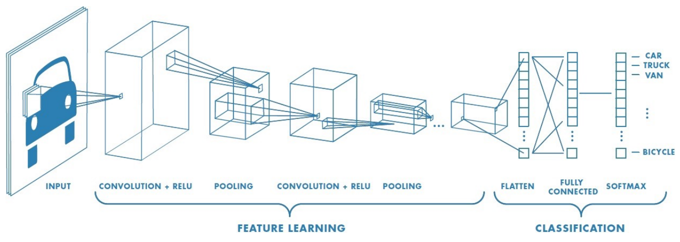


Fig. 1. The typical topology of a Convolutional Neural Network (CNN).

the art of hardware architectures specific to DNN. Section IV presents the methodologies to asses the robustness of DNN. Section V discusses the main challenges and opportunities about the deployment of DNN for safety-critical applications.

II. Background

Artificial intelligence (AI) is indeed a vast scientific field including several disciplines: from biology to computer science. We focus on the computer engineering aspects of AI, in particular the hardware architectures designed to accelerate the inference of AI applications. Let us first present a brief taxonomy of AI adapted from [15]:

- **Artificial Intelligence (AI)**: is the superset of methodologies devoted to the simulation of human intelligence processes by machines;
- **Machine Learning (ML)**: encompasses methods that enable machines to improve with experience;
- **Deep Learning (DL)**: architectures with multiple layers to learn from data sets;
- **Deep Neural Networks (DNN)**: networks composed of multiple layers that mimic the connectivity of the biological neural network;
- **Spiking Neural Networks (SNN)**: a specialized subset of DNN in which information is encoded as "spikes" instead of constant logic "level". SNNs are designed to mimic the real behavior of biological neurons.
- **Convolutional Neural Networks (CNN)**: a specialized subset of DNN in which a certain amount of layers execute convolutions. Those layers aim to easily extract transition-independent features from the input.

A. *Convolutional Neural Network (CNNs)*

A CNN [16] is a Deep Learning model generally employed in image processing and computer vision to carry out a high-end task, such as item classification, object detection and image segmentation. As shown in Figure 1, a CNN is internally organized in a sequence of *layers*, each one processing multidimensional data, dubbed *tensors*, by means of a number of *operators*. A tensor is a multi-dimensional stack of bi-dimensional value matrices, dubbed *feature maps*, that generate a multidimensional grid. The most employed operators can be grouped into the following classes:

- *Convolution*, used to extract features from the input by inferring the appropriate weights, and thus to "learn";
- *Batch normalization*, used to *fix* the data distribution, speeding up the learning process;
- *Activation function*, used to mimic the biological activation of neurons by means of a mathematical function. Common examples of activation functions are the Sigmoid, the softmax and the Rectified Linear Unit (ReLU);
- *Max-pooling* (and other similar operators for dimensionality reduction), used to reduce the size of the tensor thus increasing the degree of generalization;
- *Element-wise operators* used to carry out mathematical operations or single-element manipulation, such as addition, multiplication, exponent and bias addition.

The operators in a CNN are usually organized in a sequence of layers devoted to the feature learning: each operator takes a tensor in input and produces a tensor in output (as shown in the example in left-hand side of Figure 1). The output of the feature learning is a tensor as well; when the goal of the CNN is image segmentation or object identification, such output tensor is considered as the output of the CNN itself. On the other hand, when the CNN is used for classification purposes, such final tensor is first flattened and then fed into a fully-connected Neural Network and a softmax operator that produce the probability values representing the likelihood of the identified object to be classified according to a set of predefined classes (as shown in right-hand side of Figure 1).

Given the complexity of designing CNNs, a number of frameworks, such as TensorFlow [17], Caffe [18], PyTorch [19] and Keras [20], has been proposed. These design frameworks provide general and extensible programming interfaces to specify the structure of a CNN in terms of its dataflow and employed operators. Moreover, these frameworks provide tools to automatically train the model and an automated backend support to target several processing devices such as CPUs or Graphic Processing Units (GPUs) to optimize performance.

B. *Spiking Neural Network (SNNs)*

Artificial neural networks, inspired by the computing capabilities of the biological brain, have been and still are intensely researched. Several electronic substrates are currently studied that offer interesting characteristics to achieve the promises of energy efficiency of the biological model and the technological maturity and the programming capacities expected by the applications. The ambition of neuromorphic chips is to get closer to brain-inspired neuron and synapse computation models. Spiking Neural Networks (SNN) [21] are an important class of bio-inspired computing paradigms offering promising solutions for on-chip cognitive applications. The SNNs incorporate the concept of time in their operation models and they process data encoded in spikes.

Due to their biological plausibility and promise of low power operation, the SNNs have been widely studied both as a new paradigm for neuromorphic computing as well as a replacement for common DNNs. Irrespective of their use, all SNNs require spike-coded signals and spiking neurons.

The literature proposes various methods for numerical-to-spike conversion [22], [23] such as, for instance, the rate-coding which produces a train of spikes, by encoding the numerical value as spike frequency. Since in bio-systems the temporal placement of spikes appears random, a Poisson distribution is usually applied to rate-coding [24]. An other example of numerical-to-spike conversions is the time-coding (also known as latency-coding or time-to-first-spike) [25], [26] where the numerical value is encoded as the delay of a single spike (inversely proportional to the input amplitude).

Different spiking neuron models such as the integrate-and-fire spike response, or Hodgkin-Huxley have been proposed [27]. Typically they integrate currents from arriving spikes and generate new spikes whenever some threshold is crossed. The neuron model must be computationally simple and capable of producing firing patterns. The Hodgkin–Huxley-type models is bio-plausible but computationally prohibitive, while the integrate-and-fire model is computationally effective, but simple and less bio-plausible. When SNN is used to emulate the nervous system, Hodgkin–Huxley-type neurons are preferred, while for machine learning applications, SNNs are using (leaky) integrate and fire neurons.

As previously mentioned, when seeking for power-efficient artificial neural networks, researchers have started considering the possibility of using SNNs in machine learning applications to replace the formal DNNs. In this context, the SNN is mainly used for inference, with the training being done offline (as is the case for formal DNNs). Several training techniqes for SNNs have been proposed, the most common one is the the conversion method trains. The conversion method trains a formal neural network by traditional means (such as back-propagation for instance) and then maps the trained network on SNN [28] by transcoding the synaptic weights taking into account the numerical-to-spike conversion used by the SNN and the neuron model. Another solution is to apply backpropagation (BP) directly to train SNNs. However, the BP process needs all operations through a network to be derivable, while the spiking neuron activation function is usually not. This problem can be solved by introducing a surrogate derivable parameter [29], [30]. However, the networks trained by BP are implemented in hardware only for inference, since a BP learning is prohibitively expensive in hardware. To bring the training closer to the biology, and find means of training the SNN directly on hardware, the Spike-Timing-Dependent-Plasticity (STDP) rule proposed [31] is being considered for low power applications. The STDP is a local learning rule, which updates the synaptic weight as a function of the delay before the input and output spike. This learning rule represents a relation of temporal causality between the input and the output spikes of any given neuron. STDP has been used in unsupervised learning of single layer SNNs [24] or several layers SNNs [25]. It has also been used in supervised learning by adapting the backpropagation to STDP [32].

III. Hardware Architectures for Machine Learning

A. CNN

CNNs were initially implemented in software and executed on general purpose CPUs. In order to accelerate their bandwith, GPU-based implementations have been proposed [33] where specific kernels (i.e., portions of the code) were deployed to the GPU.

The first proposed CNN-specific hardware accelerators were implemented as ASICs [34]–[36], and they achieved orders of magnitude improvements in energy efficiency compared to GPUs. This gain nevertheless comes at the expense of flexibility, with the design cost being very high. FPGAs, on the other hand, provide a good balance between flexibility, design cost, and performance [37], [38].

Independently of the target (ASIC or FPGA), CNN-specific hardware accelerators adopt the same strategy of maximizing data reuse, an element that has been extensively studied by Chen & al. in [35]. The main architectures adopted by re-configurable accelerators such as FPGAs is a dedicated grid of Processing Element (PE) [35], while the main architecture adopted by ASICs are based on more generic systolic arrays [36]. This is mainly because a systolic array is more flexible once designed and can efficiently process matrix products, while a PE array requires tuning some parameters for efficiently execute a DNN (like the number of PEs and the size of the memory bus), making them more suitable for re-configurable accelerators.

B. SNN

As introduced in Section II-B there are issues to consider when looking for the implementation of an SNN, such as the numerical-to-spike coding, the neuron model, and, most importantly the learning rule. There are a plethora of proposals for off-line and on-line learning in SNNs, which opens up the research community to look for hardware solutions for their implementations. In this field, the research has not reached the industry-grade maturity levels and it is mostly developed at academic levels or on industrial prototypes. Today, there are several brain-inspired chips able to simulate numerous spiking neurons to investigate new kinds of computer architectures (SyNAPSE [39], TrueNorth [40], DYNAPs [41], Loihi [42], and Braindrop [43]), or to speed-up neuroscience simulations through the Human Brain Project (SpiNNaker [44] and BrainScaleS [45]). These solutions are mainly dedicated to perform inference tasks. They are designed with distinct technologies, and their working principles and capabilities all differ. Loihi and SpiNNaker are using fully digital, core-based designs, while Braindrop and BrainScaleS are based on mixed or fully analog designs. The SpiNNaker is a many core architecture with optimized spike communication network and programmable local learning, while the BrainScaleS is build with analog neural cores and digital spike communication and it is capable of programmable local learning. Digital neuromorphic prototypes, such as Loihi and TrueNorth deploy a fully digital circuit-based strategy for the neuron and synaptic model parameters and also for learning algorithms, providing a larger flexibility of network configuration. Many academic solutions focus on hybrid and heterogeneous architectures where hardware architectures operate on spike coding (rate, frequency, time) to reach better energy efficiency [46] and to allow local learning rules thanks to the bio-inspired Spike

Time Dependent Plasticity (STDP). Proposed solutions include explorations of state-of-the-art CMOS and emerging nano-electronic technologies capable of mimicking the computational primitives of spiking neural networks. In this context, the most significant improvements come from the utilization of memristive devices to emulate the synaptic plasticity and facilitate local learning [47] [48].

IV. Tools and methodologies for robustness analysis

A. CNN

There is a large literature related to the analysis of the reliability of CNNs, as surveyed in [49]. Several papers (e.g. [50], [51]) analyzed how faults corrupting the weights of the convolutional layers affect the final output of the CNN. Other works (e.g. [52], [53]) presented approaches that trace the propagation of errors through the layers of the CNN to analyze how the application is able to mask them and how the propagated errors affect the produced output. All these works consider faults corrupting both the memory storing the weights and the internal registers of the underlying processing platform. Finally, in [54] the effects of faults are analysed by observing how they affect the precision and recall of the considered object detection application.

Commonly, CNNs are accelerated by exploiting GPUs, therefore tools and methodologies for fault injection in such technology may be exploited to analyse the reliability of CNNs themselves. Several Fault Injection (FI) tools have been proposed to emulate faults in GPUs. GPU-Qin [55] exploits the CUDA-GDB debugger for NVIDIA devices to inject single bit-flips in the registers exposed by the Instruction Set Architecture (ISA); the solution is quite sophisticated and presents a 100x slowdown w.r.t. nominal execution. A similar debugging-based approach is used by CAROL-FI [56] which acts at the source code-level to inject both bit-flips and random values. CAROL-FI presents less than a 5x execution time degradation and the benefit of performing both the injection and the error propagation at the same level of the source code. However, the approach limits the injection sites to the variables. An alternative approach is adopted in SASSIFI [57] and LLFI-GPU [58]: the source code is instrumented for FI before being executed on the GPU. SASSIFI, proposed by NVIDIA, is able to corrupt all registers of the ISA by means of several injection modes, with a 5x slowdown as reported in [56]. LLFI-GPU uses a similar approach while acting at instruction level, claiming a speedup w.r.t. GPU-Qin about 42x on the execution time and similar benefits as CAROL-FI in terms of the analysis of the effects of the errors. Finally, in other works [50], [53], FI is performed again at the software level by manually modifying the application code to integrate the corruption facilities. All these tools are based on complex modification and recompilationof the target code to enable FI, thus leading to considerable performance degradation. More recently, the new NVIDIA tool, called NVBitFI, has been proposed [59]. This tool overcomes several limitations of previous tools by performing a dynamic and selective code instrumentation. On the other hand, implementation issues still persist when working with external libraries.

Working at a higher abstraction level than the architecture-level fault injection would be beneficial for two main reasons: i) to dominate the complexity of CNN applications, and ii) to speed-up and simplify experiments set-up and execution. Several simulation-based approaches have been proposed where *errors* are injected during the execution of the considered CNN. A representative example of error simulation environment for Machine Learning (ML) is TensorFI [60] which is integrated in the commonly employed TensorFlow framework. TensorFI works at the level of the application dataflow and it injects errors by manipulating the output of the operators, thus emulating the effects of faults affecting the ML operators execution. A similar strategy but integrated in other ML frameworks is the one presented in [52], [61], [62], integrated in the Keras, Pytorch and Caffe frameworks, respectively. Finally, a cross-layer approach, implemented in PyTorch, where the CNN is executed via software and the fault injection is carried out by switching down to the Hardware Description Language (HDL) description of the processing unit has been proposed in [63].

The most recent contributions, from which our proposal moves the steps, are Fidelity [64] and BinFi [65]. The former defines a set of error models adherent to what is observed after FI experiments and it then analyses the effects of the faults in terms of the Architectural Vulnerability Factor (AVF). On the other hand, the latter identifies the safety-critical bits in the considered ML application, relying on TensorFI to carry out fault injection in the operators. The effects of the faults on the produced outputs are then analyzed w.r.t. the application, to determine whether they would deeply affect its behavior, or not. Our proposal integrates and merges these two strategies: indeed, we first extract error models, like Fidelity does, and we then perform a classification of the effects of the errors on the produced outputs, like BinFI. Indeed, we believe that our proposal fills the gap between FI, Fuctional Error Simulation (FES) and an application-related resiliency classification.

1) Cross-layer CNN Reliability Analysis: Our proposal is a cross-layer framework for the reliability analysis of CNNs that relies on an error simulation engine. Such engine in turn exploits a set of validated error models previously extracted from a detailed fault injection campaign performed on the operators that compose the considered CNN. Therefore, our proposal bridges the gap between fault injection and error simulation, exploiting the advantages of both approaches.

The first step of the flow is dubbed the "Operators Extractor" that extracts all the operators from the considered CNN. For each identified operator a standalone test program, namely "Experiment Instance", is generated in Caffe. The program transmits input tensors to the GPU, executes the operator on that device, and retrieves the output tensor. These programs represent the code that will be executed in the FI experiments.

The subsequent step of the flow is the actual fault injection. Since we target CNNs accelerated on GPUs, FI environments specifically meant for this processing platforms, e.g., SASSIFI

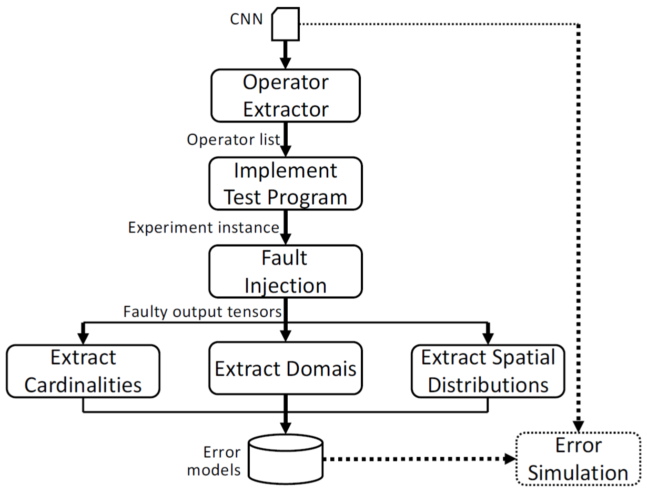


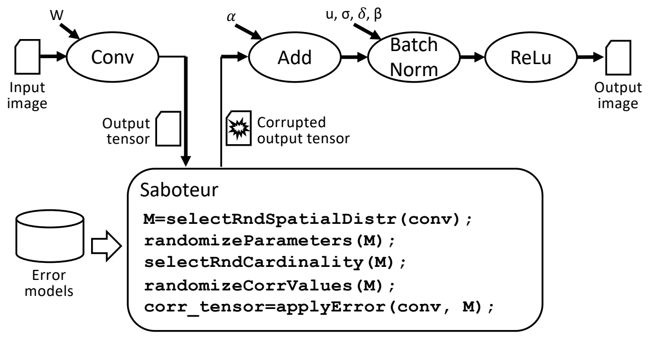
Fig. 2. The designed Fault Injection flow.

or NVBitFI, are suitable for our flow. Due to the complexity of modern GPUs and to the size of the their memory, an exhaustive FI campaign is unfeasible. Nevertheless, it is possible to exploit the extreme regularity of the Single Instruction Multiple Data (SIMD) architecture to reduce the number of required experiments. Indeed, several threads simultaneously executing the same code, elaborate on different bunch of data of the same type and are run on different instances of the same hardware resources. Thus, the output corruption patterns observed when injecting faults during the execution of one of the threads will be representative of the effects of faults in all the threads. To make our FI experiments as general as possible, in each experiment we change the input of the operator and we randomly chose the fault-injected thread.

The outputs of the FI campaign are the collected and compared against the expected output. The correct output are discarded while the erroneous ones are further inspected to identify recurrent error patterns by semi-automated scripts. The goal of this activity is the identification of recurrent corruption patterns in the output tensors, to be exploited for the definition of functional error models. Every time a corruption pattern is statistically relevant and it can be implemented by means of an algorithm, it leads to the definition of an error model. We may identify and classify the corruption patterns according to:

- the cardinality (i.e. the number) of erroneous values in the output tensor,
- the domains the erroneous values belong to, and
- the spatial distribution of the erroneous values.

These three aspects (in particular the last one) are highly influenced by the characteristics of the CNN application, the multidimensionality of the processed data, and the SIMD architecture of the GPU. The result of this modeling activity is the description of the identified error models in terms of an algorithm to reproduce their effects (erroneous values cardinality and domains and spatial pattern of the erroneous values) and the occurrence probability. This detailed information is at the basis of the library of validated error models to support the error simulation frameworks.



```
M=selectRndSpatialDistr(conv);
randomizeParameters(M);
selectRndCardinality(M);
randomizeCorrValues(M);
corr_tensor=applyError(conv, M);
```

Fig. 3. Error simulation exploiting sabouters.

After a repository of error models has been identified, the actual error simulation can be carried out. More in details, we implemented the defined error models in an error simulation engine based on TensorFlow, being it much more popular and versatile than Caffe. However the error models are valid per-se, and can be exploited in any framework. The error injection mechanism has been implemented by means of saboteurs, i.e., new operators inserted in the CNN graph intercepting the output tensor and corrupting it. The previously observed occurrence probabilities of the various error models are used to randomly select the error to inject. Figure 3 depicts the overall scheme on a toy example.

B. SNN

When considering a SNN at a high level of abstraction and without considering the on-line learning (i.e., only during the inference mode), techniques for robustness and reliability assessment developed for CNNs and DNNs can be easily adapted to SNNs, since they work at functional level. Nevertheless, when considering the online learning aspect, it is necessary to develop new methodologies dedicated to the particular properties of the SNN, as the ones proposed in [66] and [67]. In addition, when the hardware implementations are based on memristive technologies, new defects and fault models have to be considered for an accurate robustness and reliability estimation. Due to the aggressive technology scaling and the introduction of new steps at back-end-of-line for the fabrication of the memristive element, it suffers from fabrication-induced variability. Moreover, due to intrinsic properties of the memristive technologies, they are more susceptible to variations and defects, so there is a need for high quality test and fault tolerance. In addition, because of the different operation modes (analog storage and computing for local learning) compared to traditional digital memories, they require fundamentally new testing schemes. A hardware implementation of an SNN requites architectural co-localization of the processing and memory (non-Von Neumann architecture). The circuits solutions used to implement silicon neurons are application depended, but the vast majority are built with a temporal integration block, a spike generation block, a refractory period mechanism, and a spike adaptation block. Synapses are required to exhibit plasticity (i.e., modulation in their efficacy) and to support online learning algorithms, that manifest in changes in their strengths. Emerging memory devices can be used as synaptic elements thanks to their tunable conductivity, compatibility with advanced CMOS fabrication process, low power consumption, non-volatility and scalability. The synaptic conductance modulation can be emulated using: (i) the analog approach (cumulative decrease and increase of resistance), where multiple resistance states emulate long-term potentiation and depression; or (ii) the binary approach, uses two distinct resistance states per device associated with a probabilistic programming scheme. In order to rigurously analyse the faults that can occur in a Spiking Neural Network (SNN) and the assessment of fault-tolerance quality of such a network, we have developed pertinent fault models and methodologies for conducting a fault injection campaign and have identified scenarios of faulty operation happening before and after the STDP learning [66].

We propose the fault analysis to be performed by two approaches: top-down and bottom-up. In the top-down approach, the correctness of the SNN algorithm is evaluated under the effect of different faults, and the results to be validated by functional evaluation of the SNN architecture. Starting from the behavioral model of the SNN under study, we investigate how the network behaves under different issues such as: defective or dead neuron, defective or dead synapse. We evaluate the functional accuracy of the SNN during inference and learning. The obtained results will answer questions such as: which is more detrimental to the functionality of a Neural Network (NN): defective neuron or defective synapse? How many of these critical components have to fail such that the entire network fails? Which is the more critical defect location: inner or outer layer of a NN? In which state does a certain defect matter most: learning or inference? In the bottom-up approach we evaluate the effect of fabrication-induced defects and variability on the operation of the neuron and synapse and a fault modeling campaign is conducted. The faults are injected at system level and the robustness of the SNN is evaluated. This approach requires a complete analysis of possible defects, defect mapping and defect modeling. It is based on a complete analysis of circuit failure modes and a set of technology-dependent fault models, together with a comprehensive evaluation of the application-dependent spatial and temporal dependencies of the circuit failures the functional modules of the SNN: the neuron and synapse. With this analysis we can develop appropriate fault models based on the technology-specific failure mechanisms and defects as well as the SNN functional module and full network.

V. Challenges and opportunities

The research in this field has received a lot of attention, to offer initial solutions for allowing the adoption of ML in domains where robustness is paramount. However, the complexity of the context motivates towards the development and adoption of solutions that are specifically targeted for this domain, also considering the deep relation between the application and its data. As reported, a lot of effort is being devoted to design and implement hardware accelerators optimized for executing ML-based applications, creating new challenges and opportunities to explore dependability-related solutions tailored for these new building blocks, to enable their adoption in critical application environments. In all these contexts, considering the complexity of hardware, application and data, it will be pivotal to identify the most appropriate abstraction level for both the robustness analysis and its enforcement, supporting the designer in the achievement of a robust system, from the performance and dependability perspectives.

Acknowledgment

This work has been co-founded by the ANR project RE-TRUSTING, ANR-21-CE24-0015, by the ANR project EMINENT, ANR-19-CE24-0001 and by the IRGA 2021 project CAD4RMD.

References

[1] L. Deng *et al.*, “Recent advances in deep learning for speech research at microsoft,” in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE, 2013, pp. 8604–8608.

[2] A. Krizhevsky *et al.*, “Imagenet classification with deep convolutional neural networks,” *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.

[3] C. Chen *et al.*, “Deepdriving: Learning affordance for direct perception in autonomous driving,” in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 2722–2730.

[4] D. Silver *et al.*, “Mastering the game of go with deep neural networks and tree search,” *Nature*, vol. 529, no. 7587, pp. 484–489, Jan 2016. [Online]. Available: http://dx.doi.org/10.1038/nature16961

[5] A. D. Mauro *et al.*, “Always-on 674 w@4gop/s error resilient binary neural networks with aggressive sram voltage scaling on a 22-nm iot end-node,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 67, no. 11, pp. 3905–3918, 2020.

[6] B. Moons *et al.*, “14.5 envision: A 0.26-to-10tops/w subword-parallel dynamic-voltage-accuracy-frequency-scalable convolutional neural network processor in 28nm fdsoi,” in *2017 IEEE International Solid-State Circuits Conference (ISSCC)*, 2017, pp. 246–247.

[7] Y.-H. Chen *et al.*, “Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices,” *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 2, pp. 292–308, 2019.

[8] T.-J. Yang *et al.*, “Design considerations for efficient deep neural networks on processing-in-memory accelerators,” in *2019 IEEE International Electron Devices Meeting (IEDM)*, 2019, pp. 22.1.1–22.1.4.

[9] M. Ezzadeen *et al.*, “Low-overhead implementation of binarized neural networks employing robust 2t2r resistive ram bridges,” in *ESSCIRC 2021 - IEEE 47th European Solid State Circuits Conference (ESSCIRC)*, 2021, pp. 83–86.

[10] D. Markovic *et al.*, “Physics for neuromorphic computing (vol 2, pg 499, 2020),” *NATURE REVIEWS PHYSICS*, vol. 3, no. 9, pp. 671–671, 2021.

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

[12] L. Matanaluza *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.

[13] E.-I. Vatajelu *et al.*, “Special session: Reliability of hardware-implemented spiking neural networks (snn),” in *2019 IEEE 37th VLSI Test Symposium (VTS)*, 2019, pp. 1–8.

[14] T. Spyrou *et al.*, “Reliability Analysis of a Spiking Neural Network Hardware Accelerator,” in *Design, Automation and Test in Europe Conference (DATE)*, Antwerp, Belgium, Mar. 2022. [Online]. Available: https://hal.archives-ouvertes.fr/hal-03501968

[15] O. I. Abiodun *et al.*, “State-of-the-art in artificial neural network applications: A survey,” *Heliyon*, vol. 4, no. 11, p. e00938, nov 2018.

[16] Y. LeCun *et al.*, "Backpropagation applied to handwritten zip code recognition," *Neural computation*, vol. 1, no. 4, pp. 541–551, 1989.

[17] M. Abadi *et al.*, "TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems," http://tensorflow.org/, 2015, (Accessed on 03/20/2020).

[18] Y. Jia *et al.*, "Caffe: Convolutional Architecture for Fast Feature Embedding," in *Proc. Intl. Conf. Multimedia*, 2014, p. 675–678.

[19] A. Paszke *et al.*, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," in *Advances in Neural Information Processing Systems 32*, H. Wallach *et al.*, Eds. Curran Associates, Inc., 2019, pp. 8026–8037.

[20] F. Chollet *et al.*, "Keras," https://keras.io, 2015, (Accessed on 03/27/2020).

[21] W. Maass, "Networks of spiking neurons: The third generation of neural network models," *Neural Networks*, vol. 10, no. 9, pp. 1659–1671, 1997. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0893608097000117

[22] A. A. Arvind Kumar, Stefan Rotter, "Spiking activity propagation in neuronal networks: reconciling different perspectives on neural coding," *Nature Reviews Neuroscience 11*, p. 615–627, 2010.

[23] S. J. T. R Van Rullen, "Rate coding versus temporal order coding: what the retinal ganglion cells tell the visual cortex," *Neural Computation*, p. 1255–1283, 2001.

[24] M. C. Peter U. Diehl, "Unsupervised learning of digit recognition using spike- timing-dependent plasticity," *Frontiers in Computational Neuroscience, 9*, 2015.

[25] S. T. T. M. Saeed Reza Kheradpisheh, Mohammad Ganjtabesh, "Stdp-based spiking deep convolutional neural networks for object recognition," *Neural Networks, 99*, pp. 56–67, 2018.

[26] B. Rueckauer *et al.*, "Conversion of analog to spiking neural networks using sparse temporal coding," in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2018, pp. 1–5.

[27] M. P. T. Pfeil, "Deep learning with spiking neurons: Opportunities and challenges," *Frontiers in Neuroscience, 12*, 2018.

[28] Y. H. M. P. S.-C. L. Bodo Rueckauer, Iulia-Alexandra Lungu, "Conversion of continuous-valued deep networks to efficient event-driven networks for image classification," *Frontiers in Neuroscience*, 2017.

[29] T. D. Jun Haeng Lee *et al.*, "Training deep spiking neural networks using backpropagation," *Frontiers in Neuroscience*, 2016.

[30] E. O. Neftci *et al.*, "Surrogate gradient learning in spiking neural networks: Bringing the power of gradient-based optimization to spiking neural networks," *IEEE Signal Processing Magazine*, vol. 36, no. 6, pp. 51–63, 2019.

[31] Y. D. Natalia Caporale, "Spike timing-dependent plasticity: a hebbian learning rule," *Annu Rev Neurosci.*, 2008.

[32] "Deep learning in spiking neural networks," *Neural Networks*, vol. 111, pp. 47–63, 2019.

[33] A. Guzhva *et al.*, "Multifold acceleration of neural network computations using gpu," in *Artificial Neural Networks – ICANN 2009*, C. Alippi *et al.*, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 373–380.

[34] Y. Chen *et al.*, "Dadiannao: A machine-learning supercomputer," *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 609–622, 2014.

[35] ——, "Eyeriss: a spatial architecture for energy-efficient dataflow for convolutional neural networks," in *CARN*, 2016.

[36] N. P. Jouppi *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, 2017, pp. 1–12.

[37] K. Guo *et al.*, "Angel-eye: A complete design flow for mapping cnn onto customized hardware," in *2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2016, pp. 24–29.

[38] R. Reddy *et al.*, "Dlau: A scalable deep learning accelerator unit on fpga," *International Journal of Research*, vol. 5, pp. 921–928, 2018.

[39] A. S. Cassidy *et al.*, "Cognitive computing building block: A versatile and efficient digital neuron model for neurosynaptic cores," in *The 2013 International Joint Conference on Neural Networks (IJCNN)*, 2013, pp. 1–10.

[40] P. A. Merolla *et al.*, "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science*, vol. 345, no. 6197, pp. 668–673, 2014. [Online]. Available: https://www.science.org/doi/abs/10.1126/science.1254642

[41] S. Moradi *et al.*, "A scalable multicore architecture with heterogeneous memory structures for dynamic neuromorphic asynchronous processors (dynaps)," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 12, no. 1, p. 106–122, Feb 2018. [Online]. Available: http://dx.doi.org/10.1109/TBCAS.2017.2759700

[42] M. Davies *et al.*, "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, 2018.

[43] A. Neckar *et al.*, "Braindrop: A mixed-signal neuromorphic architecture with a dynamical systems-based programming model," *Proceedings of the IEEE*, vol. 107, no. 1, pp. 144–164, 2019.

[44] X. Jin *et al.*, "Modeling spiking neural networks on spinnaker," *Computing in Science Engineering*, vol. 12, no. 5, pp. 91–97, 2010.

[45] "Structural plasticity on an accelerated analog neuromorphic hardware system," *Neural Networks*, vol. 133, pp. 11–20, 2021.

[46] L. Khacef *et al.*, "Confronting machine-learning with neuroscience for neuromorphic architectures design," in *2018 International Joint Conference on Neural Networks (IJCNN)*, 2018, pp. 1–8.

[47] S. H. Jo *et al.*, "Nanoscale memristor device as synapse in neuromorphic systems," *Nano Letters*, vol. 10, no. 4, pp. 1297–1301, 2010, pMID: 20192230. [Online]. Available: https://doi.org/10.1021/nl904092h

[48] A. Sengupta *et al.*, "Magnetic tunnel junction mimics stochastic cortical spiking neurons," *Scientific Reports*, vol. 6, 10 2015.

[49] Y. Ibrahim *et al.*, "Soft errors in DNN accelerators: A comprehensive review," *Microelectronics Reliability*, vol. 115, p. 113969, 2020.

[50] A. Bosio *et al.*, "A Reliability Analysis of a Deep Neural Network," in *Proc. Latin American Test Symp.*, 2019, pp. 1–6.

[51] A. P. Arechiga *et al.*, "The Robustness of Modern Deep Learning Architectures against Single Event Upset Errors," in *Proc. High Performance extreme Computing Conf.*, 2018, pp. 1–6.

[52] B. Reagen *et al.*, "Ares: A framework for quantifying the resilience of deep neural networks," in *Proc. Design Automation Conf.*, 2018.

[53] G. Li *et al.*, "Understanding Error Propagation in Deep Learning Neural Network (DNN) Accelerators and Applications," in *Proc. Intl. Conf. High Performance Computing, Networking, Storage and Analysis*, 2017.

[54] F. Fernandes *et al.*, "Evaluation of Histogram of Oriented Gradients Soft Errors Criticality for Automotive Applications," *ACM Trans. Archit. Code Optim.*, vol. 13, no. 4, 2016.

[55] B. Fang *et al.*, "GPU-Qin: A methodology for evaluating the error resilience of GPGPU applications," in *Proc. Intl. Symp. Performance Analysis of Systems and Software*, 2014, pp. 221–230.

[56] D. Oliveira *et al.*, "Increasing the Efficiency and Efficacy of Selective-Hardening for Parallel Applications," in *Proc. Intl. Symp. Defect and Fault Tolerance in VLSI and Nanotechnology Systems*, 2019, pp. 1–6.

[57] S. Hari *et al.*, "Sassifi: An architecture-level fault injection tool for GPU application resilience evaluation," in *Proc. Intl. Symp. Performance Analysis of Systems and Software*, 2017, pp. 249–258.

[58] G. Li *et al.*, "Understanding error propagation in GPGPU applications," in *Proc. Intl. Conf. High Performance Computing, Networking, Storage and Analysis*, 2016, pp. 240–251.

[59] T. Tsai *et al.*, "NVBitFI: Dynamic Fault Injection for GPUs," in *Proc. Intl. Conf. Dependable Systems and Networks*, 2021, pp. 284–291.

[60] Z. Chen *et al.*, "TensorFI: A Flexible Fault Injection Framework for TensorFlow Applications," in *Proc. Intl. Symp. on Software Reliability Engineering*, 2020, pp. 426–435.

[61] A. Mahmoud *et al.*, "PyTorchFI: A Runtime Perturbation Tool for DNNs," in *Proc. Intl. Conf. Dependable Systems and Networks Workshops*, 2020, pp. 25–31.

[62] M. A. Neggaz *et al.*, "Are CNNs Reliable Enough for Critical Applications? An Exploratory Study," *IEEE Design & Test*, vol. 37, no. 2, pp. 76–83, 2020.

[63] A. Ruospo *et al.*, "A Pipelined Multi-Level Fault Injector for Deep Neural Networks," in *Proc. Intl. Symp. Defect and Fault Tolerance in VLSI and Nanotechnology Systems*, 2020.

[64] Y. He *et al.*, "FIdelity: Efficient Resilience Analysis Framework for Deep Learning Accelerators," in *Proc. Intl. Symp. on Microarchitecture*, 2020, pp. 270–281.

[65] Z. Chen *et al.*, "BinFI: An Efficient Fault Injector for Safety-Critical Machine Learning Systems," in *Proc. Intl. Conf. High Performance Computing, Networking, Storage and Analysis*, 2019, pp. 1–23.

[66] E.-I. Vatajelu *et al.*, "Special session: Reliability of hardware-implemented spiking neural networks (snn)," in *2019 IEEE 37th VLSI Test Symposium (VTS)*, 2019, pp. 1–8.

[67] E. I. Vatajelu *et al.*, "Fully-connected single-layer stt-mtj-based spiking neural network under process variability," in *2017 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*, 2017, pp. 21–26.