Testability and Dependability of AI Hardware: Survey, Trends, Challenges, and Perspectives

Fei Su, Chunsheng Liu, Haralampos-G. Stratigopoulos

Testability and Dependability of AI Hardware: Survey, Trends, Challenges, and Perspectives

Fei Su*, Chunsheng Liu†, Haralampos-G. Stratigopoulos‡
*Intel Coorporation, USA
†Alibaba Inc., USA
‡Sorbonne Université, CNRS, LIP6, France

***Abstract*—In recent years, there has been an expedited trend in embracing bold and radical innovation of computer architectures, aiming at the continuation of computing performance improvement despite the slowed-down physical device scaling. One new frontier in this field focuses on Artificial Intelligence (AI) hardware. While functionality of AI hardware still remains the main focus, testability and dependability of these new architectures need to be addressed before the mainstream adoption. This survey paper covers the state-of-the-art in research and development of dependability and testability solutions for AI hardware including digital or analog implementations of Artificial Neural Networks (ANNs) and Spiking Neural Networks (SNNs), used in accelerators and neuromorphic designs. Trends, challenges and perspectives are also discussed in this paper.**

***Index Terms*—AI hardware accelerators, neuromorphic computing, artificial neural networks, spiking neural networks, fault modeling and simulation, testability, dependability, fault tolerance, functional safety.**

I. Introduction

Artificial Intelligence (AI) and Machine Learning (ML) algorithms have been a subject of interest for several decades now. Although AI and ML have gone through hype cycles of disappointment and enthusiasm, recent algorithmic advancements, in particular Deep Neural Networks (DNNs) [1], as well as the availability of big data and the rapid growth of computing power, have renewed interest leading nowadays to applications in numerous distinct fields, e.g., robotics, medicine, autonomous vehicles, computer vision, speech recognition, natural language processing, gaming, etc.

DNN models are computational intensive with their back-propagation training process taking up a number of operations in the order of millions. Inference on trained models requires a single forward pass, but still the number of operations remains very high. From a hardware perspective, this poses severe challenges of data storage, movement, and processing speed on conventional Central Processing Units (CPUs) with a traditional Von Neumann computer architecture, commonly known as the memory wall problem [2]. To this end, there are intense and on-going efforts nowadays towards designing dedicated and customized processors for AI [3]–[9], referred to as AI hardware accelerators, which belong to the larger family of domain-specific computing paradigms. Widely used AI hardware accelerators today are Graphics Processing Unit (GPUs) and Field-Programmable Gate Arrays (FPGAs), but orders of magnitude of energy-speed improvement can be achieved with Application-Specific Integrated Circuits (ASICs).

Another high incentive for designing AI hardware accelerators is to push the execution of AI algorithms from the cloud closer to the sources of data onto edge devices [10]. This is driven by energy, bandwidth, speed, availability, and privacy requirements. More specifically, edge computing reduces the data transfer requirement thus saving energy and bandwidth. Saving bandwidth is important given the forecast that several tens of billions of edge devices will be connected to the internet in the near future. Several applications, e.g., autonomous vehicles, require low-latency real-time computation which is slowed down due to the communication with the cloud. Also, several applications require availability, thus they need to be less dependent on the communication with the cloud. Finally, handling data locally offers privacy as opposed to transmitting sensitive data over the cloud. Edge AI is a challenging objective since edge devices have limited resources and are often battery-operated. Typically, AI hardware accelerators embedded on edge devices perform only inference with the DNN model trained in software and uploaded upfront.

Having stressed that AI hardware accelerators are pivotal in the AI world, many would believe that neural networks on hardware inherit the remarkable fault tolerance capabilities of the biological brain. Indeed, biological neural networks are capable of regenerating, rewiring or adapting network elements to make up for damage, which is part of their neuroplasticity ability [11]. This assumption also stems from properties of neural networks, such as their high parallelism and over-provisioning, i.e., there are more neurons available than the minimum required for a certain cognitive task and many neurons end up being ineffective. However, as it will be discussed in more detail in Section IV, recent hardware-level fault injection experiments have shown that this assumption is false. A neural network is likely to be capable of learning even in the presence of a high fault rate; however, the impact on prediction accuracy can be non-negligible or even detrimental if a model is uploaded on a faulty hardware neural network or if a fault occurs during the lifetime of the hardware neural network.

For these reasons, testability and dependability of AI hardware accelerators are important issues that need to be addressed already from the design phase [12]. Inspiration can of course be drawn from known and mature methodologies applied to traditional computer architectures, but the architectural particularities of AI hardware accelerators often make such methodologies prohibitive in terms of cost and quality, requiring the development of new methodologies that are better suited and take full advantage of the said architectural particularities.

The aim of this article is to provide a survey of existing works on testability and dependability methodologies for AI hardware accelerators and discuss trends, challenges, and

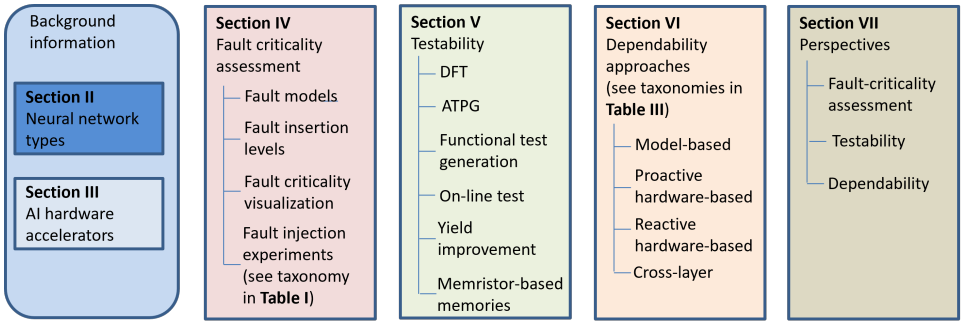


Fig. 1: Structure of article.

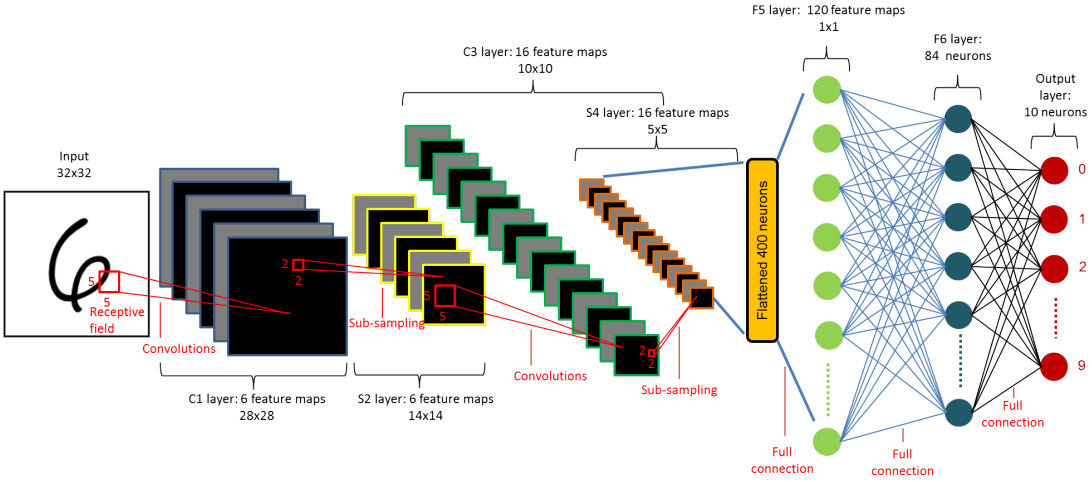


Fig. 2: Architecture of LeNet-5 CNN.

perspectives. The high-level organization of this article into sections and their main subsections is shown in Fig. 1.

II. Neural Network Types

We distinguish two types of neural networks, namely the Artificial Neural Networks (ANNs) and the Spiking Neural Networks (SNNs). Both are inspired from the brain structure composed of layers assembled by neurons and synapses interconnecting the different layers. The term “deep" in DNN refers to the number of layers going beyond of just a few, allowing to extract more complex features. The number of layers, the number of neurons within each layer, and synapse connections define a network topology.

There are three main topologies applied to both ANNs and SNNs, namely Fully-Connected (FC) networks, Convolutional Neural Networks (CNNs) [13], and Recurrent Neural Networks (RNNs) [14]. Fig. 2 shows an example CNN with FC layers forming the last layers. In FC networks, the neurons of a new layer are connected via synapses to the outputs of all neurons in the prior layer. In CNNs, a convolutional layer is composed of several feature maps. A feature map is a plane of neurons where each neuron is connected to the outputs of spatially nearby neurons contained in a lower-dimensional plane of the prior layer, referred to as a receptive field. Each neuron has a different receptive field located at different coordinates of the prior layer. In a given feature map, all neurons are constrained to share the same synaptic weights, whereas synaptic weights change from one feature map to another. Convolutional layers are alternated with sub-sampling layers which are used to down-sample the output of the preceding convolutional layer. There are different types of sub-sampling, such as max pooling and average pooling. Max pooling captures the maximum value of the receptive field and processes it to the output, whereas average pooling calculates the average value. CNNs allow synapse reuse and reduce the number of synapses compared with a FC network. In RNNs, neurons can additionally receive as input their previous state or the previous state of a neuron in a subsequent layer, thus realizing an internal memory retaining past information to forecast future outputs. RNNs are used for learning on time-series or sequential data, while FC networks and CNNs are feed-forward and inputs are independent of each other.

In ANNs, data are represented as static numerical values. Neurons apply a non-linear activation function, such as Rectified Linear Unit (ReLU), sigmoid, and tanh, on the weighted sum of outputs of other neurons, as depicted in Fig. 3. The weights are scalar values and correspond to the synaptic weights.

In SNNs, on the other hand, data are represented with spikes processed in a continuous way in time, which is similar to brain operation. Thus, they are more biologically plausible compared with ANNs, bridging the gap between ML and the biological brain in terms of computation speed and power consumption [15]. SNNs form the basis of neuromorphic computing as pioneered by Carver Mead in the 1980s [16]. The most hardware-friendly spiking neuron implementation

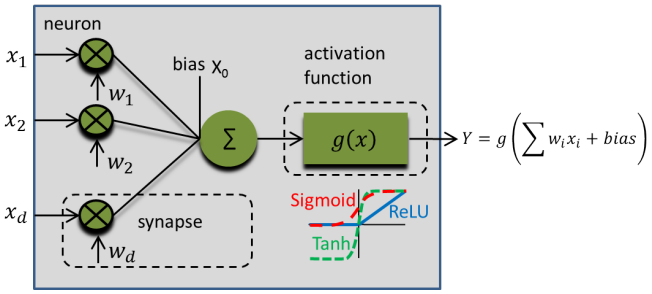


Fig. 3: Artificial neuron.

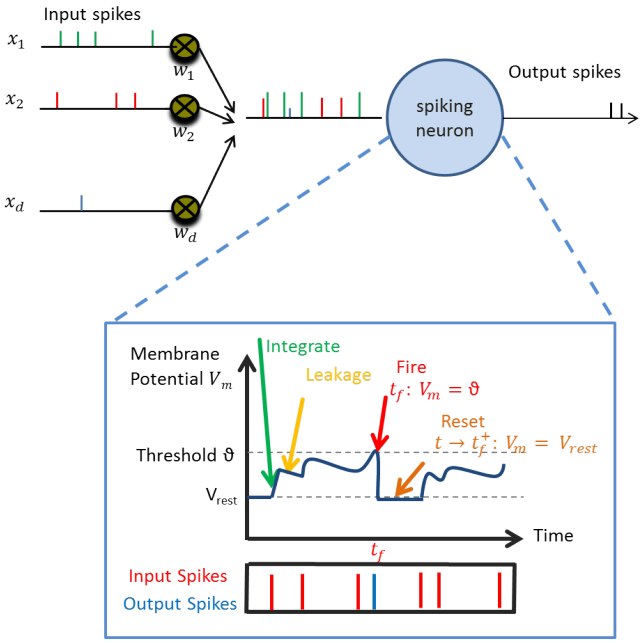


Fig. 4: Spiking neuron.

is the Integrate & Fire (I&F) model [17], depicted in Fig. 4. The neuron integrates the spikes from incoming synapses, and when the potential of its membrane exceeds a threshold it fires a spike of its own that propagates through synapses to other neurons. It also resets the threshold so as to be able to fire again. The neuron has two additional brain-inspired functionalities. It has a refractory period, i.e., it is allowed to fire only if a certain time is elapsed since the last output spike, and a leakage behavior, i.e., the membrane potential decreases between two consecutive input spikes. The synapse operation is different from ANNs and also resembles the biological synapse operation. A synapse receives spikes and in turn stimulates the membrane potential of post-synaptic neurons via a current. The most common information representation in SNNs is rate coding, whereby the information is encoded into the firing rate over an observation period, but other representations have been suggested, including time-to-first-spike and inter-spike interval.

From a hardware perspective, there is a belief that SNNs offer faster inference and lower energy consumption compared with ANNs. This belief stems from two SNN characteristics, namely the real-time asynchronous spike flow and the sparsity of the spike flow which reduces neuron activities. In contrast, ANNs have a frame-based operation, i.e., for a layer to perform its computation the layer has to wait for the computation of the previous layer to complete and every individual neuron is being evaluated. However, SNNs are harder to train compared with ANNs due to the non-continuity of the spiking neuron's transfer function, as well as the additional parameters a spiking neuron carries, e.g., threshold, leakage rate, refractory period, which could be sensitive. In general, the discussion on the relative performance between ANNs and SNNs is not trivial due to the different input type, i.e., sequence of static frames versus continuous-time event flow. Converting a dataset from frame-based to spiking format and vice versa creates bias in the comparison. In general, the advantage of one neural network type over the other is task-dependent, with the SNNs being ideally suited for processing spatio-temporal event-based sensory data. For an extensive discussion on SNNs and the comparison with their ANN counterparts, the readers are referred to [5], [18], [19].

III. AI Hardware Accelerators

Silicon implementations of neural networks appeared decades ago with early efforts demonstrating few-layers, few-neurons per layer networks [20]. Moving to larger designs for DNN acceleration, the main challenge is the memory wall that limits the throughput and increases power consumption. The design ambition is therefore to overcome the memory wall by distributing the memory within close proximity to the Processing Elements (PEs), e.g., the Multiple-Accumulate (MAC) units, or through interleaving of memory and PEs. Basic architectures include the streaming architecture composed of many cores with the layers mapped among the cores and the single-core architecture, i.e., in the form of a systolic array that parallelizes the storage and computation of the different layers [3]. SNNs typically employ the streaming architecture with a core receiving and transmitting spikes via the Address Event Representation (AER) protocol which essentially implements a Network-on-Chip (NoC) communication scheme [21]. Clearly, efficient mapping of the neural network algorithm onto the hardware becomes of utmost importance and different neural network topologies require different hardware designs to fully take advantage of neuromorphic computing.

Analog and Mixed-Signal (AMS) implementations can offer orders of magnitude lower power consumption compared with their digital counterparts, thus they are better-suited for edge computing being capable of acting directly on sensory data from world-machine interfaces [22], [23]. This is because transistors are operated in the sub-threshold region and the main operations of a neural network, i.e., addition and multiplication, can be performed efficiently in the analog domain. Addition can be performed using Kirchhoff's current law while multiplication can be performed with just a few transistors. However they are less robust due to process variations and noise.

One way to reduce the energy consumption is approximate computing which involves two strategies. The first uses approximate arithmetic units in the PEs [24]. The second is termed network compression or quantization [25]. It reduces the precision of the weights and neuron activation values by

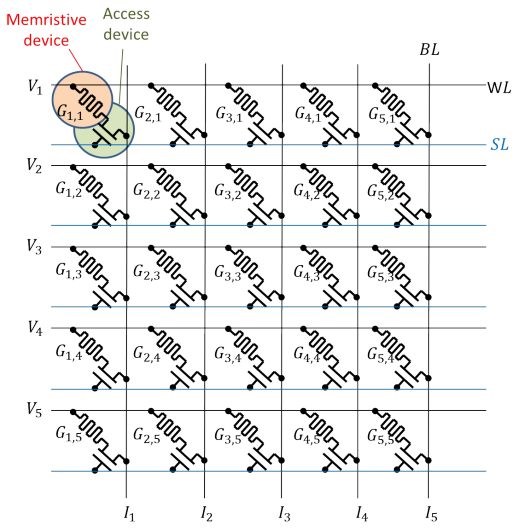


Fig. 5: Memristive crossbar array.

transforming floating point numbers into narrow few-bit integers. At the extreme, this results in Binary Neural Networks (BNNs) that use 1-bit precision [26], further simplifying the network architecture by allowing using XNORs instead of MAC units [27]. BNNs save energy and storage and can serve for implementing deep models in resource-constrained edge devices. Network compression results in accuracy loss but it may be recovered through training.

Another design paradigm with tremendous potential for overcoming the memory wall is in-memory computing where the matrix-vector multiplications are performed within the memory itself [28], [29]. In-memory computing has two main embodiments, namely performing arithmetic and logic operations within the SRAM or using memristive crossbar arrays.

A memristive crossbar array is composed of horizontal and vertical metal lines with a memristive device placed at each cross-point intersection connecting the two metal lines, as shown in Fig. 5. The conductance of the memristive device implements the synapse weight, horizontal lines are driven by the voltage output of pre-synaptic neurons, and vertical lines provide the current input of post-synaptic neurons. Each column implements the dot product $I_i = \sum_j G_{i,j} \cdot V_j$ and parallelized dot-products across the columns implement efficient *in-situ* matrix-vector multiplication $I = \mathbf{G} \cdot V$ in analog form, reducing computational complexity from $\mathcal{O}(n^2)$ to $\mathcal{O}(1)$. Each memristive device is augmented with an access device, as shown in the 1-transistor/1-resistor (1T1R) architecture of Fig. 5, that allows selecting a memristive device for programming while not disturbing the stored state of other memristive devices. A memristive crossbar array is accompanied with peripheral circuits (not shown in Fig. 5) if communication between crossbars is implemented in the digital domain. These include Digital-to-Analog Converters (DACs) and Analog-to-Digital Converters (ADCs) which contribute a large fraction of the area and power consumption of the array macro.

There are several emerging Non-Volatile Memory (NVM) devices that can be used to implement the memristive device, including Resistive Random Access Memory (ReRAM), Phase

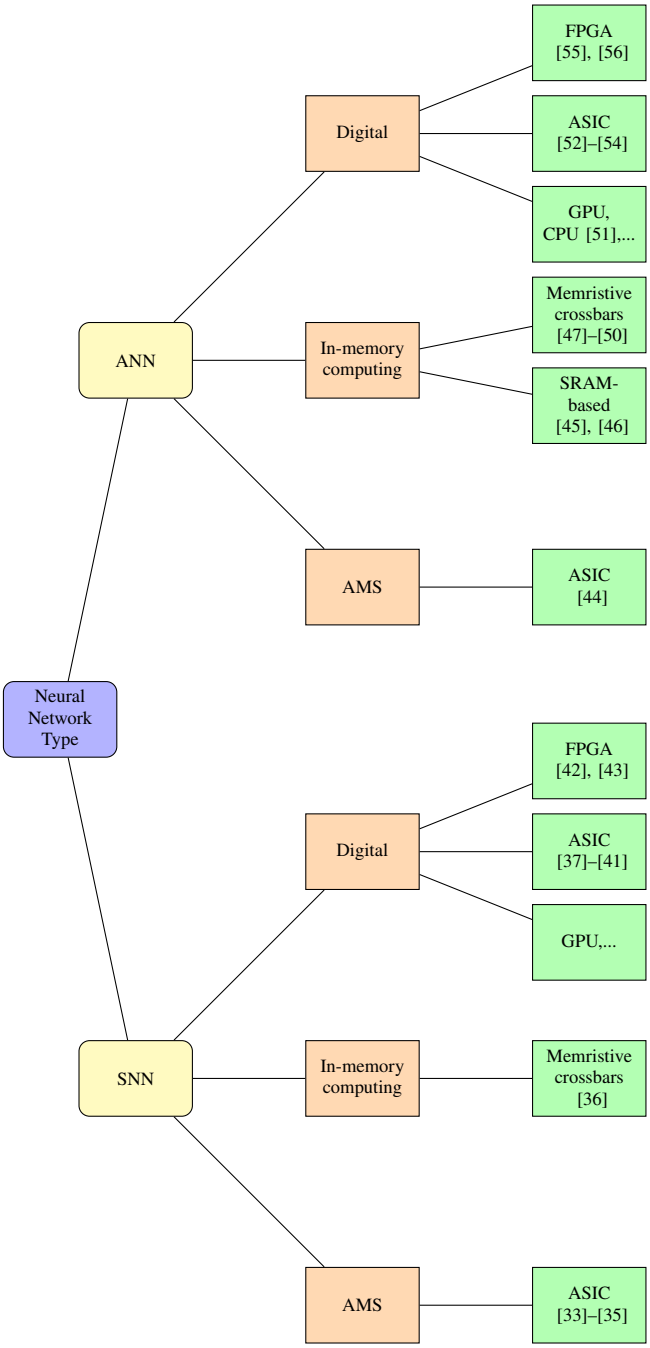


Fig. 6: Taxonomy of AI hardware accelerators.

Change Memory (PCM), and Spin Transfer Torque Magnetic Random Access Memory (STT-MRAM) [30]. These devices are compact and can perform read and write operations with low power. However, they suffer from several imperfections, such as conductance variations and drifts, which result in poor yield, stability, and endurance. Therefore, enhancing the reliability of crossbar-array computation is a subject of ongoing research.

Finally, 3D integration technologies could offer several advantages such as short interconnections, high parallelism, high bandwidth, and small form factors [31], [32].

A taxonomy of AI hardware accelerators is illustrated in Fig. 6. The first layer defines the type of neural network, i.e., ANN

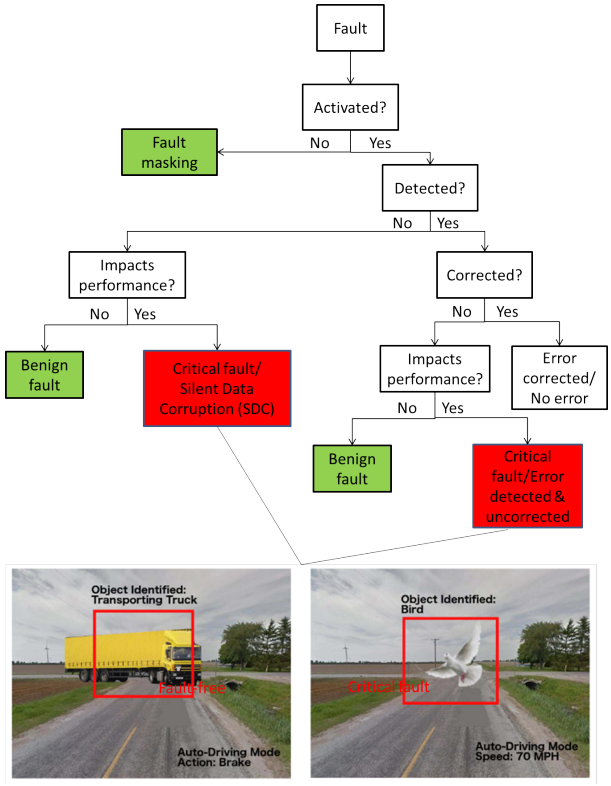


Fig. 7: Fault classification. Images are from [57].

or SNN. The second layer defines different design flavors, i.e., digital or AMS, with in-memory computing inserted as a separate category. The third layer defines the implementation, i.e., ASIC, FPGA, etc., while in-memory computing is further distinguished into digital processing-in-memory, i.e., SRAM-based in-memory computing, and memristive crossbar arrays. In Fig. 6, we provide references to representative designs with a focus on designs that have been demonstrated on silicon. Regarding memristive crossbar-array accelerators, most works present only simulation results up to now. This list of references is not meant to be complete. For recent and thorough surveys on accelerator design for ANNs and SNNs, the readers are referred to [3], [4], [6]–[9] and [5], respectively.

IV. Fault Criticality Assessment

A. Introduction

In the context of an AI hardware accelerator many faults turn out to be benign: they are masked before their effect reaches the output or produce an output change that is tolerable, i.e., it does not translate to performance loss. This is thanks to the network sparsity, the over-provisioning, the distributed computing, and the nature and sequence of mathematical computations. Some faults, however, will be critical and will affect the performance. A fault classification is shown in Fig. 7. The ability of quickly assessing the impact of faults on the AI hardware accelerator performance is very valuable for performing early reliability analysis and for guiding the development of efficient and cost-effective fault detection, fault tolerance, and fault repair schemes by placing the focus on targeting the critical faults only.

A network is viewed as a distributed system where neurons and synapses can fail independently [58]. Given the large number of synapses and neurons, the size of the fault space easily explodes, not to mention the rest of the hardware components. On the other hand, the fault impact is typically expressed in terms of accuracy drop on the testing set, which can contain several thousands of samples, while the time for a single inference can be very long. For these reasons, performing fault simulation at the hardware-level can be intractable, thus necessitating fault modeling approaches at a higher abstract level. As a matter of fact, performing exhaustive fault injection even on a higher abstract network representation may still not be feasible, thus necessitating fault sampling.

Another challenge in fault modeling is that the fault impact is determined by the interactions between the network model, the dataset, and the AI hardware accelerator. When analyzing the fault impact, the AI hardware accelerator architecture and the scheduling of network operations on its architectural components cannot be ignored [59].

Fig. 8 shows a fault injection experiment flow. Starting with a fault model, a fault list is created as a subset of the fault universe possibly using fault sampling. A single fault assumption or multiple fault scenario with user-specified fault rate can be considered in this step. Then fault injection is performed on the AI hardware accelerator which could be done at different insertion levels, i.e, in a software model, RTL-level, microarchitectural-level, gate-level, transistor-level, on an actual hardware prototype, or with radiation. For every fault scenario the fault impact is assessed and stored. After going trough the complete fault list, a report is produced, for example including the benign and critical faults, the critical fault locations, and the fault rate that can be tolerated.

Examples of fault criticality visualizations are shown in Fig. 9. In Fig. 9(a), the x-axis shows the different layers and for each layer there are two columns, each corresponding to a different fault type. A column is a colored bar possibly separated into chunks of different colors. Each chunk of the bar corresponds to a specific classification accuracy according to the color shading shown at the bottom of Fig. 9, and the projection on the y-axis shows the percentage of neurons for which the fault results in this classification accuracy. While Fig. 9(a) shows the cumulative neuron criticality across layers, Fig. 9(b) shows the per-neuron criticality as a heat map with the neuron number in the x-axis and the layer number in the y-axis. Each orthogonal corresponds to one specific neuron, and the color of each orthogonal corresponds to the classification accuracy in the presence of a fault in the neuron according to the color shading in the bottom of Fig. 9. For a given fault type, Fig. 9(c) displays the impact on classification accuracy of synapse faults in the synaptic matrix between two layers. Using such plots, one can label faults as critical or benign and identify critical fault locations across layers and within each layer.

As we will see next, most research works consider bit-flips in the memories and registers storing the network parameters, i.e., synapse weights and neuron activations. For this fault model, examples of reliability assessment are shown in Fig. 10. Bit-flips can be injected with some Bit Error Rate (BER)

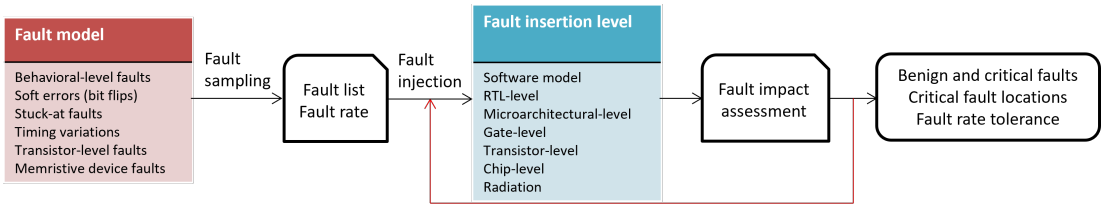
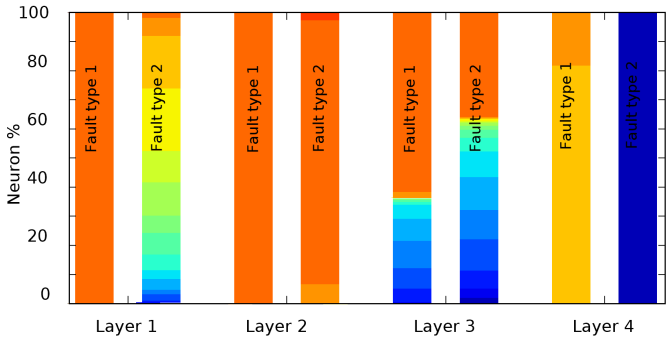
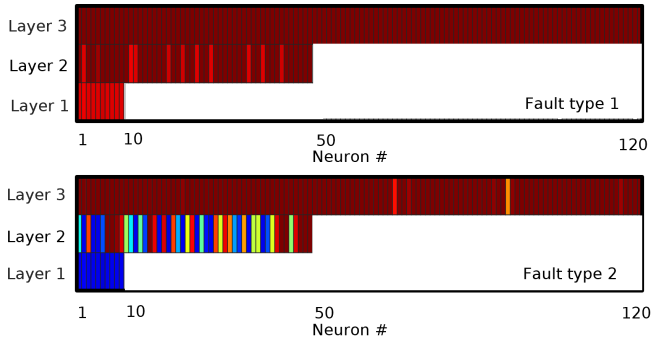


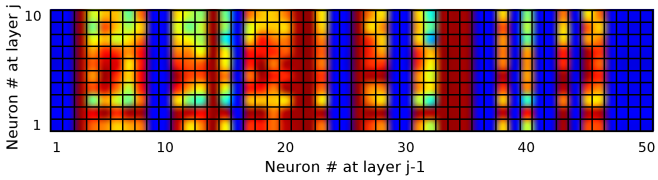
Fig. 8: Fault injection experiment flow.



(a) Cumulative neuron fault criticality across layers.



(b) Per-neuron fault criticality.



(c) Synapse fault criticality.

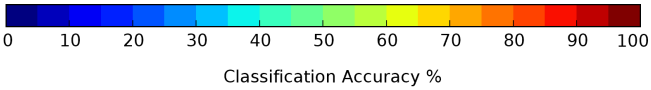
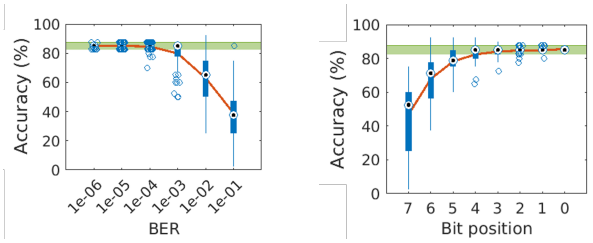


Fig. 9: Fault criticality visualization.

probability to assess the largest BER that can be tolerated, as shown in Fig. 10(a). The experiment is repeated several times and summary statistics are visualized in Fig. 10(a) using box plots. The bottom and top edges of the box indicate the 25th and 75th percentile, respectively. The whiskers extend to the most extreme data points without considering outliers, and the outliers are plotted individually using the 'o' symbol and are not always aligned vertically for illustration purpose. Fig. 10(a) also illustrates the baseline fault-free accuracy



(a) Accuracy drop for bit-flips with different BER levels.

(b) Accuracy drop for bit-flips at different positions of the word representing a network parameter.

Fig. 10: Reliability assessment using bit-flips as fault model.

shown with the green zone, the median shown with a dotted circle, and the average accuracy across repetitions of the same experiment shown with a red line. Bit-flips can also be injected at individual bit positions as shown in Fig. 10(b) where the network parameter has an 8-bit representation. For example, with the results in Fig. 10(b) we can identify those bits starting from the Least Significant Bit (LSB) that have no impact on the accuracy if they are flipped and can be left unprotected in a fault-tolerance strategy.

In Section IV-B, we survey several works demonstrating fault injection experiments and frameworks. Fig. 11 shows different fault types at different insertion levels. A taxonomy of works is provided in Table I based on the fault insertion level, while memristor crossbar-based architectures are treated as a special category. In Section IV-C we summarize general conclusions from these experiments.

B. *Fault injection experiments*

1) Software-level: The software and hardware implementation of a neural network matches closely in terms of component connectivity and data flow, thus allowing performing fault injection in software in a more time-efficient manner. This was noticed in early works [58], [61]–[63] where structural behavioral-level fault models were used in the main software operators that support the network computational task, i.e., neurons and synapses. Behavioral-level fault types included stuck-at nodes, missing or saturated neurons, errors in the summation or the evaluation of the neuron's nonlinear activation function, errors in synaptic multiplication, disabled or saturated weights, errors in learning rules, noisy inputs, etc. These behavioral-level faults can be mapped to physical fault models and root-causes in hardware, i.e., gate-level stuck-at faults and soft errors, for both digital and analog circuit implementations of neural networks [64], [65]. In [58], a the-

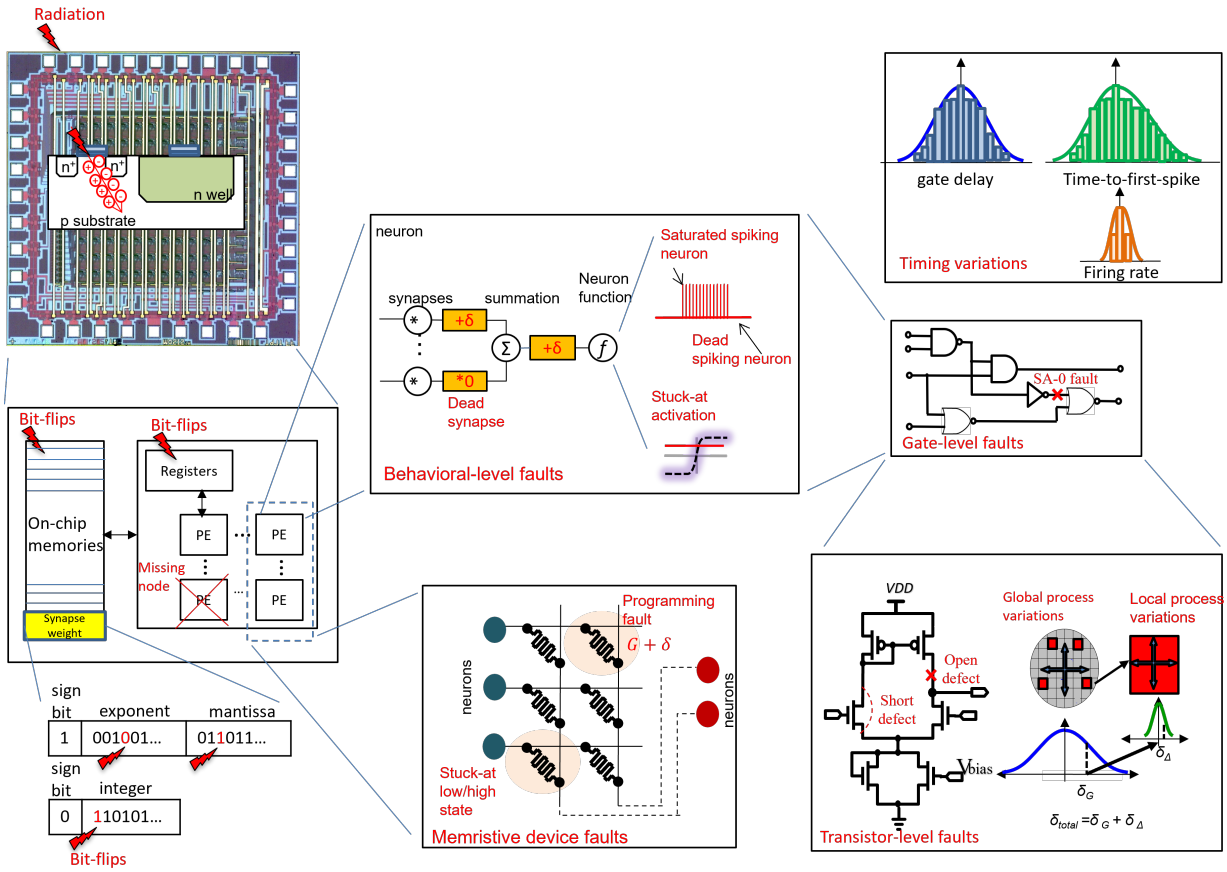


Fig. 11: Faults models at different insertion levels. The chip image corresponds to an AMS implementation of a FC network used as an on-chip classifier for Built-in Self-Test (BIST) purposes [60].

TABLE I: Taxonomy of fault injection experiments and frameworks.

Fault injection experiments and frameworks
Software-level [57], [58], [61]–[63], [66]–[91]
RTL-level [92], [93]
Microarchitectural-level [94]
Gate-level [95]–[98]
Transistor-level [99], [100]
Chip-level [101]–[110]
Radiation experiments [81]–[85], [111]–[114]
Memristor crossbar-based architectures [115]–[118]

oretical study is presented for Feed-Forward Neural Networks (FFNNs) deducing the number of failing neurons and synapses an FFNN can tolerate.

As there is a large body of works in this direction for modern AI hardware accelerators, we categorize them according to the two neural network types, namely ANNs and SNNs.

a) ANNs: In [57], the fault model used is bit-flips in datapaths and buffers. A wide range of data types are considered and bit flips are injected in different bit positions. Fault injection is carried out in the open-source DNN simulator framework Tiny-CNN written in C++, where each line of code is mapped to the corresponding hardware component so as to pinpoint the impact of the fault injection location in terms of the underlying micro-architectural components. The focus is on CNNs considering different image classification tasks. Four types of Silent Data Corruption (SDC), defined as a mismatch between the output of a faulty and the fault-free inference execution, or “fault ratings" are proposed taking into consideration that networks may rank predictions based on a confidence score. Some conclusions of this large-scale fault injection study are: (a) different DNNs have different sensitivities to SDCs depending on the topology, the types of layers, the data type used, and the position of the bit flip; (b) Failure in-Time (FIT) rates can exceed the safety standards, e.g. ISO 26262 for automotive, by orders of magnitude; (c) data types that provide more dynamic value range are more vulnerable to SDCs since there are likely to be redundant value ranges that lead to larger-value deviation under faults. This implies that just-enough numeric value-range and precision is advantageous from a reliability point of view; (d) normalization layers reduce the impact of faults by averaging fault values with adjacent correct values.

In [66], the *Ares* framework is proposed that simulates static bit-flips in the memory of the DNN accelerator. *Ares* is built on top of Keras [119], which takes high-level DNN descriptions specified in Python and executes them using either Theano [120] or TensorFlow [121] back ends. Fault injection experiments are performed for several DNN models and datasets to study the classification rate as a function of BER. Fault injection is performed across the whole network, per-layer, and across network components, i.e., weights and activation functions. Main conclusions of this study are: (a) a thresholded behavior is observed where for small BERs the classification error is zero, but there is a BER threshold beyond

which the classification error rises exponentially from zero; (b) there is a largely spread fault sensitivity or resilience across the DNN models, e.g., the threshold varies by two orders of magnitude; (c) the weight quantization impacts resilience, i.e., the larger the range of the possible weight values is the lower the threshold is; (d) fault sensitivity across network layers and components can vary by several orders of magnitude.

In [67], the *FIdelity* DNN resilience analysis framework is proposed where hardware faults are modeled in software, i.e., TensorFlow [121], thereafter high-speed software fault injection is performed. In this way an analysis speedup is achieved while maintaining the level of accuracy of RTL or mixed-mode fault injection techniques. To map hardware faults in software the key insight is that hardware and software operations closely match, and all operations affected by a fault can be systematically derived thanks to well-defined data-flow and scheduling algorithms. Given high-level architecture/hardware information and Flip-Flop (FF) FIT rate, the framework captures the effect of hardware faults to set a faulty output neuron using a reuse factor analysis for FFs. Faulty output neuron values are derived considering that each FF value already corresponds to a software-variable state. A key aspect of the framework is that it can treat logic transient errors in data-path and control FFs and not only memory errors.

In [68], a methodology is proposed to reduce the fault injection space and, thereby, the overhead of an exhaustive fault injection. The underlying observation is that most ML functions in a DNN model, i.e., convolution, ReLu, pooling, normalization, etc., are monotonic. This means that in a word representing a model parameter, there exists a SDC-boundary bit such that bit-flips at higher-order bits would lead to SDCs and bit-flips at lower-order bits would be masked. Based on this observation, the Binary Fault Injection (*BinFI*) fault simulator is proposed that bisects the fault injection space and finds the SDC-boundary bit with a binary-search like algorithm. *BinFI* is built on top of the TensorFlow framework [121] duplicating the graph with customized operators.

In [69], a fault injection framework is proposed that reproduces fault models and event rates extracted from radiation tests. The ultimate goal is to have the flexibility of a software-based fault injector with a reliability assessment precision close to this of an accelerated neutron beam radiation-based fault injection experiment in a realistic harsh environment.

The interested reader is referred to [70]–[85] for more software-based fault injection experiments studying the fault impact for different neural network models, data type representations, layer types, network sizes, pruned networks, compressed versus uncompressed networks, etc. Such experiments are also part of several other works that will be discussed in Sections V and VI and they are used for motivation or for guiding test and dependability solutions. Many of these works develop at the same time an in-house automated fault injection framework. The development of automated, fast, flexible, and accurate fault injection frameworks is an active area of research. Examples include PyTorchFI [86] and TensorFI [87], which are open-source and publicly available, and CLASSES [88]. An interesting research line is the development of ML-based frameworks that based on a small number of fault injections they estimate the vulnerability for all parameters in the DNN in a short time [89]. Such approaches will be described in more detail in Section IV-B4 for systolic-array DNN architectures at gate-level and in Section IV-B8 for memristor crossbar-array architectures. Another possibility is to use generic fault injection tools, such as SASSIFI [122], NVBitFI [123], and CAROL-FI [124], to emulate fault effects in the hardware platform, i.e., GPU, running the application.

b) SNNs: The fault tolerance characteristics of SNNs trained with different algorithms is studied in [90]. The fault model is synapse fault where a faulty synapse is zeroed out or equivalently removed. Synapses are selected to be faulty at random with different failure rates. Results show that these different algorithms have different resilience characteristics. Resilience can greatly depend on the training algorithm and dataset, and it can also show large variances according to the synapses that are selected to fail. Common conclusion is that for all considered networks resilience drops rapidly as fault rates increase, and SNNs are not inherently resilient as it is frequently cited.

In [91], the behavioral-level fault model proposed in [100] (see Section IV-B5) is used to perform accelerated fault injection in deep SNNs. The fault injection framework is built on top of the SLAYER [125] and PyTorch [126] frameworks by customizing the flow of computations and the faulty SNN is mapped onto a GPU. The general conclusion of this experiment is that saturation neuron faults are the most lethal and can severely affect inference regardless the location of the neuron in the network, and that the impact of all other fault types, i.e., dead neuron faults and timing variations, may be severe only for neurons in the last hidden and output layer. At the extreme, timing variations could result in a dead or saturated neuron. In other words, a neuron that becomes permanently active has a greater effect on inference compared to a neuron that is permanently silenced or presents timing variations in its output spike train.

2) RTL-level: In [92], fault characterization is performed on a RTL design of a typical accelerator. Fault injection is performed into the different registers that latch data during the inference, i.e., input, weights, and intermediate layer computations. The fault model includes permanent stuck-at faults and transient faults occurring in a single random cycle. In each fault injection experiment, a different fault is randomly generated and injected by selecting a random register and set of bits and a random cycle in the case of transient faults. Fault characterization is performed across the different register types, layers, components of fixed-point data representation, i.e., sign, digit, and fraction, number of PEs, and network models. General conclusions are: (a) permanent faults are more critical than transient faults; (b) stuck-at-1 faults are more critical than stuck-at-0 faults due to the sparsity of zeros; (c) registers storing intermediate data are the most vulnerable, whereas input registers are the least vulnerable; (d) permanent faults are more critical in inner layers, i.e., closer to the output, while the opposite is observed for transient faults; (e) sign, digit, and fraction are in this order more vulnerable; (f) for permanent faults the error decreases with the number of PEs, while there is no correlation in the case of transient faults.

In [93], an RTL-level fault injection framework is proposed that drastically reduces the fault simulation time. It makes use of a multi-level structure where on the lower level the inference is split in several blocks corresponding to the neural network layers that run as stand-alone application processes and on the upper level these processes are synchronized.

3) Microarchitectural-level: In software fault injection the fault model risks to be unrealistic and faults can be mapped to only a subset of hardware resources. In [94], the concept of a two-level fault injection is adopted to evaluate the effects on CNN execution of faults in the GPU's scheduler and pipeline registers, two microarchitectural components that otherwise would be hidden in an abstract high-level CNN model. Microarchitectural simulation requires a prohibitively high time. To improve efficiency, the two-level fault injection idea is composed of the following steps: (a) perform microarchitectural fault injection, i.e. transient bit-flips; (b) observe the effect on selected CNN tiles (i.e., matrix portions); (c) merge the corrupted tiles' output with the other tiles in the convolution considering their fault-free output to compose the layer's output; and (d) continue the execution of CNN at the software-level to check if the fault is eventually masked or it propagates at the output creating an error. Finally, a feedback analysis can determine the microarchitectural locations causing the observed critical errors that should be targeted for hardening.

4) Gate-level: In [95], fault injection experiments are performed on a systolic array-based DNN accelerator. The core of the systolic array is composed of a 256×256 grid of MAC units. Each weight maps to exactly one MAC unit, thus a faulty MAC unit can result in multiple faulty weights. The systolic array is developed in Verilog and synthesized at gate-level. The fault model includes stuck-at faults at gate-level and timing faults created by under-scaling the power supply which essentially emulates process variations. It is demonstrated that training on a faulty systolic array can result in significant classification drop when as few as four MAC units are faulty.

In [96], the impact of timing variations on hardware implementation of ANNs is studied. Timing variations could result from delay defects, process variations, power supply noise, crosstalk, aging, voltage over-scaling or frequency overclocking. Timing variations are modeled at gate-level by introducing an extra delay variation in the range of 10-40% into each gate relative to the nominal gate delay. Results show that ANNs are sensitive to timing variations with the error growing larger as the timing variations worsen. Accuracy loss can be alleviated to a large degree but not fully recovered if the ANN is retrained under timing errors.

In [97], [98], ML-based frameworks are proposed for analyzing the functional criticality of gate-level stuck-at faults in systolic array based AI accelerators. Fault injection targets not only the interface/boundary level of a PE, but also all internal nodes of a PE. The main challenge of such a task is on computation/simulation overhead introduced by a significantly large number of potential fault injection points, e.g., there will be tens of thousands of stuck-at faults for a single 32-bit PE, and billions for a 256x256 PE array. To this end, computationally efficient ML-based methods are proposed to speed up the analysis. The basic idea is based on the use of deep learning to predict fault criticality by utilizing the structural and data flow features. For example, in [98], a two-tier DNN-based model is presented, as illustrated in Fig. 12. The first tier DNN is trained on a data set obtained from ground-truth collection. The second tier DNN is trained on a smaller and targeted data set containing the critical faults mis-predicted as benign by the first tier DNN. A Generative Adversarial Network (GAN)-based method is further used to augment the data for the second tier DNN, in order to minimize misclassification (i.e., misclassify critical faults as benign). The transferability of the proposed method is also investigated (i.e., if a fault criticality model trained on a PE can be transferred to evaluate a different PE). The results show that there exists some inherent transferability across PEs in the same array, mainly due to their identical topologies. On the other hand, more model re-training will be needed if transferability is not met.

5) Transistor-level: Transistor-level fault simulations can be performed only at neuron-level or for small-size networks.

In [99], transistor-level short- and open-circuit defects are injected into the fundamental logic operators of a perceptron, i.e., adders and multipliers. Fault injection experiments in a shallow 2-layer classical fully-digital spatial expansion ANN architecture have demonstrated that hidden layers can tolerate defects even for high defect rates. However, depending on the affected bit or neuron, there may be single defects that can influence the inference accuracy. Defects in hidden layers can be silenced out by a re-training operation with the hardware in-the-loop even for high defect rates. In contrast, the output layer is a defect-sensitive layer and defects in this layer cannot be masked by re-training necessitating a dedicated fault tolerance scheme.

In [100], defect simulations and Monte Carlo analysis taking into consideration the technology Process Design Kit (PDK) are performed for a spiking neuron. The different faulty behaviors are collected and grouped so as to generate an abstract behavioral-level fault model for spiking neurons that capture the effects of low-level faults, i.e., transistor-level defects and process variations. Faulty behaviors turn out to be either catastrophic (i.e., dead neurons that are silenced even in the presence of input activity, saturated neurons that fire non-stop even in the absence of input activity, neurons with a stuck output, etc.) or parametric (i.e., timing variations of the output spike train such as variations in the time-to-first-spike and firing rate). This bottom-up behavioral-level fault modeling approach starting from transistor-level simulations can help generating fault models at a higher abstraction level to be used for software fault injection, while still capturing the effect of underlying root-cause transistor-level faults on the neuron's output, independent of its actual hardware implementation. For example, dead and saturated neuron behavior can be modelled in the output spike train, while timing variations can be modelled by varying various neuron parameters, i.e., the neuron's membrane potential threshold.

6) Chip-level: Software-based fault injection is fast and flexible but it ignores the behavior of the AI hardware accelerator. RTL-level, microarchitectural-level, gate-level, and transistor-level fault injection takes into consideration the

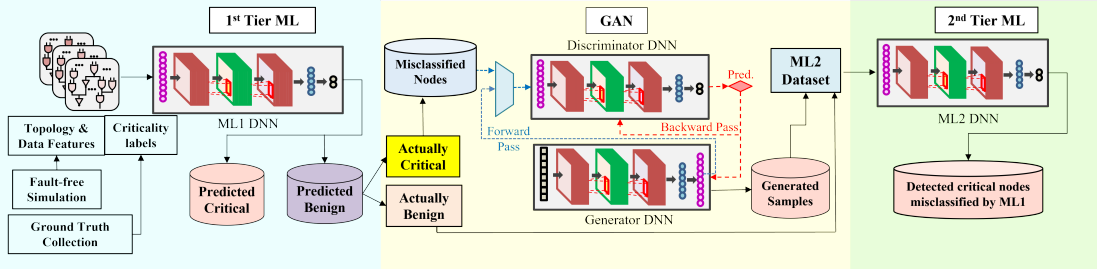


Fig. 12: ML-based method for criticality assessment [98].

hardware, but it is slow and inevitably limited on specific hardware blocks.

FPGA-based hardware accelerators offer the possibility to perform realistic fault injection, including faults that would be difficult to model with software simulation, i.e., faults affecting the configuration memory or controlling modules. They offer also the possibility to evaluate both accuracy degradation and system exceptions, such as system stall and running overtime. Fault injection experiments on accelerators implemented on FPGAs are presented in [101]–[108]. In particular, fault injection experiments are performed on the FPGA-based FINN Quantized Neural Network (QNN) accelerator [56] in [101], [103]–[105], for the tinyTPU implemented on an FPGA in [107], for FPGA implementations of custom ANN accelerators in [102], [106], and for neuromorphic FPGA-based hardware supporting SNNs in [108].

In [109], [110], the soft error reliability of CNN models running on microprocessors is investigated, analyzing the results for different components of the microprocessor and precision bitwidth configurations.

7) Radiation experiments: Experiences from radiation experiments on different GPUs running different DNN models are described in [81]–[84]. In [81], FIT rates are scaled to natural terrestrial environment. Main observations are as follows: (a) crashes are more frequent than SDCs but are less critical as they can at least be detected; (b) all reported SDCs rates are higher than the 10 FIT limit imposed by the ISO 26262 safety standard for automotive, thereby reliability of GPU-based AI accelerators is paramount; (c) FIT is dependent on the technology, i.e., for FinFET it is an order of magnitude lower than that of standard CMOS; (d) Error Correction Code (ECC) protection alone is insufficient to ensure high reliability. In [82], the FIT rate is evaluated for different data precisions, showing that it increases with precision since it depends not only on the fault propagation probability but also on the probability of the fault occurrence. In [83], the run-time of the inference is tuned based on the beam flux such that the chip experiences no more than a single bit flip event during each application run. The study shows that with the ECC/parity checking enabled, single bit errors are corrected, no SDCs is observed, and the most stringent ASIL D requirement imposed by ISO 26262 is met. However, vulnerability to permanent faults is observed, which shows that ECC/parity checking must be complemented with periodic structural tests.

Accelerated radiation testing results for DNNs running on FPGAs are reported in [85], [111]–[113]. In [85], it is shown that applying selective Triple Modular Redundancy (TMR) to only the most vulnerable layers can mask a high percentage of faults. In [111], [112] it is evaluated how reducing the bit-width used for data representation impacts the radiation sensitivity and failure rate. In [113], it is shown that QNNs trained with fault-aware training are more resilient to soft errors.

Finally, results on the reliability to neutrons of Google Coral TPU are reported in [114], considering elementary operations and several CNN models. It turns out that, despite the high error rate, most neutron induced errors only slightly modify the convolution output and do not change the detection or classification of CNNs.

8) Memristor crossbar-based architectures: The work in [115] studies the fault injection effect in memristor crossbars. The fault model includes stuck-at faults in the conductance of memristors after programming. The conductance error is defined as the difference between the final programmed value and the target value. A device with a conductance error higher than a positive threshold is considered to have a stuck-on fault, i.e., it freezes at a high conductance state. Whereas a conductance error below a negative threshold is considered to have a stuck-off fault, i.e., it freezes in a low conductance state. Fault injection experiments show that inference accuracy drops by more than 50% for a stuck-at memristor rate of 20%.

The work in [116] proposes a fault model for SNNs using memristor crossbars for the connection of the layers. Spike Timing Dependent Plasticity (STDP) is used for learning. Some specificities of the SNN design are that the output neurons are implemented with lateral inhibition and synapses are off when there is no activity on their connected neurons. The fault taxonomy is divided into different synapse faults and neuron faults. Synapse faults include dead synapse, degraded plasticity, and synapse stuck-at faults. Neuron faults include dead neurons and delayed spiking, as well as faults specific to this SNN design, i.e., stuck-at or delayed lateral inhibition and delayed synapse activation fault. Fault injection experiments considered only the worst case faults, i.e., dead neuron and dead synapse faults. Results show that a high fault density is required for noticeable decrease in recognition rate. Moreover, for dead neuron faults, learning on a faulty network is more critical than a fault occurring in a fault-free trained network.

In [117], the susceptibility of ReRAM-based crossbar arrays to single event and cumulative radiation damage is investigated. Simulations are performed using an experimentally derived memristor SPICE model. Results for an ANN trained with the MNIST dataset indicate that the system is highly resistant to transient Single Event Effects (SEEs) thanks to

the low cross section of the memristive device. Moreover, the cumulative ionizing dose level corresponding to the inference failure point is very large, thus it is concluded that ReRAM-based accelerators have high radiation tolerance in normal environments.

In [118], it is proposed to train a ML classifier to predict fault criticality in a DNN mapped to memristor crossbars. The considered fault types are stuck-on and stuck-off conductance in the memristor cell. The training set is generated by: (a) random fault injection for which the overwhelming majority of analyzed faults will be benign; and (b) a Misclassification-Driven Training (MDT) algorithm to quickly identify critical faults so as to have a balanced training set. The MDT algorithm runs an optimization where the DNN parameters, i.e., weights, are perturbed towards maximizing the prediction error. In each iteration, the most significant parameter based on gradient value is chosen. A fault is injected in this parameter and is identified as critical fault if all samples in a batch of the dataset are mispredicted. The features from the benign and critical faults used to train the ML classifier are: (a) fault location; (b) fault type; (c) parameter significance; and (d) parameter deviation amount. The fault criticality analysis can be used to develop a fault tolerance solution that targets only critical faults, thus leading to a significant reduction in the redundancy needed for fault tolerance. The proposed criticality-aware fault tolerance scheme used in this work is to introduce spare columns for remapping only columns in the memristor crossbar that include cells with critical faults.

C. General observations from fault injection experiments

Some common conclusions in the above fault injection experiments are as follows:

1) The fault impact depends on the DNN topology, type of layer, and type of activation function used. Moreover, fault sensitivity across layers and across neurons within a layer can vary by several orders of magnitude. Typically, the output layer is a highly-sensitive layer necessitating a dedicated fault tolerance scheme. Convolution and fully-connected layers tend to spread the SDCs, while sub-sampling layers tend to mask a significant portion of SDCs. Moreover, very frequently a bimodal behavior is encountered: either the accuracy is negligibly impacted by the fault, or the accuracy drops rapidly even approaching random guessing.
2) FIT rates of AI hardware accelerators can exceed safety standards, which shows that reliability and error recovery is of paramount concern.
3) The accuracy drop is contingent on the dataset, i.e., the application. The same fault can be benign for one dataset but can be critical for another.
4) Fault susceptibility depends on the data type used. DNNs using data types of higher dynamic range are more vulnerable. Still, even QNNs with 2-bit precision are shown to be vulnerable. Susceptibility also depends on the affected bit position, with the MSBs being the most critical.
5) Stuck-at-1 faults furnish the largest accuracy drop because typically over 99% of model parameters have zeros in their MSBs.
6) For systolic array-based accelerators, by increasing the number of layers or the number of neurons per layer the accuracy drop escalates [127]. This is due to the reuse of the systolic array across multiple layers.
7) For memristor crossbar-based architectures, single memristor yield and endurance is very low, necessitating yield rescuing methods.
8) For SNNs, saturation neuron faults seem to be the most lethal, although dead neuron faults can also cause significant accuracy drop.

V. Testability

A. Introduction

The goal of testability in AI hardware accelerators is no different from traditional hardware: achieving acceptable test quality under manageable cost. It is confronted by the same problems as in traditional test but with new challenges. While some challenges can be handled by existing tools and solutions, many still remain as major problems in today's Design-For-Testability (DFT) applications. Some typical issues seen in industry are discussed below.

Being domain-specific, AI hardware accelerators usually have some unique features that may not be test friendly. The most prominent one is the sea-of-core design, e.g., 1472 cores in Graphcore GC200 [128], 128x128 systolic array in Google's TPU [129], or even more such as 850K cores in CS-2 [130]. While the notion of "core" in different accelerators may be very different in size, cores in one design are usually identical or very similar. From DFT and physical design perspective, these cores may be too small to implement DFT on a per-core basis with a reasonable overhead. On the other hand, incorporating many cores in a physical partition could lead to prohibitive cost for DFT or physical implementation and verification, while not taking the advantage of the similarities among cores. Attempting to achieve best test quality with reasonable overhead, there has been plenty of research on low-cost testing of systems with identical cores [131]–[133]. However for today's AI architectures, an optimal solution might be further explored from other angles, i.e., the function structure may help increase DFT test coverage [134], and function patterns may become part of test patterns [135].

AI applications are memory intensive, hence many AI hardware accelerators require embedded memories with much larger sizes than in traditional ASIC designs. Several MBytes are common practices, i.e., 900MB in Graphcore GC200 [128]. While these memories can be extensively tested and repaired using today's Built-In Self-Test (MBIST) tools, they can present major penalties to Power, Performance and Area (PPA) [136]. Recently, many AI hardware research topics have proposed to bring the computation near to the memory or into the memory, or using large external memory such as High Bandwidth Memory (HBM) or wafer-bonding. These solutions bring in new challenges for testing. For instance, in-memory solutions may require understanding and creation

TABLE II: New testability challenges for AI hardware accelerators and possible solutions.

Challenges	Solutions
DFT efficiency for designs with large number of cores	Test architectures
Physical design issues for large die size (i.e., routing and timing)	Physical-aware DFT [136]
DFT overhead	Function-aware DFT [134], [135], [145]
Complete DFT solutions for large heterogeneous systems	Functional test generation [127], [146]–[156]
Memory-hungry designs	On-line test [157]–[165]
New market demands (i.e., 0 DPPM for automotive, on-line test)	Yield improvement [115], [166]–[169]
New architectures and faults (i.e., in-memory computing)	Fault modeling and testing of memristor-based memory technology [170]–[177]
New design paradigms (i.e., 2.5D/3D ICs)	2.5D/3D IC DFT
	Automotive-grade DFT

of new logic and physical fault models [137], while wafer-bonding necessitates better solutions for test access, test power control, and yield improvement [138].

Besides the issues mentioned above, existing test challenges for traditional ASIC design may also become increasingly intense in AI applications. For example, current large AI hardware accelerators require hierarchical DFT solutions that are scalable with design size. However, since an AI hardware accelerator is often an heterogeneous system, hierarchical DFT needs a comprehensive solution for automatic DFT insertion, verification, debugging and silicon bring up. Another popular difficulty stems from physical design, i.e., large accelerators often use tile-based design where no dedicated routing channels are reserved for global routing. However, complex DFT designs can create hundreds of global signals for scan, MBIST, debug, etc. This presents a huge overhead for top level implementation and verification. New solutions are necessary for both efficient DFT and easy physical implementations [136] [139].

AI hardware accelerators are also facing pressure from new marketing and technical trends. For example, as one of today's popular applications, automotive grade AI hardware accelerators require more stringent screening than before to ensure zero Defective Parts Per Million (DPPM), which has to be reflected in the architectural level of DFT design. Test data analysis and diagnosis are also critical for yield and reliability learning [140]. Another typical new paradigm is 3D IC design, since large AI hardware accelerators are often limited by physical geometry and cost. Chiplet designs based on 2.5D or 3D methodologies can mitigate the challenges from die size, process, cost, etc., but necessitate a complete set of test solutions from die-level, stack-level, to package-level, which is being addressed in the development of new tools and test flows [141]–[144].

Table II categorizes some of these challenges or issues (left) and possible solutions (right), which are discussed in this paper. This taxonomy is not meant to be comprehensive, but can be representative for many testability activities in the AI hardware designs. Table II cites only works specific to AI hardware, but more generic solutions will be discussed, especially on the test architecture side, that are applicable to AI hardware too.

B. DFT and ATPG

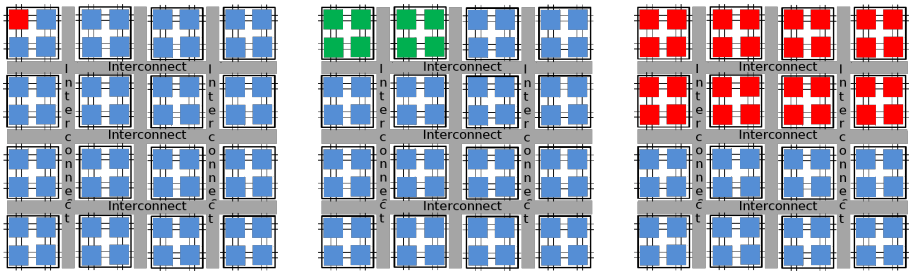
1) Test architecture: Plenty of research has been conducted on DFT solutions for identical cores to minimize test overhead and maintain test quality. The assumptions of these techniques may still be valid for AI hardware accelerators, but new solutions may be needed to handle large AI designs with limited cost.

A straightforward idea is to broadcast the test stimuli to identical cores while compare their test responses for pass/fail. In [131], a Test Access Mechanism (TAM) is designed to implement on-chip comparison of multiple identical cores. It contains multiple stages of pipelines and several configurations so that each core's test can be implemented in different modes. As such, it not only supports comparing test responses from cores for manufacturing test, but also provides diagnosability of a core during silicon bring up and yield ramp up. With on-chip comparison, test data volume can be significantly reduced, and so is test time. However, for large AI designs with many cores, such broadcasting-style solution may encounter increasing difficulty from routing and top integration. It also needs some manipulations of test patterns from standard Automatic Test Pattern Generation (ATPG) tools.

In [132], another form of TAM is proposed supporting similar features. It is a generalized time-multiplexed TAM, where the compressed test and control data streams are serialized before going into the decompressor. At core level, designers can still utilize the regular decompressor/compactor scan architecture. This simplifies the core level scan channel configuration and decouples it from top-level scan pin assignment. The designers can be flexible in architecting a core-level scan scheme without worrying too much about the top-level. This can be a major benefit for large AI hardware accelerator designs with many cores but very limited scan pin resources. Pattern re-targeting, verification and diagnosis flow are also supported, which are also critical for today's large AI hardware designs.

It can be seen that in order to take the advantage of sea-of-core design style in AI hardware, on-chip processing of test data may be preferred when implementation is feasible. There are several varieties in this domain. In [133], a TAM design is presented for chips with multiple isolated identical cores. The proposed pipelined architecture relies on forming nonlinear equations on a very limited number of output pins that compress the outputs from the identical cores and solve them off-chip to reproduce the failure information of each core. It uses test resources similar to testing a single core and also supports accurate failure diagnosis. In [178], a TAM based on majority comparison is presented. It also utilizes an on-chip comparator, yet not to compare with an expected value but with other core's test data to determine a majority value. For example, if more than half of cores in the comparison present the same value, this value is a majority value. This value is then compared with Automatic Test Equipment (ATE) data. The test cost is close to that of a single core.

In [179], yield improvement is considered when a multi-core system contains spare cores. A comparison-based TAM that is capable of handling multiple spare cores is proposed.



(a) Small core level. (b) Core group level. (c) Big partition level.

Fig. 13: DFT solutions at different levels for many core AI hardware accelerator designs.

All faulty cores can be identified via low-cost comparison, and if the spare cores are more than the faulty cores, the chip is still usable. Using spare cores is also common practice in accelerators with many cores. Such designs usually provide a configuration with all good cores and several "partial-good" configurations with different numbers or locations of good cores for yield improvement.

It can be seen that for such DFT solutions to be acceptable, several issues have to be resolved. First, we should be able to manipulate the patterns according to the scan architecture, e.g., re-targeting the core-level patterns to top-level without regenerating the pattern. Second, test quality such as coverage should not be compromised and overhead should be minimized. And finally, from engineering perspective, some critical metrics such as single core diagnosability, verification effort, and routing complexity should also be considered. Fortunately, some of these requirements are already supported well by current mainstream DFT tools.

2) Physical-aware DFT: The above works are mostly developed for traditional multi-core designs. As discussed earlier, many AI hardware accelerator designs share certain features that may render these traditional solutions either impractical or not as efficient, especially on physical design. To address these AI specific problems, some new industrial efforts are reported to make DFT solutions more physical-friendly and hence more practical.

In [136], a comprehensive set of DFT solutions targeting AI hardware accelerators are proposed. In scan test, this work identifies that although accelerators may contain many identical cores, these cores are not as big or as complex as cores in a traditional multi-core system such as a CPU. A typical accelerator for data center applications may contain thousands of "small" cores, as shown in Fig. 13(a). At this core level, any DFT insertion may incur huge PPA penalty, i.e., compression logic, wrapper logic, control logic and routing for DFT signals. It is too small for the DFT overhead to be economical. On the other hand, if we group many small cores together to create a big partition and apply DFT insertion at this level, as seen in Fig. 13(c), the run time, memory requirement, power consumption, pattern count, verification efforts, and other concerns may prevail and render it infeasible, e.g., ATPG or simulation cannot finish in a limited time. Meanwhile, the similarities among cores cannot be effectively exploited.

As a result, in [136] it is proposed to find a "sweet spot" where a suitable number of small cores are viewed as a "core group" where DFT insertion, verification, pattern generation and other activities are done at this level, as shown in Fig. 13(b). Note that in practice, this usually aligns with physical design requirements, which is probably the most straightforward solution. However, if physical partition is too big or too small, DFT can still make architectural changes to adapt to a suitable size of core group. After this core group is determined, existing technologies such as test data broadcasting, test response on-chip comparison, pattern retargeting, scan channel pin-muxing can be effectively applied.

Note that in practice the logical identical cores may not be physically identical. Synthesis and physical implementations may create various physical instances from the same logic module, converting a homogeneous system to a heterogeneous system from a physical perspective. A feasible DFT solution has to take this into account.

Streaming Scan Network (SSN) [139] is a recently introduced tool that can target these physical challenges. SSN is a bus-based scan data distribution architecture. It contains a scan data bus that travels through all cores in the design, a per-core controller (host) with IJTAG support, and regular scan compression logic. The bus is connected to chip-level scan pins and scan data for any core in the system are streamed in through the bus in the form of packets. The concept of packets is different from that in network switching, since SSN packet is a fixed-format data segment that only contains scan data, no address or opcode. The local host in each core is pre-configured through IJTAG to learn how to offload scan data from packets. The expected value can also be streamed in for on-chip comparison. The routing and heterogeneous problems with tile-based designs are also mitigated, since only a single test bus is routed through the entire chip. There is no need to pin-mux the scan channels from various cores to top-level scan pins, and the test bus interface is identical for all cores. Another benefit is that due to the flexibility of packeted test data, any cores can be tested at any time. This can help effectively control test power and improve test channel throughput. With comprehensive considerations of DFT and physical-design requirements, this solution is especially suitable for AI hardware containing many identical cores.

3) Function-aware DFT: Most of the aforementioned technologies are common DFT solutions without in-depth analysis of the function mode of hardware. Many AI hardware accelerator architectures are domain-specific or even application-specific, hence a customized DFT solution designed for a

specific AI architecture is intuitively best for PPA results. To serve this purpose, a DFT architect needs to understand how AI hardware works in function mode such that the DFT design can be optimized accordingly.

In [135], the authors realize that due to the unique architecture of AI hardware, traditional stuck-at and delay tests may not be sufficient. They study test methodologies and DFT requirements specifically for supervised ML systems. Hardware architecture of FIFO-based and scratchpad-based accelerators are analyzed. Test strategies for specific hardware components such as MAC, global buffer, activation functions, etc., are developed. These solutions are more function-like and can help bridge the gap between traditional test patterns and specific AI hardware test requirements. They are also easy for on-line test to ensure product quality.

In [145], post-manufacturing testing of DNN accelerators is discussed. It is argued that the inherent error-tolerance can be leveraged to reduce the fault model size and, thereby, the test time and cost. The idea is that if a fault does not lead to inference accuracy degradation for a given accuracy tolerance margin, then it is non-critical and can be dropped. Only critical faults will be targeted during test application. Two approaches are shown considering a gate-level implementation. The first approach is Boolean Satisfiability (SAT)-based structural testing where a SAT solver exhaustively checks all input combinations to determine the fault criticality and generates a test pattern able to detect it. The second approach is classical functional testing where the actual workload, e.g., images, are used as test inputs.

Function mode operation is also studied in [134] to improve test quality in accelerators with very large number of small cores. As suggested in [136], this scenario can be handled by grouping small cores into a core with a size suitable for both DFT and physical implementations. However it does not exploit the similarity among small cores and test quality may still be impacted if there are interactions between small cores. In typical AI hardware accelerators, there are heavy data traffic between adjacent cores, hence the test coverage on core boundaries is essential. The work in [134] studies inter-core connectivity, function data flow, and design homogeneity to derive a C-testable method that can run ATPG for only a single core to reduce test cost and maintain the coverage and diagnosability. By exploring design space, it also presents a hierarchical compaction scheme for on-chip response compaction under reasonable design constraints.

Fig. 14 shows a systolic array example consisting of 16 small cores (or PEs). Dataflow is only from left to right and from top to bottom. Each PE has registers on the input sides but not on the output side. If ATPG is performed at each PE level, coverage will be unacceptable since PE itself is not well wrapped by registers. However, if ATPG run consists of the five adjacent PEs inside the red line, faults in the green PE in the middle will be fully covered. Note that this ATPG pattern can be used to detect faults in all PEs in the same scenario. Since an architecture with small PEs usually contains a large number of them, the overhead of such a scheme is low.

Fig. 15 illustrates the sequence of testing the whole systolic array. ATPG consisting of 5 neighbouring PEs is repeatedly

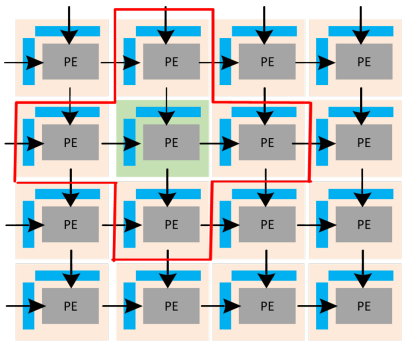


Fig. 14: Example of a 2D unidirectional pipelined dataflow in 4x4 PE-based systolic array [134].

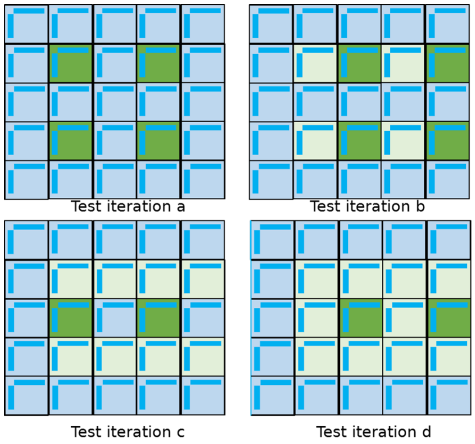


Fig. 15: Testing 5x5 PE-based systolic array in a checkboard style [134].

used to detect faults in the dark green PEs with high coverage. Each iteration will cover a different set of PEs. Light green PEs are those already covered by previous runs. As a result, for a large systolic array (e.g. 256x256), most PEs can be covered by small ATPG patterns in 4 test runs. The uncovered PEs on the borders can be fully tested in a top-off run.

4) Functional test generation: Functional test generation aims at generating inputs, e.g., images, that are capable of sensitizing the fault and propagating its effect to the output, leading to a different prediction with respect to that of the nominal fault-free network. This approach has been demonstrated for ANNs [127], [146]–[149], including memristive crossbar array-based architectures [146], [147], [149], and for SNNs [150], [151]. As shown in Fig. 16, functional tests could be original images from training and testing sets, adversarial examples generated from original images, or synthetic images generated from original images.

More specifically, starting from the available set of input samples, one approach is to select samples that are profoundly similar to other samples belonging to different output classes, i.e., a similarity metric could be average pixel intensity [127]. A second approach is to select samples that have been predicted correctly but with least confidence score [127], [151]. A third approach is to select samples that require more neural network parameter tuning effort during training, where the effort is measured with the change in the loss function in each training step [149]. In [146], [150], it is proposed to

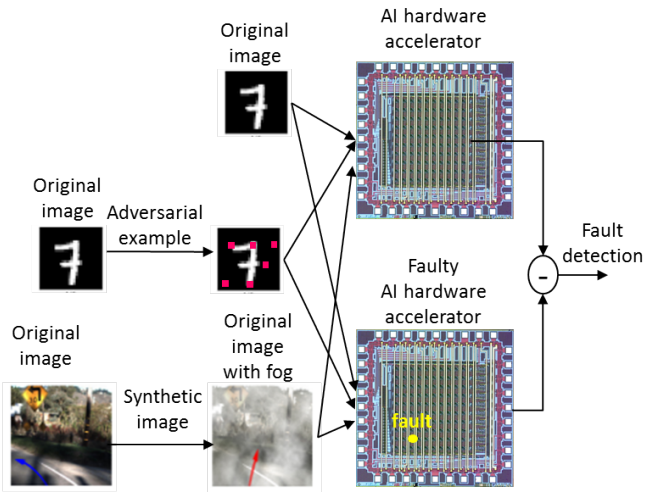


Fig. 16: Functional test generation. The street images are from [154]. The chip image is from [60].

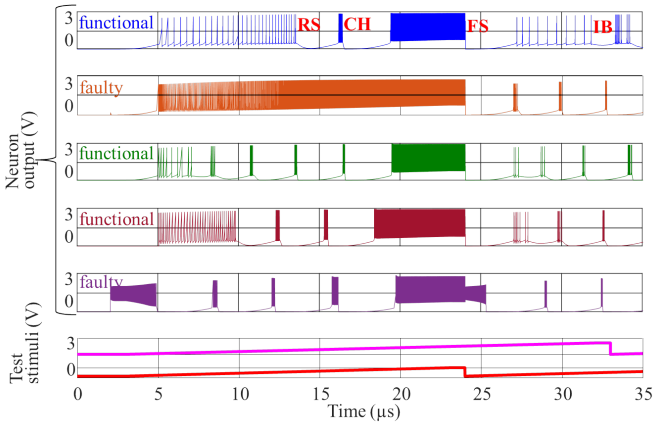


Fig. 17: Functional testing of a biologically-inspired spiking neuron.

generate adversarial input samples, i.e., perturb available input samples by adding a minimum amount of noise aiming at forcing the predictions of the nominal and faulty network to differ. Another strategy is to craft new samples by attaching watermarks to available input samples [147]. The network is deliberately trained to output a designated classification label for a watermarked input, a technique that is called backdooring. Using the watermarked inputs as validation set, a low validation accuracy indicates a fault. In general, in all aforementioned works, the objective is to use samples that are more vulnerable to misclassification when faults occur. Finally, in [148], a method is proposed for querying a network with a set of specially crafted test inputs, in order to reveal if the model parameters stored in the memory are faulty.

Functional tests can also be employed by the user of the programmed AI hardware accelerator to validate that the embedded DNN model has not undergone any malicious perturbations [152].

A related research direction is generating error-inducing corner test cases for a trained DNN, which thereafter can be used to retrain the DNN and improve its accuracy [153], [154]. These corner test cases are synthetic real-world input images resulting from realistic transformations of seed images and generated in a way such that they activate a large percentage of neurons in the DNN. For example, for DNN models controlling the perception of autonomous cars, these transformations include changing brightness, changing contrast, shearing, rotation, blurring, fog effect, rain effect, etc.

In [155], an alternative functional test generation is proposed, demonstrated for memristive crossbar-array architectures targeting detection of classification accuracy drop due to process variability. This approach is inspired from the alternate analog circuit testing paradigm [180], [181]. First, a compact test set of input images is generated with the maximum possible diversity of responses, and a feature vector is defined at the output of the network. An outlier detector in the form of an one-class classifier is trained in the space of features using as training set instances of the DNN with process variations but with acceptable accuracy. Applying the compact test set, the outlier detector serves as a first screening of non-conforming devices. Devices that pass this test are presented to a regressor which is trained using the same training set to map the features to the DNN classification accuracy. A guard-band is defined around the minimum tolerated accuracy to classify devices as passing, failing or fuzzy, where the fuzzy devices fall within the guard-band and are subject to standard testing using the complete image set to obtain a precise decision. Failing and fuzzy devices found in production testing can be combined in batches with prior training data to retrain the outlier detector and regressor.

In [156], a functional BIST scheme is proposed for biologically-inspired spiking neurons. The idea is to test that the neuron is capable of producing all the basic firing patterns, i.e., regular spiking (RS), fast spiking (FS), intrinsic bursting (IB), and chattering (CH). The test stimulus is composed of low-resolution ramps applied at the bias nodes of the neuron such that in one pass all firing patterns appear. If one or more firing patterns are missing, then the neuron is declared to be faulty. Examples of functional and faulty neuron responses are illustrated in Fig. 17.

C. On-line test

1) ATPG and functional testing: Many AI accelerators are used in datacenter applications, where reliability, availability and serviceability requirements demand certain level of on-line test of memories and logic. However, such tests involve both function mode and test mode, imposing more difficulties to DFT designs.

A recent effort is reported by Amazon AWS in [182], where high-speed serdes I/Os in an AI hardware accelerator are used to transport scan test patterns to test the processing cores. Test patterns are converted to a format compliant with corresponding protocol and transported from PCIe/USB, through standard AXI fabric, to cores. Cores under test will be idled from workload and isolated from the rest of logic. Although the major advantage of this solution is test time reduction, it supports native on-line test capability, which is critical in a cloud scenario.

Returning to the functional test generation methods in [127], [146]–[151] discussed in Section V-B4, as the resultant functional test set is compact, it can be also fed periodically during mission mode in idle times towards functional safety.

In [158], different self-test approaches are proposed for the compute units and control units of an accelerator. For compute units that do not contain complex sequential logic, test patterns based on combinational ATPG are generated. For the control units that contain finite state machines and sequential logic, it is proposed to use functional tests in the form of executing DNN layers with carefully-crafted input and weight values. The methodology is enhanced in [159] to cover both stuck-at and delay fault models for both unit types.

2) Checksums and error codes: The idea here is to build invariants into the accelerator that hold true only in fault-free operation while they are violated in the presence of faults. Thus, checking them concurrently with the operation can point to abnormal operation. Invariants can be built using checksums or error codes.

In [161], a sanity-check mechanism is proposed, in which error detection checksums are constructed by utilizing the linearity property of DNN MAC operations. These linear algorithmic checksums are added into the convolutional layers and fully connected layers of DNN models after the training. A hardware-based solution is proposed for integration into DNN inference accelerators aiming at reducing the performance overhead at the cost of a minor area and power overhead.

In [162], additional penalty terms, called balanced checksum, are introduced into DNN training. The balance checksum aims at forcing the DNN layer outputs to adhere to a linear invariant. With adding the balanced checksum into the cost function, error-checking invariants are embedded in DNN model computations. These invariants provide the computation error detection capability during the DNN inference phase, assuming the error would lead to the violation of the trained equilibrium. Furthermore, the introduced custom regularization terms even help a better generalization during the training.

In [163], several Algorithm-Based Error Detection (ABED) techniques are presented focusing on the verification of convolution operation, one of most resource-demanding operations in CNNs. Three variants of ABED are presented to use checksums for filters only, input feature maps only, or both filter and input feature maps. Implementation complexity, runtime overhead, resilience and performance trade-offs are studied and compared for the three ABED techniques. This work also address the overflow challenges of the checksum arithmetic induced by reduced-precision fixed-point operations (e.g., 8-bit integers). Resilience improvements are evaluated using analytical models, error injection experiments, as well as accelerated radiation experiments.

In [164], an AN code based fault detection mechanism is proposed to protect the MAC units of the DNN accelerator. AN codes add redundancy in the data to detect faults during arithmetic operations.

In the case where the weights of the DNN model are loaded encrypted in the memory, an on-line test scheme is proposed in [165] that employs the padding bits. Padding is used to add a number of bytes to the plaintext to reach a multiple of 16 bytes (i.e., 128 bits) since encryption is performed on 128-bit blocks. According to the most popular standard, if n bytes are added to pad the plaintext, then each of the bytes will encode the value n. If a bit-flip occurs, the on-chip decryption module will spread it creating multiple bit-flips affecting also the padding bits. The fault detection scheme is then to check that the decrypted padding bytes indeed encode the value n. Using this approach, most single faults become critical, but they become detectable at the same time.

3) Software-based: In [157], on-line test strategies based on Software Test Libraries (STL) are proposed for embedded systems running ANN applications. STL is composed of self-test routines that are executed during boot-time or run-time. The strategies are categorized into two groups according to whether they incur or not a small penalty in the inference time. Zero-penalty strategies include: (a1) run part of the STL during weight data transfer when the PEs are idle; (a2) test the inactive PEs of a low-intensive computation layer and cover all PEs in subsequent inferences using a scheduler based on a round-robin algorithm. Small-penalty strategies include: (b1) one PE is executing a self-test while the rest of the PEs share the AI workload; (b2) apply the entire STL between two consecutive inferences; (b3) arrange and apply the entire STL between successive layer computations when weight data transfer is happening. The strategies are evaluated on CNNs running on an open-source RISC-V platform. First, STL is verified to have a high stuck-at test coverage. Then, the different strategies are evaluated based on the inference time penalty and Fault Detection Time (FDT) trade-off, where FDT is the worst-case time to detect a fault from the moment of occurrence.

4) Memristor crossbar arrays: An on-line concurrent fault detection method for memristor crossbar arrays is proposed in [160]. The underlying observation is that faults affect the dynamic power consumption. An indirect simplified measure of the dynamic power consumption is used, in particular the number of logic '1s' at the outputs of the ADCs digitizing the output current of the crossbar's columns. An adder-tree design is used to count the number of '1s', which incurs a small area overhead. The time-series corresponding to the power consumption and count of '1s' show a strong correlation when faults are present, which allows using the count of '1s' as a simplified metric. When abrupt changes occur in the time-series data, the presence of faults is indicated. Changepoints are detected by examining time-series within a sliding window. For a current time point, the sliding window is centered on it. The probability density functions of the points in the left-hand and right-hand segments are estimated and compared to examine if the current time point is a changepoint. When a changepoint is detected, the percentage of faulty cells in the crossbar is estimated. A regression model is trained for this purpose off-line. A variety of independent feature variables are used, including statistics of the time-series data, average weight stored in the crossbar, and average input applied to the crossbar's input. Error correction is invoked when a high percentage of faults is estimated. For example, the faulty crossbar can be replaced with a redundant crossbar.

D. Yield improvement

In [166], the Yield and Accuracy aware Optimum Test of AI accelerators (YAOTA) framework is proposed. The framework deals with stuck-at faults in MAC units of AI accelerators and considers output bit position K up to which the inference error is acceptable. Faults in the fan-in logic cones of bit positions lower than K are considered non-critical, while fan-in logic cones of bit positions higher than K are considered critical. ATPG test patterns are applied and if only non-critical faults are found in the MAC, then the PE may be acceptable depending on how many such faults exist, the AI workload, and the error tolerance limits demanded by the application. If critical faults are found, then the PE is permanently disabled. The map of faulty PE locations is programmed in a fault status register. For Single Instruction, Multiple Data (SMID) architectures where PEs are interconnected with NoC/mesh, PEs can be individually switched-off and bypassed. For systolic array-based accelerators, a deactivation protocol is proposed without hardware-level modifications. In particular, it is proposed to deactivate the PE columns that contain PEs with critical faults and shift input data by inserting dummy rows of zeros. This approach has no area overhead but decreases the execution throughput. By adopting this framework, the manufacturer can avoid discarding the full accelerator chip because of the presence of a few faulty PEs, thereby increasing yield.

In [115], two methods are proposed to recover the fabrication yield loss of memristor crossbar-based accelerators due to high memristor defect rates. The first method consists of identifying memristors that are stuck at certain conductance levels and perform re-training of the network where only defect-free memristors are adjustable. In the case where the performance loss cannot be fully compensated by re-training, the second method presents a re-mapping algorithm where memristor columns that are heavily polluted, i.e., contain many defective memristors, are replaced by additional redundant columns.

In [167], spatial redundancy-based fault tolerance schemes are proposed for yield loss recovery of memristor crossbars. The fault model considers stuck-at fault in memristor cells, i.e., a memristor cell can be stuck-at a high resistance state or low resistance state. The fault-tolerance schemes apply to designs where the dot-product operation is mapped to two memristor crossbars. In particular, once the model is trained, the mapping allocates the positive weights to a “positive" crossbar and the negative weights to a “negative" crossbar. The proposed fault-tolerant mapping algorithm is to make the positive and negative weights eliminate the impact of faults on each other. For example, if a positive cell is stuck-at, the weight of the negative cell is enlarged accordingly to approximate the target weight. This approach works if only one of two cells in the same location in the two crossbars is faulty at a time. Spatial redundancy schemes are proposed in the case where the fault rate is high, where the same concept of pairwise fault elimination is used. These schemes make use of redundant crossbars, crossbar columns, and cells.

In [168], methods are proposed for improving the yield of memristor crossbars in the presence of memristor resistance variations and stuck-at faults. It is assumed that the resistance variations and the location of stuck-at cells can be detected. Two different methods are proposed in the case of Multilayer Perceptrons (MLPs) and CNNs. For MLPs, the problem of mapping the weight matrix of the trained model to the conductance matrix of the crossbar is formulated as a bipartite matching problem. The metric used is the summed weighted variations across the cells. Then, in a second step, the derived new weight matrix is fed as a starting solution to an off-device training algorithm. The algorithm aims at iteratively reducing the weight with the maximal deviation. This is done by scaling down the weight in each training epoch and adapting the weights of the surrounding cells to recover the classification accuracy. For CNNs, the method exploits the fact that two memristor crossbars are used to represent positive and negative weights since the conductance of a cell can only be positive. The weight is expressed as the difference between the two conductances. Therefore, there is a bit-wise redundancy in the architecture. The proposed method is to reprogram the resistance of one cell of the pair to eliminate the resistance variation in each cell. The same principle is used as a self-compensating mechanism to tolerate stuck-at faults. In a second step, off-device training and on-device training with few iterations so as to consider the limited endurance of the memristors can be performed to improve the classification accuracy.

A common technique to improve the error-resilience of DNN accelerators is to extract the memory fault map using post-manufacturing testing and perform fault-aware retraining of the model. Doing so for each faulty chip results in significant retraining overhead. In [169], it is proposed to train many faulty chips at a time. The fault maps of chips are merged into a unified fault map, which is then used for re-training a single model that will be loaded to every chip. A fault map is abstracted as a two-dimensional table where an element corresponds to a memory cell. The state of the cell is encoded to 1 or 0 for a S-A-1 or S-A-0 fault, respectively. For contradictory locations where S-A-1 of one fault map overlaps with S-A-0 of another fault map, the policy is to select the polarity that incurs less accuracy drop in the DNN inference. The re-training speedup increases with the number of fault maps merged. However, this speedup is at the expense of accuracy drop compared to per-chip retraining. Empirically it was observed that exposing the DNN gradually to faults rather than exposing it to all faults from the very start allows the DNN to learn at a faster rate and achieve better accuracy.

E. Fault modeling and testing of memristor-based memory technology

Memristors offer a compelling solution to the scalability problem of AI hardware accelerators, as they can be used as nano-scale synapses. They offer also the promising in-memory computing architecture that solves the data transfer bottleneck, as discussed in Section III. Besides these applications in an AI hardware context, memristor-based memory technology has a large potential for replacing traditional memory technologies and is in the focus of today’s research. As memristors are

susceptible to Process, Voltage, and Temperature (PVT) variations and manufacturing defects because they are fabricated with new materials and processes, there is a large body of works that aim at understanding such failure mechanisms and accurately model them to develop optimal post-manufacturing memory tests, including march test algorithms and DFT [170]–[177]. These works find applicability in the context of AI hardware accelerators implemented with memristive crossbar arrays.

VI. Dependability

A. Introduction

AI hardware is generally integrated in some intelligent or autonomous systems required to operate throughout their life cycle in a highly dependable manner. The dependability issues of AI hardware have aroused great interests in recent years. Dependability is a broad term used to define the ability of a system to deliver its intended service [183]. Any system, including AI hardware, can be viewed as a group of components integrated into one single entity to serve the purpose of delivering a certain service (e.g., AI algorithm acceleration for an AI accelerator). Throughout the life cycle of system deployment, there may be a service failure triggered by intrinsic or extrinsic effects, whereby the delivered service deviates from the intended one. Dependability of a system is the ability to avoid such service failures that are beyond the acceptance level.

Dependability encompasses a broad spectrum of attributes, which are quantities to measure dependability from various perspectives. The main attributes include reliability, availability, maintainability, and safety [183], [184].

- Reliability, availability and maintainability are three highly related attributes, which are usually measured by statistical metrics. Reliability denotes continuity of the correct service. The level of reliability is commonly specified in terms of Mean Time To Failure (MTTF) [184]. Maintainability denotes the ability to repair when a service failure has occurred. It can also be specified as a statistical term with the Mean Time To Repair (MTTR) metric which represents the expected system down time (including repair time) [184]. Lastly, availability denotes readiness for correct service. It can be expressed as a function of MTTF and MTTR as A=MTTF/((MTTF+MTTR)) [184].
- Safety denotes the ability of a system to not cause harm to people, things or the environment. Safety includes Functional Safety (FuSa) and Safety of the Intended Functionality (SOTIF). FuSa is defined as the absence of unreasonable risk due to hazards caused by malfunctions [185]–[187]. On the other hand, SOTIF focuses on absence of risks caused by performance limitations of the intended behaviors or by reasonably foreseeable misuse by the user [188].

We note that security is often not characterized as a single attribute of dependability. While highly related with dependability, security is considered as a composite notion combing confidentiality, availability and integrity attributes [184].

From dependability perspective, there are various threats leading to potential violation of the targeted goal. A threat at the component operation layer is usually called as fault. There are two main categories of faults: intrinsic faults and extrinsic ones. The former may be originated by aging effects, device variability, latent manufacturing defects, susceptibility to environmental conditions (e.g., radiation causing soft errors, electrical/mechanical stress). On the other hand, there are some faults caused by system inputs (e.g., malicious inputs to AI system, or user misuse), which are said to be extrinsic or external.

There are a variety of techniques to improve dependability of a system, including fault prevention, removal, tolerance and prediction. Fault tolerance is one of the most popular means aiming at tolerating a fault in a functional system. There are different levels of fault tolerance requirements, e.g., fail-operational, fail-safe, etc. A fail-operational system upholds the continued functionality and intended services in the presence of a fault. There are two main subcategories: upholding service without performance degradation, and with degraded performance. The latter is commonly referred to as fail-degraded or fail-reduced. A fail-safe system aims at transitioning the system to a well-defined condition to maintain a safe state in the event of faults. Functional safety mechanism is one example of means to achieve a fail-safe property. Note that there is another term of "fail-silent", which is described as the guarantee of no service (e.g., no system output) in the event of failures. Such a silent state can be viewed as a specific defined safe state, thus from this perspective fail-silent can be considered as a subcategory of fail-safe.

As AI hardware provides service to more and more mission-critical or safety-critical applications, these hardware elements need to be evaluated for compliance with the dependability goal (e.g., safety). In general, they share the same dependability theory foundation and requirements as other hardware (e.g., traditional general-purpose processors). However, there are several novel dependability challenges as well as opportunities introduced by unique characteristics of AI hardware computing architecture, application and also R&D cultures.

First, AI hardware goes under a new computing paradigm called "Domain-Specific Computing" [189]. Domain-specific computer architecture with domain-specific hardware acceleration has been introduced in recent years to address performance needs that general-purpose computing is hard to meet. This emerging computing paradigm shift is expected to bring new opportunities to AI hardware dependability method development. For example, while many traditional application (domain)-agnostic fault tolerance techniques, e.g., ECC or TMR, are commonly used in general-purpose computing, an alternative technique with exploiting domain-specific characteristics of AI hardware could be pursued to achieve better efficiency in terms of PPA.

As discussed in Section IV, many research works suggest that DNNs have inherent resilience to moderate variation of parameters and activations. Such approximate nature of DNNs enables development of approximate computing to support efficient AI learning in resource-constrained hardware, especially for inference. However, the actual impact of AI hardware

faults could be more severe on AI application service results, e.g., classification accuracy. It demands thorough hardware fault analysis and novel lightweight fault tolerance techniques exploiting architectural properties of AI hardware.

We also note that some research suggests deep learning models may have inherent weakness against input perturbation e.g., adversarial examples. Adversarial robustness of DNNs has received particular attention and there is a rapidly growing body of research work in this field [190], [191]. Such adversarial inputs can be viewed as external/extrinsic faults. In this survey paper we focus on dependability against intrinsic faults, i.e., those induced by AI hardware internally while potentially stressed by environmental effects or workloads. Other security threats that will not be covered in this survey include DNN model IP theft [192]–[194], backdoor attacks on DNNs performed when training is outsourced [195], [196], and fault injection attacks [197], [198].

Another characteristic of AI hardware is that it is often an integral part of some AI-based solution consisting of multiple interacting system layers – from hardware/physical to software/application. From this perspective, AI hardware dependability strategies should use a system-based approach beyond the techniques limited to local hardware. The concept of cross-layer dependability or cross-layer resilience [199], which leverages the inherent fault-tolerance of multiple layers, should be used for AI hardware to exploit domain-specific fault at the system level. Moreover, heterogeneous computing containing AI accelerators along with general-purpose CPUs and/or FPGAs has gained mainstream adoption in computing industry [200]. There exist far greater opportunities for exploiting heterogeneity to achieve system-level dependability.

Deep learning-based AI has become a revolutionary tool in many industry fields, with seemly-unlimited potential to outclass traditional techniques. This is a burgeoning field filled with the opportunities as well as chaos, much like the new kind of "Wild West". We see industry and academia are eager to push out AI innovations, with new architectures and higher performance expressed mainly with the Tera Operations Per Second (TOPS) metric, so as to battle for technical leadership in this rapidly growing field. In general, the AI field is permeated by a pioneering and risk-taking spirit. On the other hand, conservatism is fundamental in the dependability field (specially for safety). It is in a sharp contrast with the pioneering spirit and self-regulation philosophy. A paradigm shift is needed to bridge the gap between them. Over the past few years there has been growing efforts on this direction. For example, Europe has started legislation to make the use of AI safer and more ethical, such as in critical infrastructure impacting people's lives and health [201]. Still there is a considerable gap between the AI dependability goal and the available solutions. This is a research frontier where the technical community can contribute more to bridge the gap by introducing new methods.

Design-for-dependability aims at enhancing the reliability, availability, maintainability, and safety features of the AI hardware accelerator. All these attributes boil down to rendering the AI hardware accelerator error-resilient. We classify the existing design-for-dependability approaches into four categories, as illustrated in Fig. 18. The first category includes model-based approaches where the goal is to derive a model that meets the performance requirements and additionally it has intrinsically built-in or programmed error-resilient capabilitiessuch that by construction when mapped onto hardware it is capable of tolerating certain hardware-level faults. The second category includes proactive hardware-based techniques where the goal is to make the accelerator design passively tolerate certain hardware-level faults. The third category includes reactive hardware-based techniques where the goal is to make the accelerator react to an occurring fault in real-time, including built-in monitoring of fault occurrence and low-latency error recovery whenever a fault has occurred. The final fourth category includes cross-layer approaches where the error tolerance objective is shared between model and hardware.

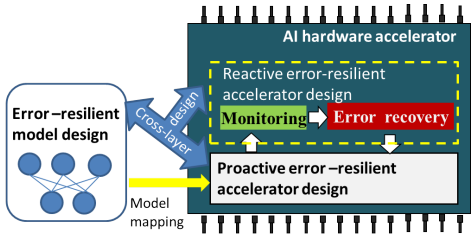


Fig. 18: Design-for-dependability approaches.

A taxonomy of existing techniques under the different categories is provided in Table III. These techniques will be presented in more detail next.

B. Model-based approaches

Fig. 19 combines and illustrates model-based approaches that will be discussed next in detail.

1) Model training modification: A number of works propose to achieve fault tolerance by modifying training. A first method is to add artificial faults and noise into the network during training such that the network learns to tolerate faults [61], [202], [203]. A second method is to restrict weights to have low values since intuitively fault tolerance degrades by the use of large values [202]. A third method is to add a penalty term to the training cost function that takes into account errors that arise due to faults, and multiply the penalty with a regularization parameter that controls the trade-off between the degree of fault tolerance and inference accuracy. The underlying idea is to bias the solution toward a fault-tolerant network. Approaches in this category include constraining the weights to lie within a limited range toward an even weight distribution [204]–[206]. A fourth method is to combine the training process and fault tolerance objective into an optimization problem solved by nonlinear optimization algorithms with the aim to learn a network model that performs the desired task and at the same time fulfills fault tolerance constraints [207]–[210]. A fifth method proposed in [211] considers a constructive training in the presence of faults, where neurons are incrementally added whenever the network fails to learn until a satisfactory learning or a user-defined maximum network size is reached.

TABLE III: Taxonomy of design-for-dependability approaches.

Model-based	Proactive hardware-based	Reactive hardware-based	Cross-layer
Model training modification [61], [90], [91], [202]–[223]	Memory cell re-design [224], [225]	Weight-shifting [226]	Model/hardware co-design [227], [228]
Model modification [229]–[231]	Memory aging mitigation [232]	Re-learning [233]	Fault-aware pruning with re-training [95]
Fault-tolerant model search [234]	Activation clipping [57], [81], [235]–[238]	Algorithmic-based fault-tolerance [81], [239]–[244]	Fault-aware mapping [101], [245]–[249]
	Redundancy-based [77], [85], [91], [101], [104], [250]–[256]	Fault masking [91], [92], [257], [258], [259], [260], [261]	Variation-aware mapping for memristor crossbar arrays [262], [263]
	ECC [81], [83], [84], [264]	ML-based [265]	Adaptive training after testing [146], [266]
	Razor [257], [267], [268]	Neuron adaptation [269]	Aging-aware on-line training of memristor crossbar arrays [270], [271]
	Hardening against radiation [272]		Thermal-aware optimization of memristor crossbar arrays [273]–[275]

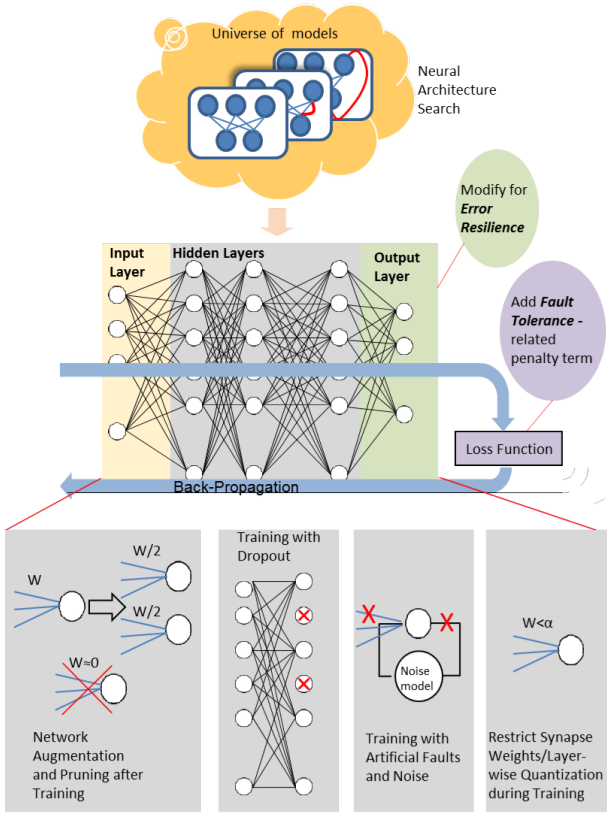


Fig. 19: Model-based approaches.

The aforementioned approaches are early works targeting shallow FC networks and considering faults at behavioral-level. They laid the foundation of several approaches for modern AI hardware accelerators presented recently which are discussed next. A thorough and comprehensive review of these early approaches is provided in [212].

a) Fault-aware training: In [213] and [91], it is demonstrated for ANNs and SNNs, respectively, that training with dropout improves the error-resilience. Dropout was originally proposed in [276] to prevent over-fitting and reduce the generalization error on unseen data. The idea is to temporarily remove neurons during training with some probability p, along with their incoming and outgoing connections. At test time, the final outgoing synapse weights of a neuron are multiplied by p. For a network with n neurons, there are 2^n "thinned" scaled-down networks, and training with dropout combines exponentially many thinned network models. The motivation is that model combination nearly always improves performance, and dropout achieves this efficiently in one training session. The reason why dropout is a natural fault-aware training approach is that it equalizes the importance of neurons across the network, resulting in more uniform and sparse activity across the network. Therefore, if a neuron becomes faulty, this turns out to have no effect on the overall inference accuracy. In [91], it is demonstrated that training the SNN with dropout can nullify the effect of dead neuron faults and neuron timing variations in all hidden layers, while the SNN can withstand a multiple fault scenario with high dead neuron rates. A technique equivalent to dropout, called erasure regularization, is to set neuron activations and weights to zero during training [214].

In [215], an error injection layer is developed that allows injecting faults according to a fault model during training time. The FINN FPGA-based QNN accelerator for CNNs [56] is adopted for the study. The focus is on two main fault types for CNNs, namely single channel stuck-at faults and same pixel in all channels stuck-at. Training is performed on a GPU and fault injection on FPGA. Results show that this fault-aware training approach: (a) improves the error-free accuracy behaving like a regularizer; (b) leads to highly fault-tolerant networks with accuracy very close to the error-free one; (c) offers an improved hardware cost vs. worst case accuracy trade-off when selective TMR is used to compensate errors in the most critical layers.

Another fault-aware training approach is to inject bit errors in the weights during the training process. This strategy has been investigated in [214], [216], [217] showing that it allows margin for voltage reduction in the memory of the DNN accelerator, thereby helping to reduce the energy consumption. In other words, the accuracy drop due to bit errors resulting from voltage under-scaling can be compensated by this fault-aware training approach.

b) Training with noise: In [218]–[220], it is shown for memristor crossbar-based architectures that injecting noise during software training enhances the robustness of inference to the non-ideal effects of memristor crossbars. In [218], a Gaussian noise source is incorporated at the crossbar outputs, while in [219], [220] a random noise term is injected to the

weights during training.

c) Co-optimizing inference accuracy and fault-tolerance: Techniques to unify inference accuracy maximization and fault-tolerance improvement optimization are proposed in [90], [221], [222]. In [90], a variant of an evolutionary optimization-based training algorithm for SNNs is proposed where the fitness function is re-designed aiming at improving the error-resilience capability. In particular, the fitness function becomes a weighted sum of the baseline accuracy and the average accuracy obtained on a faulty version of the network when imposing a certain synapse fault rate. In [221], process variations and noise are modelled as random variables and are incorporated into the weights of the neural network during training. In [222], a framework is presented that utilizes a Bayesian neural network to conduct a variation- and defect-aware training. The approaches in [221], [222] are demonstrated for memristor crossbar-based architectures.

d) Restricting numerical ranges: The range of parameters inside each layer of a DNN can vary a lot. This can be a major source of vulnerability to bit errors in DNNs. For example, considering a conventional fixed point data format, the variation in the first few MSBs can be very detrimental for small parameter values. In [219], it is proposed to use the Dynamical Fixed Point (DFP) data representation formation which allows to adaptively change the location of the decimal point based on the range of data. In particular, by left shifting the decimal point position we can make sure that there is no unused MSBs. In [223], to reduce the vulnerability surface, layer-wise quantization techniques are proposed to tighten the quantization margins to match the utilized range in each DNN layer. Also, a new regularization method, called outlier regularization, is introduced in the training phase to further tighten the numerical range and shape the parameter distributions.

2) Model modification: In [229], it is proposed to augment the trained network by replicating critical neurons and their associated connections. A neuron and its replica have half the weights of the original neuron to maintain the network mapping. The underlying idea is that if a critical neuron fails then the effect on the inference will be lower thanks to the spatial redundancy.

In [230], it is proposed to prune unimportant nodes in the network according to a sensitivity analysis and then re-train the pruned network. Redundant nodes are also introduced so as to share the task of critical nodes.

In [231], a method is proposed to enhance the error-resilience of DNNs by modifying just the output layer that performs the binary classification. Typically, an ensemble of independent logistic classifiers is used, each implementing a winner-takes-all rule by one-hot encoding. Error-correcting Output Code (ECOC) learning is applied to optimize the coding matrix and increase the Hamming distance of codewords assigned to different classes. This work proposes a collaborative logistic classifier extended from the logistic classifier to ease the neuron competition and improve the error capacity. Increasing the decision distance on final classification is shown to rectify the accuracy degradation induced by faults across the complete architecture. The method is cost-effective, scales to any network size, and can be easily integrated with existing hardware-level fault tolerance techniques.

3) Fault-tolerant model search: In [234], a Neural Architecture Search (NAS) algorithm, such as the one proposed in [277], is employed to discover a fault-tolerant architecture. The employed NAS algorithm uses reinforcement learning rewarding architectures towards maximizing performance. In this work, the NAS algorithm is modified to add a second term in the reward that expresses fault tolerance to bit flips (FT-NAS). Another version of the algorithm computes the first term of the reward, i.e., the classification accuracy, by inducing faults during training (FTT-NAS). The hand-designed networks show performance degradation already with a very small bit-flip rate. Instead, the network found by FT-NAS shows a graceful degradation with increasing error rate, whereas the network found by FTT-NAS achieves near baseline accuracy for high error rate. The discovered fault-tolerant architectures are inspected and they are found to establish double connections between some pairs of nodes. In other words, sensitive connections are identified by the algorithm and redundant paths are added for defending against faults.

C. Proactive hardware-based approaches

Traditional fault tolerance methods continue to play critical roles in AI hardware. For example, ECC is used to protect the memories of AI hardware accelerators [81], [83], [84], [264]. The Razor technique [278], aiming at detecting and correcting circuit timing errors, is also used in some AI hardware accelerator designs [257], [267], [268]. Besides these standard domain-agnostic fault tolerance techniques, there are other different proactive hardware-based approaches for AI hardware, illustrated in Fig. 20, that will be described in more detail in this section.

1) Memory cell re-design: In [224], a passive fault-tolerance method for ReRAM-based crossbars is proposed by re-designing the memory cell to have a 2-transistor/2-resistor (2T2R) structure, where each bit of information is stored in a differential fashion. In particular, the pair Low Resistive State (LRS)/High Resistive State (HRS) means logic value zero, while the pair HRS/LRS means logic value one. Readout is performed by comparing the resistance values of the two differential devices, thus doubling the memory read window with regards to the conventional 1T1R cell architecture, shown in Fig. 5. This differential architecture reduces the amount of bit errors due to device variations and limited endurance. Its benefits are demonstrated on a BNN. This inherent fault-tolerant architecture has auxiliary advantages. Weak programming conditions can be applied achieving energy savings. It also features outstanding endurance opening the way to the possibility of training neural networks on-chip.

In [225], a hardened SRAM cell is proposed for DNN accelerators. Based on the key observation of sparsity in DNNs, i.e., weights have a strong bias towards zero, and that bit flipping from zero to one is more likely to cause a failure of DNN outputs, the proposed memory cell provides robust immunity against node upsets and reduces the leakage current dramatically when zero is stored in the cell.

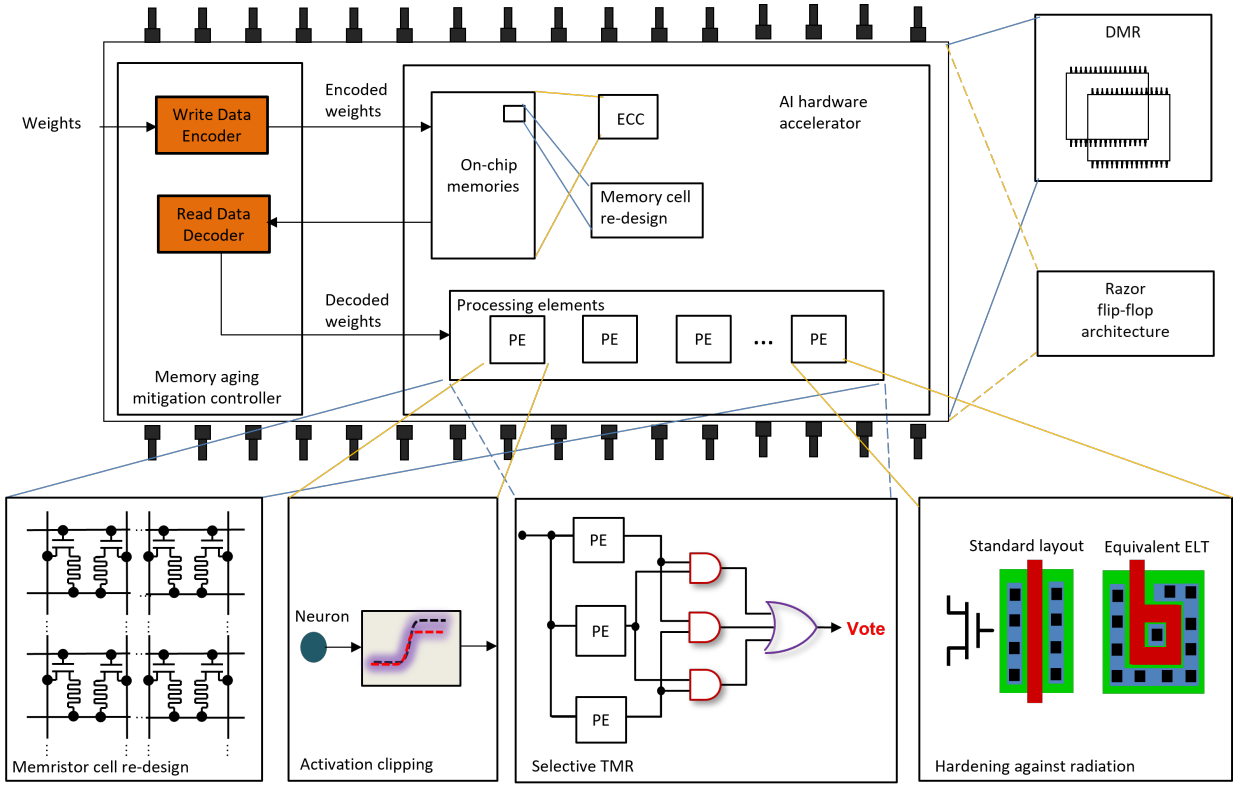


Fig. 20: Proactive hardware-based approaches.

2) Memory aging mitigation: A low-overhead aging mitigation scheme of weight memory buffers in DNN accelerators is proposed in [232]. The underlying observation is that optimized aging can be achieved by balancing the duty-cycle of the memory. To this end, a micro-architecture is proposed composed of a Write Data Encoder (WDE) for encoding the weights before writing them to the on-chip memory, and a Read Data Decoder (RDD) which performs the inverse function when reading the data from the on-chip memory and before passing it to the PEs. The WDE XORes the incoming weights with a common 1-bit enable signal that is generated by a True Random Bit Generator (TRBG), thus adding a sense of randomness on the bits to be written in the memory. The output of the TRBG is periodically inverted by XORing it with a bitstring stored in a register to account for the scenario where the TRBG is biased towards either '0' or '1'. The RDD performs the same XOR operation as the WDE on the outgoing bits. Results show that this scheme offers maximum aging-mitigation for any data representation and across different accelerators and DNN models.

3) Activation clipping: In [235], it is observed that as the fault rate increases, the activation of neurons becomes more intense. In [57], when the activation output of a neuron exceeds by 10% the expected range of values it is considered as a symptom of an error occurring. To this end, in [235] it is proposed to use a clipped version of the activation function such that when activation exceeds a threshold, then the neuron is silenced. A search algorithm using the area under the curve accuracy vs. fault rate as metric is proposed to find the optimal threshold that maximizes classification accuracy under different fault rates. This strategy is investigated also in [236]–[238]. In [236], values are truncated to the maximum value observed in the training set. While in [235], [236] activation functions are bounded globally per layer, in [237] the truncation value is fine-grained per neuron. In [238], to avoid the risk of false positives, it is proposed to compute several single statistics on neurons' output values, i.e., minimum, maximum, average, and standard deviation. If at least two different statistics are out of range, then a fault detection is flagged.

In [81], it is proposed to redesign the maxpool layer of CNNs so as to halter the fault propagation. The redesign consists in evaluating if the value of the max element is higher than a threshold and, if so, then halt the processing of the frame and move on to the next frame, or use the second largest element if it is reasonably small.

4) Redundancy-based: State-of-the-art AI hardware accelerators for autonomous driving vehicles employ Dual Modular Redundancy (DMR) to ensure safety for the system [250], [251], which requires substantial hardware resources.

One idea is to perform selective TMR applied to the most critical layers instead of a full TMR, which is inspired by the observation that different layers have different sensitivity to faults [85], [91], [101], [104]. Selective TMR is feasible resource-wise and the resultant area and power consumption overhead can be tolerated. Typically, the most critical output layer is protected with TMR, which is enough to achieve a high level of fault masking. For deep networks the output layer accounts for a small percentage of neurons of the whole network, thus the percentage overhead of applying TMR only to the output layer scales down.

Redundancy-based fault tolerance can also be applied at different hierarchy levels, for example TMR of critical kernels

[77], DMR of critical feature maps [252], TMR of MSBs in computational blocks such as adders and multipliers [253], and TMR of critical neurons [254].

In [255], a redundancy-based fault-tolerance strategy, called Hybrid Computing Architecture (HyCA), is proposed for the 2-D array of PEs that greatly reduces the overhead of the classical DMR. The basic idea is to add a separate set of Dot-Production Processing Units (DPPUs) in parallel to the original computing array of PEs. HyCA can be utilized to scan the entire 2-D array and detect the faulty PEs at runtime, and recompute all the operations that are mapped to the faulty PEs, independently of the location of faulty PEs.

Finally, a redundancy-based fault-tolerance strategy based on ensemble learning is proposed in [256]. Ensemble learning consists of training a set of independent smaller and weak (i.e, with lower accuracy) base networks, using different network structures, learning algorithms, and training datasets. Thereafter, the results are combined, i.e., using voting or averaging, to improve the task performance. The idea is that when one or more weak networks fail due to a fault, the ensemble of other networks can still operate reliably.

5) Hardening against radiation: In general, ionizing radiation, depending on the energy of the incident particle and the time of exposure, can give rise to transient events or permanent damage, such as bit-flips, shift in the transistor's threshold voltage, and increase in the leakage current. Transistor hardening refers to applying changes in the layout so as to tolerate exposure to ionizing radiation. In [272], a spiking neuron design is hardened by redesigning the transistors' layout using an Enclosed Layout Transistor (ELT) topology for the gate. This particular neuron uses a memristive device to implement the memory element, i.e. the membrane, of the neuron. The area overhead with respect to the original design excluding the memristive device is 4.51x. However, taking into account the memristive device, it is argued that area overhead is negligible because the memristive device is placed on top of the CMOS subsystem during the back-end phase which requires an extensive area.

D. Reactive hardware-based approaches

Fig. 21 illustrates different reactive hardware-based approaches described in this section, separating the two underlying mechanisms, namely fault/error detection and localization and fault/error mitigation.

1) Weight-shifting: In [226], the weight-shifting fault-recovery mechanism is proposed. If an incoming synapse of a neuron is detected faulty, then the loss is compensated by adapting the weights of other synapses. If a neuron is faulty, then its outgoing synapses are treated as faulty.

2) Re-learning: In [233], a high-level biologically-inspired model of the cortical structure of the brain is developed capable of performing feed-forward sensory processing and automatic abstraction for visual inputs. The model is trained using Hebbian learning with repeated exposure to input samples. A software version of the model is deployed on a GPU for fault tolerance experimentation. The fault model considers neuron stuck-at faults, i.e., neurons that do not

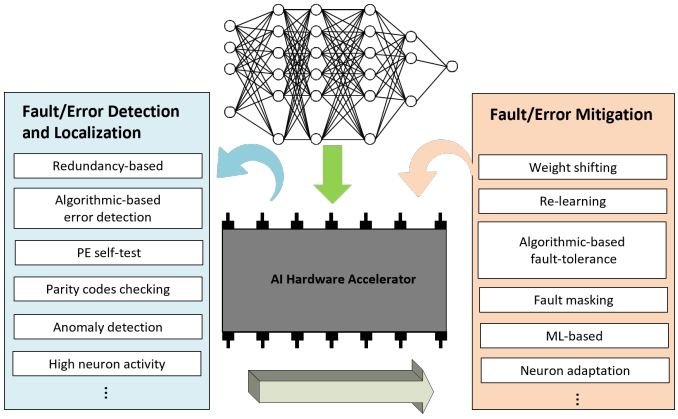


Fig. 21: Reactive hardware-based approaches.

fire when they should (stuck-at-0) or they fire when they should not (stuck-at-1). Single and multiple fault scenarios are studied including spatially distributed and clustered faults. For stuck-at-0 neurons, the network is capable of re-learning as their functionality is taken over by neighboring neurons. On the other hand, stuck-at-1 neurons can severely degrade the performance and upon detection are disabled and the network re-learns. Detection is performed by interrupting the operation and recomputing the response of the winning minicolumn of neurons on two neighboring minicolumns. A voting scheme is used to determine a defective minicolumn. This is a form of TMR but using the existing redundancy. The model's accuracy with re-learning shows a graceful degradation to faults and a large number of faults can be tolerated.

3) Algorithmic-based fault tolerance: Algorithmic-Based Fault Tolerance (ABFT), originally proposed in [279], is a low-cost solution for detecting and correcting abnormal behavior in matrix-matrix multiplications based on checksums. As neural network operation heavily relies on matrix-matrix multiplications, ABFT finds a natural application for enabling fault-tolerance in AI hardware accelerators with several ABFT schemes being proposed to date in the literature [81], [239]–[244].

As an example, in [242], the compute underutilization of inference-optimized GPUs is exploited by evaluating the computing resource bottleneck for GPU kernels. The metric being used is a comparison between the arithmetic intensity of the kernel (in GPU terminology a GPU kernel consists of multiple threads that can be executed in parallel) versus the Compute-to-Memory-Bandwidth ratio (CMR) of the GPU. A kernel is compute-bound if the arithmetic intensity is higher than the CMR; otherwise, it is memory-bandwidth bound. For a memory-bandwidth bound kernel (i.e., with low arithmetic intensity running on a high CMR hardware), there is an opportunity to leverage underutilization of compute units to allow ABFT execution on unused resources. Motivated by this observation, a finer-grained ABFT scheme is proposed, referred as thread-level ABFT, as illustrated in Fig. 22. Performing ABFT at the thread level can exploit compute underutilization of bandwidth-bound kernel to reduce execution time overhead of ABFT. Furthermore, an arithmetic-intensity-guided ABFT is proposed that selects the best ABFT scheme

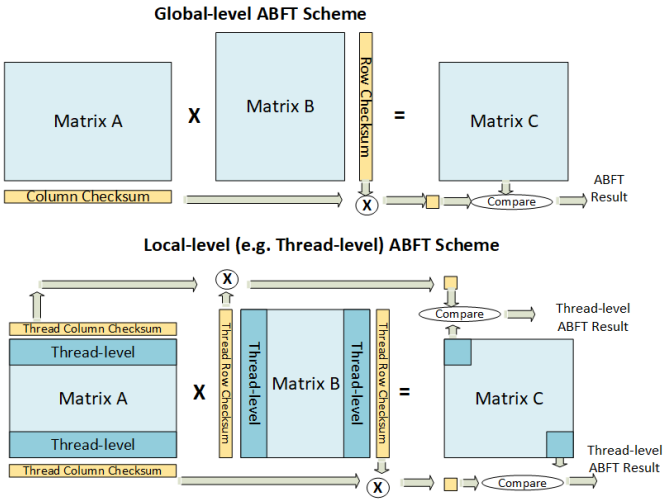


Fig. 22: ABFT global and local schemes.

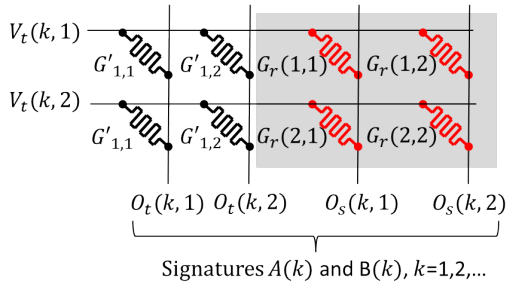


Fig. 23: Memristor crossbar checksums.

for each individual layer of the network, e.g., global (kernel)-level ABFT for compute-bound layer, and thread-level ABFT for memory-bandwidth-bound layers.

An implementation of ABFT for memristor crossbar-array architectures is proposed in [240]. As illustrated in Fig. 23, a crossbar of size $r_{xbar} \times c_{xbar}$ is partitioned into smaller crossbars of size $r_t \times c_t$. For each smaller crossbar two extra columns are added. In the first column, the cell in row i computes the non-weighted checksum $G_r(i,1) = \sum_{j=1}^{c_t} G(i,j)$, where $G(i,j)$ is the nominal expected conductance value of the cell in position (i,j) of the crossbar. In the second column, the cell in row i computes the weighted checksum $G_r(i,2) = \sum_{j=1}^{c_t} W_G(j) \cdot G(i,j)$, where $W_G(j) = j$. For each smaller crossbar M test input vectors are applied, denoted by $\mathbf{V_t}(k) = [V_t(k,1), \cdots, V_t(k,r_t)]$, where $V_t(k,i) = V_0 \cdot W_t(k,i)$, V_0 is a unit voltage, $W_t(k,i) = (f(i))^{k-1}$, $f(i) = 2^{i-1}$, $k = 1, \cdots, M$. The outputs of the two checksum columns for test input k are $O_s(k,1) = \sum_{i=1}^{r_t} V_t(k,i) \cdot G_r(i,1)$ and $O_s(k,2) = \sum_{i=1}^{r_t} V_t(k,i) \cdot G_r(i,2)$. The output of crossbar column j for test input k is $O_t(k,j) = \sum_{i=1}^{r} {}_t V_t(k,i) \cdot G^{'}(i,j)$, where $G^{'}(i,j)$ is the actual conductance value of the cell in the (i,j) position of the crossbar. Two signatures are defined for test input k, namely $A(k) = \left(\sum_{j=1}^{c_t} O_t(k,j) - O_s(k,1)\right)/V_0 = \sum_{j=1}^{c_t} \sum_{i=1}^{r_t} W_t(k,i) \cdot [G^{'}(i,j) - G(i,j)]$ and $B(k) = \left(\sum_{j=1}^{c_t} W_G(j) O_t(k,j) - O_s(k,2)\right)/V_0 = \sum_{j=1}^{c_t} \sum_{i=1}^{r_t} W_G(j) \cdot W_t(k,i) \cdot [G^{'}(i,j) - G(i,j)]$. In fault-free operation, $A(k) = B(k) = 0$. Based on the percentage of faulty cells, the size

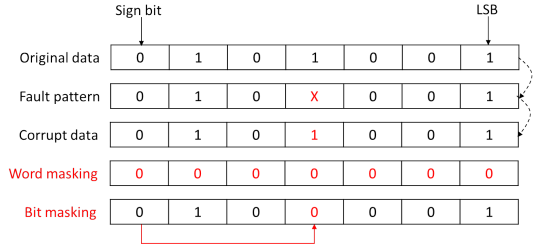


Fig. 24: Word and bit masking error mitigation techniques.

$r_t \times c_t$ is chosen such that no more than 2 faults occur in a small crossbar. In this case, using $M = 4$ test inputs we can perform fault localization and compute conductance deviations in faulty cells in both the crossbar and the checksum columns using the 2 signatures. In particular, we can write 8 equations with 6 unknowns, i.e., the fault locations, denoted by (x_1, y_1) and (x_2, y_2), and the conductance deviations, denoted by d_1 and d_2, for the two faults, and solve the system of equations with linear algebra.

4) Fault masking: In [92], [257], memory bit-flip mitigation schemes are proposed with no costly fault-tolerance operations relying on the sparsity of data. The assumption made is that information is available on which bits are affected, for example using Razor shadow latches that can detect faults by monitoring circuit delays [278]. The schemes are based on masking faulty bits. The two main schemes, namely word and bit masking, proposed in the case of fixed-point data representation, are illustrated in Fig. 24. Word masking sets all bits of the corrupted register to zero. This is equivalent to setting the synapse weight to zero which intuitively, due to sparsity of the network, will have a lesser impact on the accuracy as opposed to leaving uncorrected a $0 \rightarrow 1$ bit-flip in a high-order position. Bit masking sets a faulty bit equal to the sign bit and can tolerate more faults than word masking. It achieves a similar effect by rounding the synapse weight towards zero.

In [258], a soft error detection and correction scheme is proposed for CNNs accelerated on FPGAs. Fault injection analysis shows that Single Event Upsets (SEUs) on PEs are far more consequent than SEUs occurring in memory. Moreover, SEUs in MSBs are shown to be far more critical. It is proposed to execute a self-test of PEs during free cycles motivated by the fact that the average PE utilization ratio is usually below 85% during inference. The self-test consists in exercising the higher bits of multiplexers and adders in the PE separately, and this traversal overhead can be easily confined within the free cycle. Temporary error mitigation is achieved by using zero setting upon SEU detection, instead of re-configuring the PE immediately.

In [259], a fault-tolerant design of the systolic Output Stationary (OS) DNN architecture is proposed. Faults in the datapath, i.e., outputs of PEs, are detected on-line and mitigated. A functional on-line test approach is proposed where neighboring PEs are tested separately by applying the same input (i.e., one PE needs to be taken off-line) and checking if their outputs are identical. The fault mitigation approach is to mask the faulty PE's output to zero. As a PE roughly corresponds to a

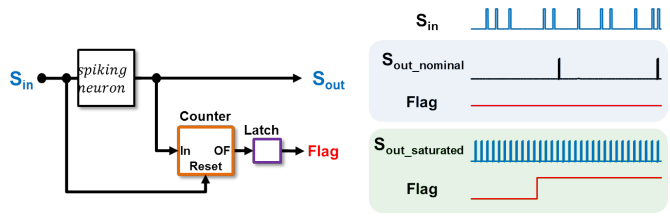


Fig. 25: Symptom detector for a spiking neuron.

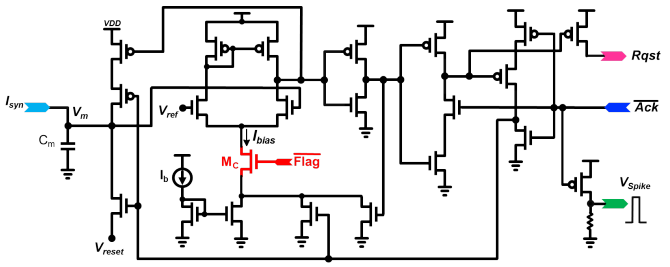


Fig. 26: Spiking neuron design with cut-off transistor enabled when the neuron starts saturating raising the flag signal high.

single neuron, performing training with dropout can augment robustness. This fault-tolerance approach shows no latency in the inference and in terms of area overhead it requires the addition of 3 MUXes per PE and an external comparator for the whole PE array.

In [260], the Opportunistic Parity (OP) fault mitigation technique is proposed for protecting CNN weights. OP is based on the observation that errors in the LSBs of the weights can be tolerated. The idea is to flip the LSB if needed such that the weight has even parity. Checking the parity code can detect an odd number of bit flips. Noting that a memory word can be large and multiple weights can be stored in one memory word, we can adjust parity for individual weights or for the entire memory word. When a parity error is detected, the weight values are replaced with zeros.

Regarding SNNs, with the passive neuron fault-tolerance scheme based on dropout in place, active neuron fault tolerance in hidden layers needs only to address neuron saturation (see Sections IV-B1b and VI-B1a) [91]. A compact on-line monitor can be used per neuron to detect this symptom [91]. The monitor, shown in Fig. 25, is based on a small-size counter that counts the number of spikes a neuron produces after every single input spike and has a reset port connected to the input of the neuron. A saturated neuron will produce spikes with higher frequency than usual, causing the counter to overflow before an incoming spike resets it again. A latch is set when overflow happens and an error flag is raised. On the other hand, in fault-free operation, the neuron needs to integrate multiple input spikes before it can produce a spike of its own, hence the counter is always reset, and the error flag signal stays at zero. If saturation is detected, the “fault hopping" concept is proposed as a recovery mechanism [91]. The idea is to turn a saturated neuron into a dead neuron since the network can withstand dead neuron faults. This simplifies the hardware implementation requiring adding a single extra transistor per neuron. An example is shown in Fig. 26 where a transistor shown in red is added to cut-off the biasing of the spiking neuron when the flag signal indicating neuron saturation goes high.

In [261], a run-time soft-error mitigation technique for SNNs is proposed. A fault criticality analysis shows that increased weights and neuron saturation are the only faults that can decrease inference accuracy. For synaptic faults it is proposed to perform weight bounding. In particular, if the weight is greater than a threshold, then it is replaced with a pre-defined value, i.e., zero or maximum weight value from the nominal SNN. For neuron saturation faults, if the membrane voltage stays above the threshold for more than two clock cycles, then spike generation is disabled similar to [91].

5) ML-based: In [265], a ML-based method is proposed to detect an anomaly in a DNN and mitigate the effect at run time. The fault model is transient faults in the form of random single bit-flips in the buffer memories and data paths of the accelerator. For a given input, each layer of the DNN provides a set of feature activations (i.e., the respective neuron output values). A unified feature activation trace is generated by concatenating the feature activations of all layers. Then, a small FFNN, named as Error Detection and Mitigation Network (EDMN), is trained in this feature space to perform anomaly detection due to critical bit-flips, as well as to predict and recover the correct classification result for error mitigation. The training data is generated by random bit-flip injection simulations recording the feature and classification result. Furthermore, the small EDMN can be safeguarded against faults by using classic methods, e.g., TMR.

6) Neuron adaptation: An application-specific fault-tolerant design of a SNN implemented in an FPGA in proposed in [269]. The SNN is used to control the motion of a robotic car, i.e., speed and direction, establishing an obstacle avoidance task. There are four motor neurons controlling the forward (F), right (R), left (L), and reverse (REV) movements. The neuron’s excitatory synapse receives input current according to the obstacle distance. The prioritization is achieved via the inhibitory synapses. The neuron’s spiking rate detects the activity of the corresponding motor, i.e., F, R, L, or REV. Fault tolerance is achieved by using many synapses instead of one receiving the same input. The neuron monitors the total injected current from all synapses during a time window, and if an abrupt or abnormal variation is noticed, then this points to a fault occurring to one or more synapses. Fault tolerance in this context means retaining the same firing rate. This is achieved with one of two mechanisms: (a) adjust the neuron’s threshold and (b) adjust the operating frequency of the neuron. Mechanism (b) is achieved via the dynamic partial reconfiguration feature of the FPGA that provides a way to generate custom clocks on-the-fly. The adjustment scheme is not continuous, but it is based on a Look Up Table (LUT) for given expected-erroneous pairs of input currents.

E. Cross-layer approaches

Fig. 27 illustrates together different cross-layer hardware-based approaches that will be described in detail in this section.

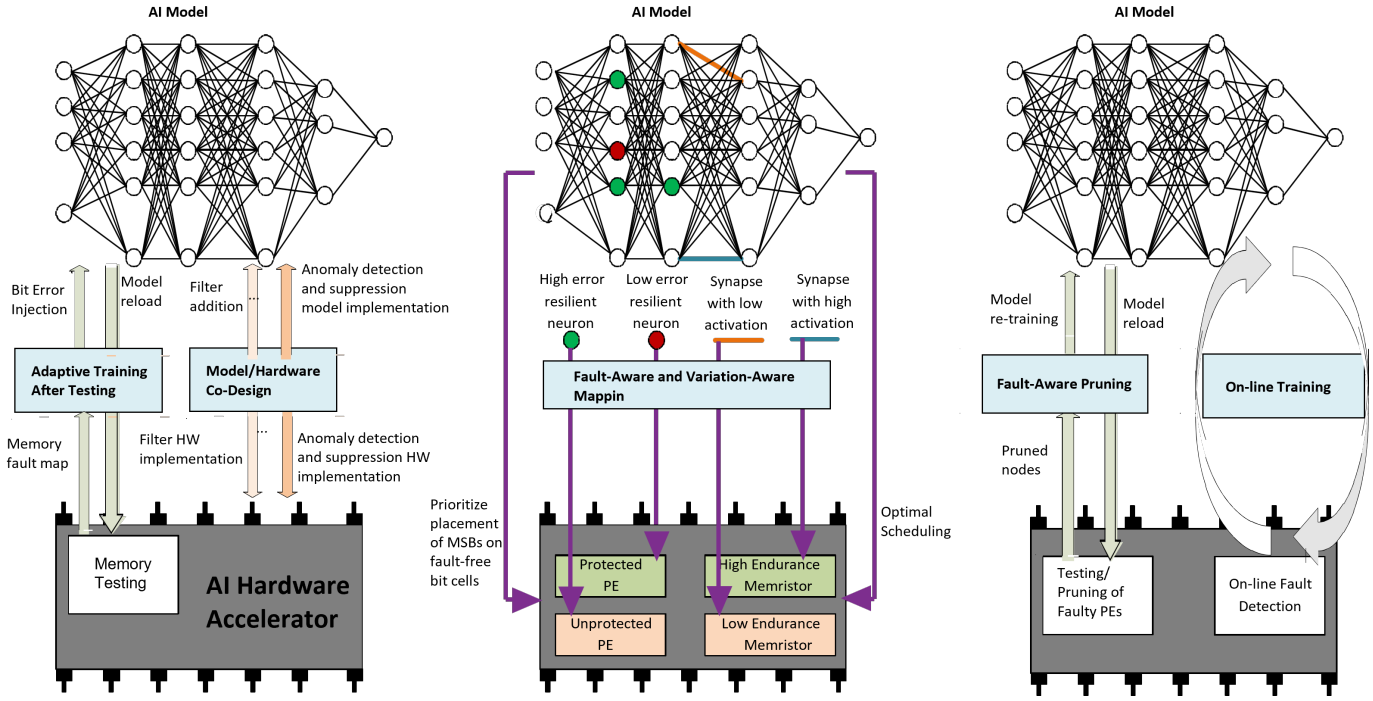


Fig. 27: Cross-layer hardware-based approaches.

1) Model/hardware co-design: In [227], a method is proposed to maintain DNN accuracy under high error rates by suppressing the numerical contributions of anomalous activation. It first integrates anomaly detection and suppression layers into DNN models. To address the training challenges due to the discontinuous nature of these layers, a two-stage training process is proposed to ensure a fast convergence with competitive accuracy. A hardware module is proposed to perform anomaly detection and suppression at the inference phase of the DNN accelerator.

In [228], a median feature selection technique is introduced to alleviate the impact of bit errors before the numerical operation of each layer. It is observed that the critical bit errors are often those leading to a significant numerical increase in the activation or weight magnitude. Such errors exhibit characteristics similar to the spike noise patterns in the image processing field, where order-statistics filters have been proven to be effective against large spike noises. Therefore, DNN models are first trained with integrated median filters. After achieving the desired accuracy in training, the model is deployed on the AI accelerator with dedicated hardware performing median filtering operations.

2) Fault-aware pruning with re-training: In [95], a fault-tolerance scheme is proposed for systolic array-based DNN accelerators, depicted in Fig. 28. In a first step, the scheme includes fault-aware pruning where the faulty MAC is bypassed using multiplexing, which is equivalent to setting the MAC's weight to zero. In a second step, the pruned systolic array is re-trained. It is demonstrated that this fault tolerance scheme can maintain a classification accuracy close to the baseline even when up to half of the MAC units are faulty. It is also demonstrated that with the fault-tolerance scheme in place, more aggressive voltage under-scaling can be employed

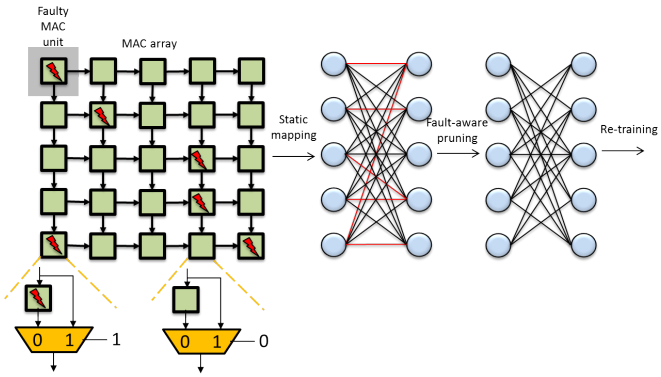


Fig. 28: Fault-aware pruning followed by re-training.

to provide energy savings while not sacrificing classification accuracy.

3) Fault-aware mapping: In [245], the underlying hypothesis is that neurons with strong contribution have a high impact on the inference accuracy if they are faulty, thus a strong contribution implies low error resiliency. To derive the ranking, an algorithm is proposed based on Taylor decomposition of the network and layer-wise propagation of the contribution. The average contribution is considered taking the mean over the training set. Thereafter, it is proposed to design an accelerator to have a number of protected PEs and memory buffers, where protected means that they are safeguarded against faults by utilizing spatial or temporal redundancy and error correction mechanisms. The neurons with the lower error resiliency are mapped to protected elements, whereas neurons with the highest error resiliency are mapped to unprotected and unreliable elements.

In the case of overlay architectures that consist of an array of PEs on which layers or portion of layers are scheduled

to be executed in sequence, whenever a single PE is faulty this affects multiple outputs both within a layer or among layers. Thus, the portion of the neural network affected by the corrupted PEs depends on the scheduling. One zero-overhead approach, therefore, would be to identify and utilize the optimal scheduling that minimizes the accuracy drop [101].

In [246], first the faulty PEs are pruned after testing, similarly to the approach discussed in Section VI-E2. Given the saliency of the weights, it is proposed to map neurons of a layer on different segments of the hardware such that the sum of saliency of the weights that are mapped on pruned PEs during inference is minimized.

In [247], it is proposed to first identify the most critical neurons, then determine an optimal scheduling that distributes evenly the critical neurons to the available PEs such that if a PE exhibits a fault this affects the functionality of a limited number of neurons.

In [248], it is proposed to first derive the memory fault map using testing, then apply a fault-aware mapping consisting in bit shuffling to prioritize placing the MSBs on the non-faulty memory cells. This strategy is also investigated in [249].

4) Variation-aware mapping for memristor crossbar arrays: Line-resistances degrade the voltage levels along the crossbar columns, thereby inducing more errors at the columns away from the drivers. In [262], it is proposed to rank the DNN kernels based on a sensitivity analysis and re-arrange the columns such that the most sensitive kernels are mapped closer to the drivers.

In [263], it is shown with circuit simulations that a memristor crossbar presents current imbalance, i.e., asymmetry in the current propagating through its different memristors. This current instability is due to the parasitic components on the horizontal and vertical wires of the crossbar that result in voltage drops. For example, the current on the largest path from a pre-synaptic neuron to a post-synaptic neuron, i.e., the path that traverses the top horizontal line through the upper right memristor and down the far right vertical line, is smaller compared to the current on the smallest path, i.e., the path that includes only the bottom left memristor. This current variation results in endurance variability of memristors in the crossbar, where endurance is defined as the ratio of average failure time and switching activity. For example, the memristor in the upper right corner will have higher endurance, whereas the memristor in the bottom left corner will have the lowest endurance. Based on this observation, the *eSpine* framework is proposed for endurance-aware mapping of SNNs to neuromorphic hardware. Given the SNN workload, the objective of eSpine is to find an intelligent mapping of neurons and synapses to neuromorphic hardware, such that synapses with high activation are implemented on memristors with high endurance, and vice versa.

5) Adaptive training after testing: In [266], a methodology is proposed, named Memory Adaptive Training with In-situ Canaries (MATIC), aiming at aggressive voltage scaling of weight SRAMs in AI hardware accelerators obtaining significant energy savings, while maintaining the inference accuracy. The idea is to perform read-after-write and read-after-read operations on each SRAM address of the chip, in order to generate a profile or failure map of the marginal, failure prone bit cells. Then memory adaptive training is performed where the profiled bit errors are injected in the training process enabling the DNN to compensate via learning. In this way, during normal operation, by applying voltage scaling a significant fraction of resultant bit errors is passively tolerated. Further, tuneable accuracy-energy trade-offs can be achieved by using a select set of bit cells that are on the margin of read failure as canary. Canaries are replicated critical bit cells that can detect imminent failures. Such in-situ canary bits can be polled at run-time to determine whether voltage modifications should be applied to maintain an advantageous accuracy-energy trade-off.

In [146], periodical on-line testing is performed using a functional test set of adversarial examples, as discussed in Section V-B4. If the network is found faulty, memory fault diagnosis is performed using a march test. If a soft fault has occurred, the remedy is to refresh the memory with a model back-up stored in the edge device. If the fault is permanent, the fault map is sent to the cloud for model retraining after masking the faults, and afterwards the model is re-transmitted to the edge device.

6) Aging-aware on-line training of memristor crossbar arrays: The writing endurance of memristor cells ranges from 10^6 to 10^8 write operations, whereas the training phase can take 10^5 to 10^7 iterations. Therefore, on-line training of memristor crossbar-based accelerators causes degradation of the valid resistance range of the memristor, an effect called aging in the memristor, resulting in most memristor cells becoming faulty. In [270], [271], frameworks are proposed combining software training and hardware tuning to counter the aging effect. For example, in [270], first the threshold-training method is introduced to reduce the number of write operations in each iteration. The observation is that for the vast majority of weights the weight update is very small. In this case, the weight update is suppressed. Second, after a fixed number of iterations, a fault detection method is executed to detect stuck-at faults and update the status of cells. The method, called quiescent-voltage comparison, consists in the following steps: (a) divide the crossbar into smaller crossbars; (b) for each crossbar perform a write operation with the same write change to every cell; (c) compare the actual crossbar outputs to the expected reference output and if a discrepancy is found then it means that at least one cell in the selected crossbar is stuck-at and cannot be updated when we write an increment. By using smaller crossbar size we increase fault detection accuracy at the expense of higher test time. After fault detection, the third method exploits the fact that over half of the weights are zero. The idea is to map zero weights to cells that have stuck-at-0 faults. This can be achieved by reordering the column and rows of the weight matrix. To do this efficiently while respecting the inherent connection of cascaded layers, it is proposed to reorder only the neurons.

7) Thermal-aware optimization of memristor crossbar arrays: Temperature increase changes the conductance value of a memristor cell and decreases its endurance. In [273]–[275], thermal-aware training and on-line optimization schemes are proposed to resolve the temperature-dependent retention issues

of memristors with one-time DNN deployment.

VII. Perspectives

A. Fault criticality assessment

Lesson learned from published fault injection experiments is that not all faults are equal. Most end up being benign, i.e., they are masked, their effect on the output is not large enough to result in accuracy loss, or the accuracy loss is insignificant. On the other hand, there are some suspect critical faults, e.g., $0 \rightarrow 1$ bit flips in high order bit positions, stuck-at-1 artificial neuron activations, saturated spiking neurons, and faults in last layers especially in the output layer. Yet, the locations of critical faults cannot be safely presumed and depend on many factors, such as the network architecture (e.g., depth, number of channels, etc.), the sparsity of the network (e.g., percentage of near-zero weights), and the cognitive task (e.g., dataset). Given that the fault space explodes for deep networks, to speed up reliability analysis one of the main challenges is reducing safely the fault space aiming at circumventing simulation of faults that would prove to be benign. For example, we can use ML to predict fault criticality and block the simulation unless a fault has some likelihood of being critical. This will allow avoiding speculative and unguided fault sampling and evaluating more faults, thus identifying with higher probability the critical faults that will need to be dealt with fault tolerance techniques. Identifying the critical faults will allow better targeting the fault tolerance strategies and reducing test costs.

Along this direction, we require faster automated fault injection frameworks. More “tricks" to speed up simulation can be integrated in current frameworks. For example, as every layer is computed sequentially, if a fault is masked in an intermediate layer, then simulation can be stopped early. Or, for a fault in a given layer, we can start the simulation from this layer considering the golden fault-free response of the previous layer.

A second main challenge is developing fault models for higher-level descriptions of the accelerator such that they are plausible in hardware and capture well foreseen hardware-level faults. Many works consider faults at an abstract behavioral level that do not necessarily map to hardware or their probability of occurrence from a hardware point of view is very small. Most fault injection experiments consider a subset of possible faults or a subset of the sub-blocks of an accelerator. Fault injection experiments on actual hardware or radiation experiments can shed more light on the impact of faults but ideally the impact should be assessed earlier at the design stage so as to add the necessary provisions on-chip for fault tolerance.

B. Testability

As discussed above, a plethora of new testability features and methodologies have been proposed in the literature and adopted in practical AI hardware accelerator designs. However, moving forward, there is still pressing need for novel DFT solutions to target existing and upcoming challenges. We have seen various demands emerging on the horizon for products in the next several years.

First, as part of ordinary ASIC flow, DFT activities are tightly associated with design and physical design in many aspects, such as turn-around cycle, physical design tools, methodologies, and even computing resources. Since many today’s large AI hardware accelerator designs are indeed challenging the design and physical design limitations, posing direct threats to project delivery, successful DFT solutions should recognize and attempt to help mitigate these threats. For example, large AI hardware accelerator designs usually cannot be readily fit into existing computing resources (i.e., servers, emulators etc.), hence design needs to be sliced into multiple modes and multiple partitions. Without a feasible solution, the number of modes and partitions may quickly get out of control and computing resources will soon be depleted. In many cases, DFT verification may demand even more resources than function verification. Existing tools provide basic help, yet a major function and DFT verification framework still has to be hand-crafted to ensure resource availability and design space coverage. On the physical design side, many issues have been addressed in earlier sections and current DFT tools have had a strong focus on solving these problems. However, major challenges continue to bother the DFT owners, such as timing budget, PPA request, signal routability, performance correlation, etc., not only in large AI hardware accelerator designs for cloud utilization, but also in smaller designs used in edge devices, which are extremely sensitive to power and area. As such, DFT architects have to be well-versed with all stages of the design flow and ensure their customized DFT architecture and flow can accommodate the design and physical design requirements to meet the target of PPA and time-to-market.

Second, new AI hardware accelerator architectures may require the advances of novel DFT architectures, algorithms or models. Needless to mention the exotic neuromorphic and in-memory computing hardware, which have spurred research on many new DFT architectures and fault models. Even the traditional style of design may encounter a much higher level of challenges than before. For example, as mentioned earlier, DFT for large AI hardware accelerator designs need to be multi-mode and multi-partition in design and verification spaces. This may incur an excessive number of test patterns, causing ATE memory overflow or unaccepted test expenses. This is not a new problem, but exacerbated in AI hardware accelerators. Since there are no readily adopted solutions, DFT architects need to be creative on designing some on-chip hardware for low-cost test. On the other side, problems rarely seen before may emerge as new norm. For example, some AI hardware accelerator architectures feature unique logic that is not ATPG friendly, e.g., a very deep logic depth at post-synthesis requesting very high coverage. Traditional test point insertion may lead to unacceptable performance or area penalty. To handle such issues, upgrades in ATPG algorithms or DFT architecture may be needed. Moreover, the extensive use of large size SRAMs at advanced tech-nodes may see new types of memory faults, and MBIST algorithms may need to be updated too.

In addition, as already a trend in traditional design, efficient and effective DFT activities need to be both left-shift and right-

shift. Left-shift refers to the DFT involvement in early design stages, e.g., DFT flow starts with design architecting and floor-planning, most DFT implementations are done at RTL stage, etc. Right-shift refers to the DFT activities extending well into or even beyond post-silicon stages such as bring up, diagnosis, volume production and in-field debug, so that the entire product life-cycle quality can be ensured. Since most AI hardware accelerator designs are domain-specific with very tight schedule, left-shift can help shrink design cycle and meet the time-to-market target. Meanwhile, the new AI hardware accelerator architectures or components are usually not fully proved over their service time, thus a right-shift strategy is crucial to fully capture silicon characterization and product behavior over its life cycle. This learning is particularly important for the mission critical products such as automobile and data center, where reliability is of top concern and in-field test may be dictated. From DFT perspective, such designs not only require a complete solution from regular scan, MBIST, I/O test, to on-line test, but also a close correlation between test and function operation to ensure a high quality product. This goal may be as challenging as the design itself.

Finally, as Moore's law slows down, 3D IC has been proposed as a major solution to performance gain, cost reduction and shrink of form factor. While traditional interposer-based 2.5D solutions connect dies horizontally and have been widely adopted, 3D designs stack dies in vertical dimension and currently have focused mostly on external memories such as HBM. Test for 3D IC with memory dies have seen major advances in addressing several challenges. First, faults on Through-Silicon Vias (TSV) and memories have to be tested. Second, these faults need diagnosis solutions for repair and quick yield ramp up. Finally, at-speed self-test is needed to reduce cost and ensure fault coverage. Moving forward, as stacked logic die becomes widely adopted, the test, diagnosis and repair of interconnects between dies and test access of stacked dies seem to be the next challenge.

C. Dependability

From the perspective of AI hardware and system dependability, one fundamental challenge is on specifiability of AI-based functionality. The traditional dependability assurance, e.g., safety assurance required by standards such as ISO 26262 or IEC 61508, is based on the assumption that there exists a full specification of the targeted functionality. These specifications are then used to guide risk analysis, dependability (e.g. safety) management, concept development and validation activities. The full specification assumption holds valid for most traditional rule-based programmed approaches. On the other hand, AI-based functionality may not be fully specifiable. For example, functionality of object recognition in autonomous driving applications can only partially be specified using rules. While lack of full specification is exactly one basic driving factor of employing data-driven AI methods in these application domains, it creates a big challenge to dependability assurance, especially under the existing framework.

AI specifiability challenge is also related to its interpretability challenge, especially for DNNs. While there is a significant research effort on "Explainable AI", many advanced DNN models have hitherto remained non-interpretable. This characteristic of AI-based system becomes an obstacle to directly applying traditional white-box verification and testing methods, which are common in traditional dependability assurance practices. As we have surveyed, most works in AI hardware dependability focus on fault tolerance techniques. While fault tolerance is an important means to mitigate dependability issues, verification and testing are vital components to meet the assurance requirements, especially from standard compliance perspective (e.g. to meet ASIL requirements defined in ISO 26262). Existing functional safety standards, e.g. ISO26262 and ISO21448, do not explicitly address the specific characteristics of the AI system. Lack of both specifiability and interpretability renders the challenges of applying a traditional ISO26262 style approach to AI systems, unless these obstacles are removed, or a new alternative approach is taken.

One outstanding challenge is on how to define efficiently measurable metrics for evaluation of dependability of AI hardware and systems, both on-line and off-line. Most of recent work we have surveyed are based on DNN models, and the metrics used for dependability evaluation are primarily based on prediction accuracy. While this metric is suitable for off-line analysis with ground-truth info available (e.g. fault criticality analysis), it is challenging to use the accuracy metric during on-line dependability management where real-time assessment is needed and often ground truth may not exist. For these types of applications, an alternative metric may be needed to quickly assess the dependability state of AI hardware and system.

Also, currently most research has been focused on supervised learning for DNNs. There are other paradigms of AI/ML, including unsupervised learning and reinforcement learning. Dependability of AI hardware for these paradigms are still under-explored. Moreover, there is an emerging end-to-end DNN approach (e.g., DNN is trained to infer the control directly from sensor data inputs) in many AI application domains. How to define the appropriate metrics for such an approach remains a challenge.

Fault prevention and fault tolerance are current focus areas of AI hardware dependability research field, as witnessed by this survey. Fault prediction may start attracting more attention in some dependability-critical applications, where proactive management is much desired. A growing interest is calling for more research on this direction. For example, applying an AI approach to fault prediction of AI hardware may be a good example of creating a virtuous cycle of "AI for AI".

Finally, from AI hardware perspective, some AI hardware architectures are highly specified for training, while the others target at optimization for inference workload. While most recent dependability study focus on inference AI hardware, the dependability assurance of AI hardware used for training also deserves attention, especially with the edge computing and federated learning paradigms where training is moved from cloud to edge devices.

VIII. Conclusions

In this paper we presented a systematic survey on state-of-the-art research and development of AI hardware testability

and dependability. With the emergence of more hardware innovations to address AI computing challenges, testability and dependability challenges of AI hardware should be addressed to meet both manufacturing quality and in-field service assurance requirements. This paper covers the research of this new field, which has rapidly been evolving especially over the past few years. Although much work has been done, in the future many open challenges remain, including fault criticality assessment with dramatically exploding fault space, hardware-aware fault modeling at the high abstraction level, practical DFT for large-scale designs and 3-D/2.5D advanced packaging, dependability verification and validation with the limited specifiability and interpretability of AI models, efficient metrics for in-field real-time dependability evaluation, etc. To address these existing and upcoming challenges, continuous advances of innovations from both industry and academia are expected over the next years.

ACKNOWLEDGMENTS

The work of H.-G. Stratigopoulos was supported by the ANR RE-TRUSTING project under Grant N^{o} ANR-21-CE24-0015-03.

REFERENCES

[1] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, MIT Press, 2016.

[2] Wm. A. Wulf and S. A. McKee, "Hitting the memory wall: Implications of the obvious," *SIGARCH Comput. Archit. News*, vol. 23, no. 1, pp. 20–24, Mar. 1995.

[3] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proc. IEEE*, vol. 105, no. 12, pp. 2295–2329, Nov. 2017.

[4] A. Reuther, P. Michaleas, M. Jones, V. Gadepally, S. Samsi, and J. Kepner, "Survey and benchmarking of machine learning accelerators," in *Proc. High Perform. Extreme Comput. Conf. (HPEC)*, Sep. 2019.

[5] M. Bouvier, A. Valentian, T. Mesquida, F. Rummens, M. Reyboz, E. Vianello, and E. Beigne, "Spiking neural networks hardware implementations and challenges: A survey," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 15, no. 2, Apr. 2019.

[6] Y. Chen, Y. Xie, L. Song, F. Chen, and T. Tang, "A survey of accelerator architectures for deep neural networks," *Engineering*, vol. 6, no. 3, pp. 264–274, Mar. 2020.

[7] L. Deng, G. Li, S. Han, L. Shi, and Y. Xie, "Model compression and hardware acceleration for neural networks: A comprehensive survey," *Proc. IEEE*, vol. 108, no. 4, pp. 485–532, Apr. 2020.

[8] M. Capra, B. Bussolino, A. Marchisio, M. Shafique, G. Masera, and M. Martina, "An updated survey of efficient hardware architectures for accelerating deep convolutional neural networks," *Future Internet*, vol. 12, no. 7, 2020.

[9] S. Bavikadi *et al.*, "A survey on machine learning accelerators and evolutionary hardware platforms," *IEEE Des. Test*, vol. 39, no. 3, pp. 91–116, Jun. 2022.

[10] M. Shafique *et al.*, "An overview of next-generation architectures for machine learning: Roadmap, opportunities and challenges in the IoT era," in *Proc. Design Autom. Test Europe Conf. (DATE)*, Mar. 2018, pp. 827–832.

[11] M. Costandi, *Neuroplasticity*, The MIT Press, 2016.

[12] S. Dave, A. Marchisio, M. A. Hanif, A. Guesmi, A. Shrivastava, I. Alouani, and M. Shafique, "Special Session: Towards an agile design methodology for efficient, reliable, and secure ML systems," in *Proc. 40th IEEE VLSI Test Symp. (VTS)*, Apr. 2022.

[13] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

[14] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997.

[15] W. Maass, "Networks of spiking neurons: The third generation of neural network models," *Neural Netw.*, vol. 10, no. 9, pp. 1659–1671, Dec. 1997.

[16] C. Mead, *Analog VLSI and Neural Systems*, Addison Wesley, 1989.

[17] G. Indiveri *et al.*, "Neuromorphic silicon neuron circuits," *Front. Neurosci.*, vol. 5, May 2011, Article 73.

[18] M. Pfeiffer and T. Pfeil, "Deep learning with spiking neurons: opportunities and challenges," *Front. Neurosci.*, vol. 12, Oct. 2018, Article 774.

[19] L. A. Camuñas-Mesa, B. Linares-Barranco, and T. Serrano-Gotarredona, "Spiking neural networks and their memristor-CMOS hardware implementations," *Materials*, vol. 12, no. 17, Aug. 2019, Article 2745.

[20] M. Valle, "Analog VLSI implementation of artificial neural networks with supervised on-chip learning," *Analog Integr. Circuits Signal Process.*, vol. 33, pp. 263–287, Dec. 2002.

[21] S.-C. Liu, T. Delbruck, G. Indiveri, A. Whatley, and R. Douglas, *Event-based neuromorphic systems*, John Wiley & Sons, 2014.

[22] B. Chatterjee, P. Panda, S. Maity, A. Biswas, K. Roy, and S. Sen, "Exploiting inherent error resiliency of deep neural networks to achieve extreme energy efficiency through mixed-signal neurons," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 27, no. 6, pp. 1365–1377, Mar. 2019.

[23] A. Rubino, C. Livanelioglu, N. Qiao, M. Payvand, and G. Indiveri, "Ultra-low-power FDSOI neural circuits for extreme-edge neuromorphic intelligence," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 68, no. 1, pp. 45–56, Nov. 2021.

[24] Z. Du, A. Lingamneni, Y. Chen, K. V. Palem, O. Temam, and C. Wu, "Leveraging the error resilience of neural networks for designing highly energy efficient accelerators," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 34, no. 8, pp. 1223–1235, Aug. 2015.

[25] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," in *Proc. Int. Conf. Mach. Learn. (ICML)*, Jul. 2015, p. 1737–1746.

[26] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or -1," *arXiv:1602.02830 [cs.LG]*, Mar. 2016.

[27] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-Net: Imagenet classification using binary convolutional neural networks," *arXiv:1603.05279 [cs.CV]*, Aug. 2016.

[28] T. P. Xiao, C. H. Bennett, B. Feinberg, S. Agarwal, and M. J. Marinella, "Analog architectures for neural network acceleration based on non-volatile memory," *Appl. Phys. Rev.*, vol. 7, no. 3, pp. 031301, Sep. 2020.

[29] A. Ankit, I. Chakraborty, A. Agrawal, M. Ali, and K. Roy, "Circuits and architectures for in-memory computing-based machine learning accelerators," *IEEE Micro*, vol. 40, no. 6, pp. 8–22, Nov./Dec. 2020.

[30] S. Yu, "Neuro-inspired computing with emerging nonvolatile memories," *Proc. IEEE*, vol. 106, no. 2, pp. 260–285, Feb. 2018.

[31] B. Belhadj, A. Valentian, P. Vivet, M. Duranton, L. He, and O. Temam, "The improbable but highly appropriate marriage of 3D stacking and neuromorphic accelerators," in *Proc. Int. Conf. Compil. Archit. Synth. Embed. Syst. (CASES)*, Oct. 2014, pp. 1–9.

[32] D. Kim, J. Kung, S. Chai, S. Yalamanchili, and S. Mukhopadhyay, "Neurocube: A programmable digital neuromorphic architecture with high-density 3D memory," in *Proc. ACM/IEEE 43rd Annual Int. Symp. Comput. Archit. (ISCA)*, Jun. 2016, pp. 380–392.

[33] S. Moradi, N. Qiao, F. Stefanini, and G. Indiveri, "A scalable multicore architecture with heterogeneous memory structures for dynamic neuromorphic asynchronous processors (DYNAPs)," *IEEE Trans. Biomed. Circuits Syst.*, vol. 12, no. 1, pp. 106–122, Nov. 2018.

[34] J. Schemmel, D. Brüderle, A. Grübl, M. Hock, K. Meier, and S. Millner, "A wafer-scale neuromorphic hardware system for large-scale neural modeling," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May/Jun. 2010, pp. 1947–1950.

[35] B. V. Benjamin *et al.*, "Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations," *Proc. IEEE*, vol. 102, no. 5, pp. 699–716, Apr. 2014.

[36] A. Valentian *et al.*, "Fully integrated spiking neural network with analog neurons and RRAM synapses," in *Proc. IEEE Int. Electron Devices Meeting (IEDM)*, Dec. 2019, pp. 14.3.1–14.3.4.

[37] G. K. Chen, R. Kumar, H. E. Sumbul, P. C. Knag, and R. K. Krishnamurthy, "A 4096-neuron 1M-synapse 3.8-pJ/SOP spiking neural network with on-chip STDP learning and sparse weights in 10-nm FinFET CMOS," *IEEE J. Solid-State Circuits*, vol. 54, no. 4, pp. 992–1002, Apr. 2019.

[38] C. Frenkel, M. Lefebvre, J.-D. Legat, and D. Bol, "A 0.086-mm^2 12.7-pJ/SOP 64k-synapse 256-neuron online-learning digital spiking neuromorphic processor in 28-nm CMOS," *IEEE Trans. Biomed. Circuits Syst.*, vol. 13, no. 1, pp. 145–158, Feb. 2019.

[39] S. B. Furber, F. Galluppi, S. Temple, and L. A. Plana, "The SpiNNaker Project," *Proc. IEEE*, vol. 102, no. 5, pp. 652–665, May 2014.

[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, Aug. 2014.

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

[42] H. Mostafa, B. U. Pedroni, S. Sheik, and G. Cauwenberghs, "Fast classification using sparsely active spiking networks," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2017.

[43] L. A. Camuñas-Mesa, Y. L. Domínguez-Cordero, A. Linares-Barranco, T. Serrano-Gotarredona, and B. Linares-Barranco, "A configurable event-driven convolutional node with rate saturation mechanism for modular convnet systems implementation," *Front. Neurosci.*, vol. 12, Feb. 2018, Article 63.

[44] D. Bankman, L. Yang, B. Moons, M. Verhelst, and B. Murmann, "An always-on 3.8 μJ/86% CIFAR-10 mixed-signal binary CNN processor with all memory on chip in 28-nm CMOS," *IEEE J. Solid-State Circuits*, vol. 54, no. 1, pp. 158–172, Jan. 2019.

[45] J. Zhang, Z. Wang, and N. Verma, "In-memory computation of a machine-learning classifier in a standard 6T SRAM array," *IEEE J. Solid-State Circuits*, vol. 52, no. 4, pp. 915–924, Apr. 2017.

[46] C. Eckert *et al.*, "Neural cache: Bit-serial in-cache acceleration of deep neural networks," in *Proc. ACM/IEEE 45th Annual Int. Symp. Comput. Archit. (ISCA)*, Jun. 2018, pp. 383–396.

[47] P. Chi *et al.*, "PRIME: A novel processing-in-memory architecture for neural network computation in reRAM-based main memory," in *Proc. ACM/IEEE 43rd Annual Int. Symp. Comput. Archit. (ISCA)*, Jun. 2016, pp. 27–39.

[48] A. Shafiee *et al.*, "ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," in *Proc. ACM/IEEE 43rd Annual Int. Symp. Comput. Archit. (ISCA)*, Jun. 2016, pp. 14–26.

[49] L. Song, X. Qian, H. Li, and Y. Chen, "PipeLayer: A pipelined ReRAM-based accelerator for deep learning," in *IEEE Proc. Int. Symp. High-Perform. Comput. Archit. (HPCA)*, Feb. 2017, pp. 541–552.

[50] C.-X. Xue *et al.*, "A CMOS-integrated compute-in-memory macro based on resistive random-access memory for AI edge devices," *Nat. Electron.*, vol. 4, pp. 81–90, Jan. 2021.

[51] N. Nassif *et al.*, "Sapphire rapids: The next-generation intel xeon scalable processor," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, Feb. 2022, vol. 65, pp. 44–46.

[52] Y. Chen, T. Chen, Z. Xu, N. Sun, and O. Temam, "DianNao family: Energy-efficient hardware accelerators for machine learning," *Commun. ACM*, vol. 59, no. 11, pp. 105–112, Nov. 2016.

[53] Y. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE J. Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, Jan. 2017.

[54] N. P. Jouppi *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proc. ACM/IEEE Annual Int. Symp. Comput. Archit. (ISCA)*, Jun. 2017.

[55] C. Farabet, C. Poulet, J. Y. Han, and Y. LeCun, "CNP: An FPGA-based processor for convolutional networks," in *Proc. Int. Conf. Field-Program. Log. Appl. (FPL)*, Aug./Sep. 2009.

[56] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre, and K. Vissers, "FINN: A framework for fast, scalable binarized neural network inference," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays (FPGA)*, Feb. 2017, p. 65–74.

[57] G. Li *et al.*, "Understanding error propagation in deep learning neural network (DNN) accelerators and applications," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal. (SC)*, Nov. 2017.

[58] E. M. El Mhamdi and R. Guerraoui, "When neurons fail," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. (IPDPS)*, May/Jun. 2017, pp. 1028–1037.

[59] F. H. Bahnsen, V. Klebe, and G. Fey, "Effect analysis of low-level hardware faults on neural networks using emulated inference," in *Proc. Int. Conf. Mod. Circuits Syst. Technol. (MOCAST)*, Jul. 2021.

[60] D. Maliuk, H.-G. Stratigopoulos, H. Huang, and Y. Makris, "Analog neural network design for RF built-in self-test," in *Proc. IEEE Int. Test Conf. (ITC)*, Nov. 2010, Paper 23.2.

[61] C.H. Sequin and R.D. Clay, "Fault tolerance in artificial neural networks," in *Proc. Int. Jt. Conf. Neural Netw. (IJCNN)*, Jun. 1990, vol. 1, pp. 703–708.

[62] G. Bolt, "Fault models for artificial neural networks," in *Proc. Int. Jt. Conf. Neural Netw. (IJCNN)*, Nov. 1991, vol. 2, pp. 1373–1378.

[63] P. Chandra and Y. Singh, "Fault tolerance of feedforward artificial neural networks - A framework of study," in *Proc. Int. Jt. Conf. Neural Netw. (IJCNN)*, 2003, vol. 1.

[64] D.B.I. Feltham and W. Maly, "Physically realistic fault models for analog CMOS neural networks," *IEEE J. Solid-State Circuits*, vol. 26, no. 9, pp. 1223–1229, Sep. 1991.

[65] A. S. Orgenci, G. Dundar, and S. Balkur, "Fault-tolerant training of neural networks in the presence of MOS transistor mismatches," *IEEE Trans. Circuits Syst. II. Analog Digit. Signal Process.*, vol. 48, no. 3, pp. 272–281, Mar. 2001.

[66] B. Reagen *et al.*, "Ares: A framework for quantifying the resilience of deep neural networks," in *Proc. 55th ACM/ESDA/IEEE Design Autom. Conf. (DAC)*, Jun. 2018.

[67] Y. He, P. Balaprakash, and Y. Li, "FIdelity: Efficient resilience analysis framework for deep learning accelerators," in *Proc. 53rd Annu. IEEE/ACM Int. Symp. Microarchit. (MICRO)*, Oct. 2020, pp. 270–281.

[68] Z. Chen, G. Li, K. Pattabiraman, and N. DeBardeleben, "*BinFI*: An efficient fault injector for safety-critical machine learning systems," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal. (SC)*, Nov. 2019.

[69] L. M. Luza *et al.*, "Emulating the effects of radiation-induced soft-errors for the reliability assessment of neural networks," *IEEE Trans. Emerg. Topics Comput.*, vol. 10, no. 4, pp. 1867–1882, Oct./Dec. 2022.

[70] A. P. Arechiga and A. J. Michaels, "The robustness of modern deep learning architectures against single event upset errors," in *Proc. High Perform. Extreme Comput. Conf. (HPEC)*, Sep. 2018.

[71] M. Sabbagh, C. Gongye, Y. Fei, and Y. Wang, "Evaluating fault resiliency of compressed deep neural networks," in *Proc. IEEE Int. Conf. Embed. Softw. Syst. (ICESS)*, Jun. 2019.

[72] A. Bosio, P. Bernardi, A. Ruospo, and E. Sanchez, "A reliability analysis of a deep neural network," in *Proc. IEEE Lat. Am. Test Symp. (LATS)*, Mar. 2019.

[73] A. Ruospo, A. Bosio, A. Ianne, and E. Sanchez, "Evaluating convolutional neural networks reliability depending on their data representation," in *Proc. 23rd Euromicro Conf. Digit. Syst. Des. (DSD)*, Aug. 2020, pp. 672–679.

[74] M. A. Neggaz, I. Alouani, S. Niar, and F. Kurdahi, "Are CNNs reliable enough for critical applications? An exploratory study," *IEEE Des. Test*, vol. 37, no. 2, pp. 76–83, Apr. 2020.

[75] K. Givaki *et al.*, "On the resilience of deep learning for reduced-voltage FPGAs," in *Proc. Euromicro Int. Conf. Parallel Distrib. Netw. Based Process. (PDP)*, Mar. 2020, pp. 110–117.

[76] Z. Gao *et al.*, "Reliability evaluation of pruned neural networks against errors on parameters," in *Proc. IEEE Int. Symp. Defect Fault Toler. VLSI Nanotechnol. Syst. (DFT)*, Oct. 2020.

[77] Y. Ibrahim *et al.*, "Soft error resilience of deep residual networks for object recognition," *IEEE Access*, vol. 8, pp. 19490–19503, Jan. 2020.

[78] K. Adam, I. I. Mohd, and Y. M. Younis, "The impact of the soft errors in convolutional neural network on GPUs: Alexnet as case study," *Procedia Comput. Sci.*, vol. 182, pp. 89–94, 2021.

[79] E. Malekzadeh, N. Rohbani, Z. Lu, and M. Ebrahimi, "The impact of faults on DNNs: A case study," in *Proc. IEEE Int. Symp. Defect Fault Toler. VLSI Nanotechnol. Syst. (DFT)*, Oct. 2021.

[80] Z. Wan, A. Anwar, Y.-S. Hsiao, T. Jia, V. J. Reddi, and A. Raychowdhury, "Analyzing and improving fault tolerance of learning-based navigation systems," in *Proc. Design Autom. Conf. (DAC)*, Dec. 2021, pp. 841–846.

[81] F. F. dos Santos, P. F. Pimenta, C. Lunardi, L. Draghetti, L. Carro, D. Kaeli, and P. Rech, "Analyzing and increasing the reliability of convolutional neural networks on GPUs," *IEEE Trans. Reliab.*, vol. 68, no. 2, pp. 663–677, Jun. 2019.

[82] F. F. dos Santos, P. Navaux, L. Carro, and P. Rech, "Impact of reduced precision in the reliability of deep neural networks for object detection," in *Proc. IEEE Eur. Test Symp. (ETS)*, May 2019.

[83] A. Lotfi *et al.*, "Resiliency of automotive object detection networks on GPU architectures," in *Proc. IEEE Int. Test Conf. (ITC)*, Nov. 2019.

[84] F. F. Dos Santos *et al.*, "Characterizing a neutron-induced fault model for deep neural networks," *IEEE Trans. Nucl. Sci.*, 2022.

[85] F. Libano, B. Wilson, J. Anderson, M. J. Wirthlin, C. Cazzaniga, C. Frost, and P. Rech, "Selective hardening for neural networks in FPGAs," *IEEE Trans. Nucl. Sci.*, vol. 66, no. 1, pp. 216–222, Jan. 2019.

[86] A. Mahmoud *et al.*, "PyTorchFI: A runtime perturbation tool for DNNs," in *Proc. 50th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw. Workshops (DSN-W)*, Jun./Jul. 2020, pp. 25–31.

[87] Z. Chen, N. Narayanan, B. Fang, G. Li, K. Pattabiraman, and N. DeBardeleben, "TensorFI: A flexible fault injection framework for tensorflow applications," in *Proc. IEEE Int. Symp. Softw. Rel. Eng. (ISSRE)*, Oct. 2020, pp. 426–435.

[88] C. Bolchini, L. Cassano, A. Miele, and A. Toschi, "Fast and accurate error simulation for CNNs against soft errors," *IEEE Trans. Comput.*, 2022.

[89] Y. Zhang, H. Itsuji, T. Uezono, T. Toba, and M. Hashimoto, "Estimating vulnerability of all model parameters in DNN with a small number of fault injections," in *Proc. Design Autom. Test Europe Conf. (DATE)*, Mar. 2022, pp. 60–63.

[90] C. D. Schuman *et al.*, "Resilience and robustness of spiking neural networks for neuromorphic systems," in *Proc. Int. Jt. Conf. Neural Netw. (IJCNN)*, Jul. 2020.

[91] T. Spyrou, S. A. El-Sayed, E. Afacan, L. A. Camuñas-Mesa, B. Linares-Barranco, and H.-G. Stratigopoulos, "Neuron fault tolerance in spiking neural networks," in *Proc. Design Autom. Test Europe Conf. (DATE)*, Feb. 2021.

[92] B. Salami, O. S. Unsal, and A. C. Kestelman, "On the resilience of RTL NN accelerators: Fault characterization and mitigation," in *Proc. Int. Symp. Comput. Archit. High Perform. Comput. (SBAC-PAD)*, Sep. 2018, pp. 322–329.

[93] A. Ruospo, A. Balaara, A. Bosio, and E. Sanchez, "A pipelined multi-level fault injector for deep neural networks," in *Proc. IEEE Int. Symp. Defect Fault Toler. VLSI Nanotechnol. Syst. (DFT)*, Oct. 2020.

[94] J. E. Rodriguez Condia, F. F. dos Santos, M. Sonza Reorda, and P. Rech, "Combining architectural simulation and software fault injection for a fast and accurate CNNs reliability evaluation on GPUs," in *Proc. IEEE VLSI Test Symp. (VTS)*, Apr. 2021.

[95] J. J. Zhang, K. Basu, and S. Garg, "Fault-tolerant systolic array based accelerators for deep neural network execution," *IEEE Des. Test*, vol. 36, no. 5, pp. 44–53, Oct. 2019.

[96] J. Deng *et al.*, "Retraining-based timing error mitigation for hardware neural networks," in *Proc. Design Autom. Test Europe Conf. (DATE)*, Mar. 2015, pp. 593–596.

[97] A. Chaudhuri, J. Talukdar, J. Jung, G. Nam, and K. Chakrabarty, "Fault-criticality assessment for AI accelerators using graph convolutional networks," in *Proc. Design Autom. Test Europe Conf. (DATE)*, Feb. 2021, pp. 1596–1599.

[98] A. Chaudhuri, J. Talukdar, F. Su, and K. Chakrabarty, "Functional criticality analysis of structural faults in AI accelerators," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 41, no. 12, pp. 5657–5670, Dec. 2022.

[99] O. Temam, "A defect-tolerant accelerator for emerging high-performance applications," in *Proc. Annual Int. Symp. Comput. Archit. (ISCA)*, Jun. 2012, pp. 356–367.

[100] S. A. El-Sayed, T. Spyrou, E. Afacan, L. A. Camuñas-Mesa, B. Linares-Barranco, and H.-G. Stratigopoulos, "Spiking neuron hardware-level fault modeling," in *Proc. 26th IEEE Int. Symp. On-Line Test. Robust Syst. Des. (IOLTS)*, Jul. 2020.

[101] G. Gambardella, J. Kappauf, M. Blott, C. Doehring, M. Kumm, P. Zipf, and K. Vissers, "Efficient error-tolerant quantized neural network accelerators," in *Proc. IEEE Int. Symp. Defect Fault Toler. VLSI Nanotechnol. Syst. (DFT)*, OCt. 2019.

[102] C. De Sio, S. Azimi, and L. Sterpone, "An emulation platform for evaluating the reliability of deep neural networks," in *Proc. IEEE Int. Symp. Defect Fault Toler. VLSI Nanotechnol. Syst. (DFT)*, Oct. 2020.

[103] N. Khoshavi, C. Broyles, and Y. Bi, "Compression or corruption? a study on the effects of transient faults on BNN inference accelerators," in *Proc. IEEE Int. Symp. Qual. Electron. Design (ISQED)*, Mar. 2020, pp. 99–104.

[104] N. Khoshavi, A. Roohi, C. Broyles, S. Sargolzaei, Y. Bi, and D. Z. Pan, "SHIELDeNN: Online accelerated framework for fault-tolerant deep neural network architectures," in *Proc. Design Autom. Conf. (DAC)*, Jul 2020.

[105] I. Souvatzoglou, A. Papadimitriou, A. Sari, V. Vlagkoulis, and M. Psarakis, "Analyzing the single event upset vulnerability of binarized neural networks on SRAM FPGAs," in *Proc. IEEE Int. Symp. Defect Fault Toler. VLSI Nanotechnol. Syst. (DFT)*, Oct. 2021.

[106] D. Xu *et al.*, "Reliability evaluation and analysis of FPGA-based neural network acceleration system," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 29, no. 3, pp. 472–484, Mar. 2021.

[107] P. Corneliou, P. Nikolaou, M. K. Michael, and T. Theocharides, "Fine-grained vulnerability analysis of resource constrained neural inference accelerators," in *Proc. IEEE Int. Symp. Defect Fault Toler. VLSI Nanotechnol. Syst. (DFT)*, Oct. 2021.

[108] T. Spyrou, S. A. El-Sayed, E. Afacan, L. A. Camuñas-Mesa, B. Linares-Barranco, and H.-G. Stratigopoulos, "Reliability analysis of a spiking neural network hardware accelerator," in *Proc. Design Autom. Test Europe Conf. (DATE)*, Mar. 2022.

[109] G. Abich, J. Gava, R. Garibotti, R. Reis, and L. Ost, "Applying lightweight soft error mitigation techniques to embedded mixed precision deep neural networks," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 68, no. 11, pp. 4772–4782, Nov. 2021.

[110] G. Abich, R. Garibotti, R. Reis, and L. Ost, "The impact of soft errors in memory units of edge devices executing convolutional neural networks," *IEEE Trans. Circuits Syst. II: Express Br.*, vol. 69, no. 3, pp. 679–683, Mar. 2022.

[111] L. M. Luza *et al.*, "Investigating the impact of radiation-induced soft errors on the reliability of approximate computing systems," in *Proc. IEEE Int. Symp. Defect Fault Toler. VLSI Nanotechnol. Syst. (DFT)*, Oct. 2020.

[112] F. Libano, P. Rech, B. Neuman, J. Leavitt, M. Wirthlin, and J. Brunhaver, "How reduced data precision and degree of parallelism impact the reliability of convolutional neural networks on FPGAs," *IEEE Trans. Nucl. Sci.*, vol. 68, no. 5, pp. 865–872, May 2021.

[113] G. Gambardella, N. J. Fraser, U. Zahid, G. Furano, and M. Blott, "Accelerated radiation test on quantized neural networks trained with fault aware training," in *Proc. IEEE Aerosp. Conf. (AERO)*, Mar. 2022.

[114] R. L. Rech Junior *et al.*, "High energy and thermal neutron sensitivity of google tensor processing units," *IEEE Trans. Nucl. Sci.*, vol. 69, no. 3, pp. 567–575, Mar. 2022.

[115] C. Liu, M. Hu, J. P. Strachan, and H. Li, "Rescuing memristor-based neuromorphic design with high defects," in *Proc. 54th ACM/EDAC/IEEE Design Autom. Conf.*, Jun. 2017.

[116] E. Vatajelu, G. Di Natale, and L. Anghel, "Special session: Reliability of hardware-implemented spiking neural networks (SNN)," in *Proc. IEEE VLSI Test Symp. (VTS)*, Apr. 2019.

[117] Z. Ye, R. Liu, J. L. Taggart, H. J. Barnaby, and S. Yu, "Evaluation of radiation effects in RRAM-based neuromorphic computing system for inference," *IEEE Trans. Nucl. Sci.*, vol. 66, no. 1, pp. 97–103, Jan. 2019.

[118] C.-Y. Chen and K. Chakrabarty, "Efficient identification of critical faults in memristor crossbars for deep neural networks," in *Proc. Design Autom. Test Europe Conf. (DATE)*, Feb. 2021.

[119] F. Chollet *et al.*, "Keras," https://keras.io, 2015.

[120] J. Bergstra *et al.*, "Theano: A CPU and GPU math compiler in python," Jan. 2010.

[121] M. Abadi *et al.*, "TensorFlow: A system for large-scale machine learning," in *Proc. 12th USENIX Symp. Oper. Syst. Des. Implement. (OSDI)*, Nov. 2016, p. 265–283.

[122] S. K. S. Hari, T. Tsai, M. Stephenson, S. W. Keckler, and J. Emer, "SASSIFI: An architecture-level fault injection tool for GPU application resilience evaluation," in *IEEE Int. Symp. Perform. Anal. Syst. Softw. (ISPASS)*, 2017, pp. 249–258.

[123] T. Tsai, S. K. S. Hari, M. Sullivan, O. Villa, and S. W. Keckler, "NVBitFI: Dynamic fault injection for GPUs," in *Proc. 51st Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw. (DSN)*, Jun. 2021, pp. 284–291.

[124] D. Oliveira *et al.*, "Experimental and analytical study of Xeon Phi reliability," in *Proc. Int. Conf. High Perform. Comput. Netw. Storage Anal. (SC)*, Nov. 2017.

[125] S. B. Shrestha and G. Orchard, "SLAYER: Spike layer error reassignment in time," in *Proc. Adv. Neural Inf. Process. Syst. (NeurIPS)*, 2018, pp. 1412–1421.

[126] A. Paszke *et al.*, "PyTorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., pp. 8024–8035. Curran Associates, Inc., 2019.

[127] S. Kundu, S. Banerjee, A. Raha, S. Natarajan, and K. Basu, "Toward functional safety of systolic array-based deep learning hardware accelerators," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 29, no. 3, pp. 485–498, Jan. 2021.

[128] S. Knowles, "Designing the colossus Mk2 IPU," *Hot Chips*, 2021.

[129] T. Norrie *et al.*, "The design process for Google's training chips: TPUv2 and TPUv3," *IEEE Micro*, vol. 41, no. 2, pp. 56–63, Mar./Apr. 2021.

[130] G. Lauterbach, "The path to successful wafer-scale integration: The cerebras story," *IEEE Micro*, vol. 41, no. 6, pp. 52–57, Nov./Dec. 2021.

[131] G. Giles, J. Wang, A. Sehgal, K. J. Balakrishnan, and J. Wingfield, "Test access mechanism for multiple identical cores," in *Proc. IEEE Int. Test Conf. (ITC)*, Oct. 2008.

[132] K. Chakravadhanula, V. Chickermane, D. Pearl, A. Garg, R. Khurana, S. Mukherjee, and P. Nagaraj, "SmartScan - hierarchical test compression for pin-limited low power designs," in *Proc. IEEE Int. Test Conf. (ITC)*, Sep. 2013.

[133] M. Sharma, A. Dutta, W.-T. Cheng, B. Benware, and M. Kassab, "A novel test access mechanism for failure diagnosis of multiple isolated identical cores," in *Proc. IEEE Int. Test Conf. (ITC)*, Sep. 2011.

[134] A. Chaudhuri, C. Liu, X. Fan, and K. Chakrabarty, "C-testing and efficient fault localization for AI accelerators," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 41, no. 7, pp. 2348–2361, Jul. 2022.

[135] S. Motaman, S. Ghosh, and J. Park, "A perspective on test methodologies for supervised machine learning accelerators," *IEEE Trans. Emerg. Sel. Topics Power Electron.*, vol. 9, no. 3, pp. 562–569, Aug. 2019.

[136] Y. Huang and R. Singhai, "Tutorial 1B: AI chip technologies and DFT methodologies," in *Proc. IEEE Int. Syst.-on-Chip Conf. (SOCC)*, Sep. 2019.

[137] H. Jia *et al.*, "A programmable neural-network inference accelerator based on scalable in-memory computing," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, Feb. 2021, pp. 236–237.

[138] D. Niu *et al.*, "184QPS/W 64Mb/mm^2 3D logic-to-DRAM hybrid bonding with process-near-memory engine for recommendation system," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, Feb. 2022.

[139] J. Cote *et al.*, "Streaming scan network (SSN): An efficient packetized data network for testing of complex SoCs," in *Proc. IEEE Int. Test Conf. (ITC)*, Nov. 2020.

[140] G. Boschi, E. Spano, H. Grigoryan, A. Kumar, and G. Harutyunyan, "Die-to-die testing and ECC error mitigation in automotive and industrial safety applications," in *Proc. IEEE Int. Test Conf. (ITC)*, Nov. 2020.

[141] M. Hutner, G. Tshagharyan, and G. Harutyunyan, "Testing HBM2 in at-speed mode," in *Innovative Practices Session on In-System Test and Reliability of Memories, Proc. IEEE VLSI Test Symp. (VTS)*, Apr. 2019.

[142] G. Harutyunyan and Y. Zorian, "An effective embedded test & diagnosis solution for external memories," in *Proc. IEEE Int. Symp. On-Line Test. Robust Syst. Des. (IOLTS)*, Jul. 2015, pp. 168–170.

[143] J. Mekkoth, S. Bandyopadhyay, and A. Kumar, "Efficient infrastructure for external memory DRAM test," in *Innovative Practices Session on In-System Test and Reliability of Memories, Proc. IEEE VLSI Test Symp. (VTS)*, Apr. 2019.

[144] "IEEE standard for test access architecture for three-dimensional stacked integrated circuits," *IEEE Std 1838-2019*, pp. 1–73, 2020.

[145] A. Gebregiorgis and M. B. Tahoori, "Testing of neuromorphic circuits: Structural vs functional," in *Proc. IEEE Int. Test Conf. (ITC)*, Nov. 2019, Paper 3.2.

[146] W. Li, Y. Wang, H. Li, and X. Li, "RRAMedy: Protecting ReRAM-based neural network from permanent and soft faults during its lifetime," in *Proc. IEEE Int. Conf. Comput. Des. (ICCD)*, Nov. 2019, pp. 91–99.

[147] C.-Y. Chen and K. Chakrabarty, "On-line functional testing of memristor-mapped deep neural networks using backdoored checksums," in *Proc. IEEE Int. Test Conf. (ITC)*, Oct. 2021, pp. 83–92.

[148] O. Aramoon and G. Qu, "Provably accurate memory fault detection method for deep neural networks," in *Proc. Great Lakes Symp. VLSI (GLSVLSI)*, Jun. 2021, pp. 443–448.

[149] S. T. Ahmed and M. B. Tahoori, "Campact functional test generation for memristive deep learning implementations using approximate gradient ranking," in *Proc. IEEE Int. Test Conf. (ITC)*, Sep. 2022, pp. 239–248.

[150] H.-Y. Tseng, I-W. Chiu, M.-T. Wu, and J. C.-M. Li, "Machine learning-based test pattern generation for neuromorphic chips," in *IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Nov. 2021.

[151] S. A. El-Sayed, T. Spyrou, L. A. Camuñas-Mesa, and H.-G. Stratigopoulos, "Compact functional testing for neuromorphic computing circuits," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, 2022.

[152] B. Luo, Y. Li, L. Wei, and Q. Xu, "On functional test generation for deep neural network IPs," in *Proc. Design Autom. Test Europe Conf. (DATE)*, Mar. 2019, pp. 1010–1015.

[153] K. Pei, Y. Cao, J. Yang, and S. Jana, "DeepXplore: Automated whitebox testing of deep learning systems," in *Proc. 26th ACM Symp. Oper. Syst. Princ. (SOSP)*, Oct. 2017.

[154] Y. Tian, K. Pei, S. Jana, and B. Ray, "DeepTest: Automated testing of deep-neural-network-driven autonomous cars," in *Proc. IEEE/ACM Int. Conf. Softw. Eng. (ICSE)*, May/Jun. 2018, p. 303–314.

[155] K. Ma, A. Saha, C. Amarnath, and A. Chatterjee, "Efficient low cost alternative testing of analog crossbar arrays for deep neural networks," in *Proc. IEEE Int. Test Conf. (ITC)*, Sep. 2022, pp. 499–503.

[156] S. A. El-Sayed, L. A. Camuñas-Mesa, B. Linares-Barranco, and H.-G. Stratigopoulos, "Self-testing analog spiking neuron circuit," in *Proc. Int. Conf. Synth. Model. Anal. Simulat. Methods Appl. Circuit Design (SMACD)*, Jul. 2019.

[157] A. Ruospo, D. Piumatti, A. Floridia, and E. Sanchez, "A suitability analysis of software based testing strategies for the on-line testing of artificial neural networks applications in embedded devices," in *Proc. IEEE Int. Symp. On-Line Test. Robust Syst. Des. (IOLTS)*, 2021.

[158] Y. He, T. Uezono, and Y. Li, "Efficient functional in-field self-test for deep learning accelerators," in *Proc. IEEE Int. Test Conf. (ITC)*, Oct. 2021, pp. 93–102.

[159] T. Uezono, Y. He, and Y. Li, "Achieving automotive safety requirements through functional in-field self-test for deep learning accelerators," in *Proc. IEEE Int. Test Conf. (ITC)*, Sep. 2022, pp. 465–473.

[160] M. Liu and K. Chakrabarty, "Online fault detection in ReRAM-based computing systems by monitoring dynamic power consumption," in *Proc. IEEE Int. Test Conf. (ITC)*, Nov. 2020.

[161] E. Ozen and A. Orailoglu, "Low-cost error detection in deep neural network accelerators with linear algorithmic checksums," *J. Electron. Test.: Theory Appl.*, vol. 36, no. 6, pp. 703–718, Dec. 2020.

[162] E. Ozen and A. Orailoglu, "Concurrent monitoring of operational health in neural networks through balanced output partitions," in *Proc. IEEE Asia South–Pac. Design Autom. Conf. (ASP-DAC)*, Jan. 2020, pp. 169–174.

[163] S. Hari, M. Sullivan, T. Tsai, and S. Keckler, "Making convolutions resilient via algorithm-based error detection techniques," *IEEE Trans. Dependable Secure Comput.*, vol. 19, no. 4, pp. 2546–2558, Jul./Aug. 2022.

[164] B. F. Goldstein *et al.*, "A lightweight error-resiliency mechanism for deep neural networks," in *Proc. IEEE Int. Symp. Qual. Electron. Design (ISQED)*, Apr. 2021, pp. 311–316.

[165] N. I. Deligiannis, R. Cantoro, M. Sonza Reorda, M. Traiola, and E. Valea, "Towards the integration of reliability and security mechanisms to enhance the fault resilience of neural networks," *IEEE Access*, vol. 9, pp. 155998–156012, Nov. 2021.

[166] M. Sadi and U. Guin, "Test and yield loss reduction of AI and deep learning accelerators," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 32, no. 7, pp. 1124–1135, Jan. 2021.

[167] L. Xia *et al.*, "Stuck-at fault tolerance in RRAM computing systems," *IEEE J. Emerg. Sel. Top. Circuits Syst.*, vol. 8, no. 1, pp. 102–115, Mar. 2018.

[168] Z. Song, Y. Sun, L. Chen, T. Li, N. Jing, X. Liang, and L. Jiang, "ITT-RNA: Imperfection tolerable training for RRAM-crossbar-based deep neural-network accelerator," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 40, no. 1, pp. 129–142, Apr. 2021.

[169] L.-H. Hoang, M. A. Hanif, and M. Shafique, "TRe-map: Towards reducing the overheads of fault-aware retraining of deep neural networks by merging fault maps," in *Proc. 24th Euromicro Conf. Digit. Syst. Des. (DSD)*, Sep. 2021, pp. 434–441.

[170] S. Kannan, J. Rajendran, R. Karri, and O. Sinanoglu, "Sneak-path testing of crossbar-based nonvolatile random access memories," *IEEE Trans Nanotechnol.*, vol. 12, no. 3, pp. 413–426, May 2013.

[171] C.-Y. Chen *et al.*, "RRAM defect modeling and failure analysis based on march test and a novel squeeze-search scheme," *IEEE Trans. Comput.*, vol. 64, no. 1, pp. 180–190, Jan. 2015.

[172] E. I. Vatajelu, P. Prinetto, M. Taouil, and S. Hamdioui, "Challenges and solutions in emerging memory testing," *IEEE Trans. Emerg. Topics Comput.*, vol. 7, no. 3, pp. 493–506, Jul./Sep. 2019.

[173] L. Wu *et al.*, "Defect and fault modeling framework for STT-MRAM testing," *IEEE Trans. Emerg. Topics Comput.*, vol. 9, no. 2, pp. 707–723, Apr./Jun. 2021.

[174] P. Liu, Z. You, J. Wu, B. Liu, Y. Han, and K. Chakrabarty, "Fault modeling and efficient testing of memristor-based memory," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 68, no. 11, pp. 4444–4455, Nov. 2021.

[175] P. Girard, Y. Cheng, A. Virazel, W. Zhao, R. Bishnoi, and M. B. Tahoori, "A survey of test and reliability solutions for magnetic random access memories," *Proc. IEEE*, vol. 109, no. 2, pp. 149–169, Feb. 2021.

[176] L. Wu, S. Rao, M. Taouil, E. J. Marinissen, G. S. Kar, and S. Hamdioui, "Characterization, modeling, and test of intermediate state defects in

STT-MRAMs," *IEEE Trans. Comput.*, vol. 71, no. 9, pp. 2219–2233, Sep. 2022.

[177] M. Fieback *et al.*, "Defects, fault modeling, and test development framework for RRAMs," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 18, no. 3, Apr. 2022.

[178] T. Han, I. Choi, and S. Kang, "Majority-based test access mechanism for parallel testing of multiple identical cores," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 23, no. 8, pp. 1439–1447, Aug. 2014.

[179] A. Ramdas and O. Sinanoglu, "Testing chips with spare identical cores," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 32, no. 7, pp. 1124–1135, Jun. 2013.

[180] P. N. Variyam, S. Cherubal, and A. Chatterjee, "Prediction of analog performance parameters using fast transient testing," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 21, no. 3, pp. 349–361, Mar. 2002.

[181] H.-G. Stratigopoulos and S. Mir, "Adaptive alternate analog test," *IEEE Des. Test Comput.*, vol. 29, no. 4, pp. 71–79, Jun. 2012.

[182] A. Pandey, B. Tully, A. Samudra, A. Nagarandal, K. Natarajan, and R. Singhal, "Novel technique for manufacturing & in-system testing of large scale SoC using functional protocol based high-speed I/O," in *Proc. IEEE VLSI Test Symp. (VTS)*, Apr. 2022.

[183] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Trans. Dependable Secure Comput.*, vol. 1, no. 1, pp. 11–33, Oct. 2004.

[184] G. Buja and R. Menis, "Dependability and functional safety: Applications in industrial electronics systems," *IEEE Ind. Electron. Mag.*, vol. 6, no. 3, pp. 4–12, Sep. 2012.

[185] "ISO 26262: Road vehicles-functional safety," 2018.

[186] "IEC 61508: Edition 2.0 functional safety," 2010.

[187] "RTCA/DO-178C: Software considerations in airborne systems and equipment certification," 2012.

[188] "ISO PAS 21448: Road vehicles – safety of the intended functionality," 2019.

[189] J.L. Hennessy and D.A. Patterson, "A new golden age for computer architecture," *Commun. ACM*, vol. 62, no. 2, pp. 48–60, Feb. 2019.

[190] J. Goodfellow, I, J.n Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *arXiv:1412.6572v3 [stat.ML]*, Mar. 2015.

[191] N. Akhtar and A. Mian, "Threat of adversarial attacks on deep learning in computer vision: A survey," *IEEE Access*, vol. 6, pp. 14410–14430, Feb. 2018.

[192] T. Graepel, K. Lauter, and M. Naehrig, "ML confidential: Machine learning on encrypted data," in *Proc. Int. Conf. Inf. Secur. Cryptol. (ICISC)*, Dec. 2012, p. 1–21.

[193] B. Wang and N. Z. Gong, "Stealing hyperparameters in machine learning," in *Proc. IEEE Symp. Secur. Priv. (SP)*, May 2018, pp. 36–52.

[194] W. Hua, Z. Zhang, and G. E. Suh, "Reverse engineering convolutional neural networks through side-channel information leaks," in *Proc. 55th ACM/ESDA/IEEE Design Autom. Conf. (DAC)*, Jun. 2018.

[195] T. Gu, K. Liu, B. Dolan-Gavitt, and S. Garg, "BadNets: Evaluating backdooring attacks on deep neural networks," *IEEE Access*, vol. 7, pp. 47230–47244, Apr. 2019.

[196] H. Chen, C. Fu, J. Zhao, and F. Koushanfar, "DeepInspect: A black-box trojan detection and mitigation framework for deep neural networks," in *Proc. 28th Int. Jt. Conf. Artif. Intell. (IJCAI)*, Jul. 2019, pp. 4658–4664.

[197] Y. Liu, L. Wei, B. Luo, and Q. Xu, "Fault injection attack on deep neural network," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Nov. 2017, pp. 131–138.

[198] A. S. Rakin, Z. He, and D. Fan, "Bit-flip attack: Crushing neural network with progressive bit search," in *2019 IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct./Nov. 2019, pp. 1211–1220.

[199] E. Cheng *et al.*, "(Invited) Cross-layer resilience: Challenges, insights, and the road ahead," in *Proc. 56th ACM/IEEE Design Autom. Conf. (DAC)*, Jun. 2019.

[200] S. Mittal and J.S. Vetter, "A survey of CPU-GPU heterogeneous computing techniques," *ACM Comput. Surv.*, vol. 47, no. 4, pp. 1–35, Jul. 2015.

[201] European Commission, "Ethics guidelines for trustworthy artificial intelligence (AI)," https://digital-strategy.ec.europa.eu/en/library/ethics-guidelines-trustworthy-ai, Apr. 2019, Online.

[202] C.-T. Chin, K. Mehrotra, C.K. Mohan, and S. Rankat, "Training techniques to obtain fault-tolerant neural networks," in *Proc. IEEE Int. Symp. Fault-Toler. Comput. (FTCS)*, Jun. 1994, pp. 360–369.

[203] B. S. Arad and A. El-Amawy, "On fault tolerant training of feedforward neural networks," *Neural Netw.*, vol. 10, no. 3, pp. 539–553, 1997.

[204] N. Wei, S. Yang, and S. Tong, "A modified learning algorithm for improving the fault tolerance of BP networks," in *Proc. Int. Conf. Neural Netw. (ICNN)*, Jun. 1996, vol. 1, pp. 247–252.

[205] P.J. Edwards and A.F. Murray, "Penalty terms for fault tolerance," in *Proc. Int. Conf. Neural Netw. (ICNN)*, Jun. 1997, vol. 2, pp. 943–947.

[206] S. Cavalieri and O. Mirabella, "A novel learning algorithm which improves the partial fault tolerance of multilayer neural networks," *Neural Netw.*, vol. 12, no. 1, pp. 91–106, Jan. 1999.

[207] C. Neti, M.H. Schneider, and E.D. Young, "Maximally fault tolerant neural networks," *IEEE Trans. Neural Netw.*, vol. 3, no. 1, pp. 14–23, Jan. 1992.

[208] D. Deodhare, M. Vidyasagar, and S. Sathiya Keethi, "Synthesis of fault-tolerant feedforward neural networks using minimax optimization," *IEEE Transactions on Neural Networks*, vol. 9, no. 5, pp. 891–900, 1998.

[209] Z.-H. Zhou and S.-F. Chen, "Evolving fault-tolerant neural networks," *Neural. Comput. Appl.*, vol. 11, pp. 156–160, Jun. 2003.

[210] E. Sugawara, M. Fukushi, and S. Horiguchi, "Fault tolerant multi-layer neural networks with GA training," in *Proc. IEEE Int. Symp. Defect Fault Toler. VLSI Syst. (DFT)*, Nov. 2003, pp. 328–335.

[211] N.C. Hammadi, T. Ohmameuda, K. Kaneko, and H. Ito, "Fault tolerant constructive algorithm for feedforward neural networks," in *Proc. IEEE Pacific Rim Int. Symp. Fault-Toler. Syst. (PRFTS)*, Dec. 1997, pp. 215–220.

[212] C. Torres-Huitzil and B. Girau, "Fault and error tolerance in neural networks: A review," *IEEE Access*, vol. 5, pp. 17322 – 17341, Aug. 2017.

[213] R. A. Solovyev, A. L. Stempkovsky, and D. V. Telpukhov, "Study of fault tolerance methods for hardware implementations of convolutional neural networks," *Opt. Mem. Neural Networks*, vol. 28, no. 2, pp. 82–88, Apr. 2019.

[214] G. B. Hacene, F. Leduc-Primeau, A. B. Soussia, V. Gripon, and F. Gagnon, "Training modern deep neural networks for memory-fault robustness," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2019.

[215] U. Zahid, G. Gambardella, N. J. Fraser, M. Blott, and K. Vissers, "FAT: Training neural networks for reliable inference under hardware faults," in *Proc. IEEE Int. Test Conf. (ITC)*, Nov. 2020.

[216] L. Yang and B. Murmann, "SRAM voltage scaling for energy-efficient convolutional neural networks," in *Proc. IEEE Int. Symp. Qual. Electron. Design (ISQED)*, Mar. 2017, pp. 7–12.

[217] R. V. W. Putra, M. A. Hanif, and M. Shafique, "SparkXD: A framework for resilient and energy-efficient spiking neural network inference using approximate DRAM," in *Proc. 58th Design Autom. Conf. (DAC)*, Dec. 2021, p. 379–384.

[218] Z. He, J. Lin, R. Ewetz, J.-S. Yuan, and D. Fan, "Noise injection adaption: End-to-end ReRAM crossbar non-ideal effect adaption for neural network mapping," in *Proc. 56th ACM/IEEE Design Autom. Conf. (DAC)*, Jun. 2019.

[219] Y. Long, X. She, and S. Mukhopadhyay, "Design of reliable DNN accelerator with un-reliable ReRAM," in *Proc. Design Autom. Test Europe Conf. (DATE)*, Mar. 2019, pp. 1769–1774.

[220] V. Joshi *et al.*, "Accurate deep neural network inference using computational phase-change memory," *Nat. Commun.*, vol. 11, no. 1, pp. 1–13, May 2020.

[221] Y. Zhu *et al.*, "Statistical training for neuromorphic computing using memristor-based crossbars considering process variations and noise," in *Proc. Design Autom. Test Europe Conf. (DATE)*, Mar. 2020, pp. 1590–1593.

[222] D. Gao, G.L. Zhang, X. Yin, B. Li, U. Schlichtmann, and C. Zhuo, "Reliable memristor-based neuromorphic design using variation-and defect-aware training," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Nov. 2021.

[223] E. Ozen and A. Orailoglu, "SNR: Squeezing numerical range defuses bit error vulnerability surface in deep neural networks," *ACM Trans. Embed. Comput. Syst.*, vol. 20, no. 5s, Sep. 2021.

[224] M. Bocquet, T. Hirtzlin, J.-O. Klein, E. Nowak, E. Vianello, J.-M. Portal, and D. Querlioz, "Embracing the unreliability of memory devices for neuromorphic computing," in *Proc. IEEE Int. Reliab. Phys. Symp. (IRPS)*, Apr./May 2020.

[225] A. Azizimazreah, Y. Gu, X. Gu, and L. Chen, "Tolerating soft errors in deep learning accelerators with reliable on-chip memory designs," in *Proc. IEEE Int. Conf. Netw. Archit. Storage (NAS)*, Oct. 2018.

[226] C. Khunasaraphan, K. Vanapipat, and C. Lursinsap, "Weight shifting techniques for self-recovery neural networks," *IEEE Trans. Neural Netw.*, vol. 5, no. 4, pp. 651–658, Jul. 1994.

[227] E. Ozen and A. Orailoglu, "Just say zero: containing critical bit-error propagation in deep neural networks with anomalous feature suppression," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Nov. 2020.

[228] E. Ozen and A. Orailoglu, "Boosting bit-error resilience of DNN accelerators through median feature selection," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 11, pp. 3250–3262, Nov. 2020.

[229] M.D. Emmerson and R.I. Damper, "Determining and improving the fault tolerance of multilayer perceptrons in a pattern-recognition application," *IEEE Trans. Neural Netw.*, vol. 4, no. 5, pp. 788–793, 1993.

[230] C.-T. Chiu, K. Mehrotra, C.K. Mohan, and S. Ranka, "Robustness of feedforward neural networks," in *Proc. IEEE Int. Conf. Neural Netw. (ICNN)*, Mar./Apr. 1993, vol. 2, pp. 783–788.

[231] T. Liu, W. Wen, L. Jiang, Y. Wang, C. Yang, and G. Quan, "A fault-tolerant neural network architecture," in *Proc. 56th ACM/IEEE Design Autom. Conf. (DAC)*, Jun. 2019.

[232] M. A. Hanif and M. Shafique, "DNN-life: An energy-efficient aging mitigation framework for improving the lifetime of on-chip weight memories in deep neural network hardware architectures," in *Proc. Design Autom. Test Europe Conf. (DATE)*, Feb. 2021.

[233] A. Hashmi, H. Berry, O. Temam, and M. Lipasti, "Automatic abstraction and fault tolerance in cortical microachitectures," in *Proc. ACM/IEEE Annual Int. Symp. Comput. Archit. (ISCA)*, Jun. 2011, pp. 1–10.

[234] W. Li, X. Ning, G. Ge, X. Chen, Y. Wang, and H. Yang, "FTT-NAS: Discovering fault-tolerant neural architecture," in *Proc. 25th Asia South–Pac. Design Autom. Conf. (ASP-DAC)*, Jan. 2020, pp. 211–216.

[235] L.-H. Hoang, M. A. Hanif, and M. Shafique, "FT-ClipAct: Resilience analysis of deep neural networks and improving their fault tolerance using clipped activation," in *Proc. Design Autom. Test Europe Conf. (DATE)*, Mar. 2020, p. 1241–1246.

[236] Z. Chen, G. Li, and K. Pattabiraman, "A low-cost fault corrector for deep neural networks through range restriction," in *Proc. 51st Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw. (DSN)*, Jun. 2021.

[237] B. Ghavami, M. Sadati, Z. Fang, and L. Shannon, "FitAct: Error resilient deep neural networks via fine-grained post-trainable activation functions," in *Proc. Design Autom. Test Europe Conf. (DATE)*, Mar. 2022, pp. 1239–1244.

[238] S. Burel, A. Evans, and L. Anghel, "Improving DNN fault tolerance in semantic segmentation applications," in *Proc. IEEE Int. Symp. Defect Fault Toler. VLSI Nanotechnol. Syst. (DFT)*, Oct. 2022.

[239] Z. Xu and J. Abraham, "Safety design of a convolutional neural network accelerator with error localization and correction," in *Proc. IEEE Int. Test Conf. (ITC)*, Nov. 2019, Paper 12.3.

[240] M. Liu, L. Xia, Y. Wang, and K. Chakrabarty, "Algorithmic fault detection for RRAM-based matrix operations," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 25, no. 3, pp. 29:1–29:31, May 2020.

[241] K. Zhao *et al.*, "FT-CNN: Algorithm-based fault tolerance for convolutional neural networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 7, pp. 1677–1689, Jul. 2021.

[242] J. Kosaian and K. V. Rashmi, "Arithmetic-intensity-guided fault tolerance for neural network inference on GPUs," in *Proc. Int. Conf. High Perform. Comput. Netw. Storage Anal. (SC)*, Nov. 2021.

[243] D. Filippas, N. Margomenos, N. Mitianoudis, C. Nicopoulos, and G. Dimitrakopoulos, "Low-cost online convolution checksum checker," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 30, no. 2, pp. 201–212, Feb. 2022.

[244] C. S. Mummidi, S. Bal, B. F. Goldstein, S. Srinivasan, and S. Kundu, "A highly-efficient error detection technique for general matrix multiplication using tiled processing on SIMD architecture," in *Proc. IEEE Int. Conf. Comput. Des. (ICCD)*, Oct. 2022, pp. 529–536.

[245] C. Schorn, A. Guntoro, and G. Ascheid, "Accurate neuron resilience prediction for a flexible reliability management in neural network accelerators," in *Proc. Design Autom. Test Europe Conf. (DATE)*, Mar. 2018, pp. 979–984.

[246] M. A. Hanif and M. Shafique, "SalvageDNN: salvaging deep neural network accelerators with permanent faults through saliency-driven fault-aware mapping," *Phil. Trans. R. Soc. A*, vol. 378, no. 2164, Feb. 2020.

[247] A. Ruospo and E. Sanchez, "On the reliability assessment of artificial neural networks running on AI-oriented MPSoCs," *Appl. Sci.*, vol. 11, no. 14, Jul. 2021.

[248] R. V. W. Putra, M. A. Hanif, and M. Shafique, "ReSpawn: Energy-efficient fault-tolerance for spiking neural networks considering unreliable memories," in *Proc. ACM/IEEE Int. Conf. Comput.-Aided Design (ICCAD)*, Nov. 2021.

[249] S.-K. Lu, Y.-S. Wu, J.-H. Hong, and K. Miyase, "Fault resilience techniques for flash memory of DNN accelerators," in *Proc. IEEE Int. Test Conf. Asia (ITC-Asia)*, Aug. 2022.

[250] E. Talpes *et al.*, "Compute solution for Tesla's full self-driving computer," *IEEE Micro*, vol. 40, no. 2, pp. 25–35, Mar./Apr. 2020.

[251] K. Matsubara *et al.*, "4.2 a 12nm autonomous-driving processor with 60.4TOPS, 13.8TOPS/W CNN executed by task-separated ASIL D control," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, Feb. 2021, vol. 64, pp. 56–58.

[252] M. Abdulrahman *et al.*, "HarDNN: Feature map vulnerability evaluation in CNNs," *CoRR*, vol. abs/2002.09786, Dec. 2020.

[253] H. R. Mahdiani, S. M. Fakhraie, and C. Lucas, "Relaxed fault-tolerant hardware implementation of neural networks in the presence of multiple transient errors," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 23, no. 8, pp. 1215–1228, Aug. 2012.

[254] A. Ruospo, G. Gavarini, I. Bragaglia, M. Traiola, A. Bosio, and E. Sanchez, "Selective hardening of critical neurons in deep neural networks," in *Proc. IEEE Int. Symp. Des. Diagn. Electron. Circuits Syst. (DDECS)*, Apr. 2022, pp. 136–141.

[255] C. Liu, C. Chu, D. Xu, Y. Wang, Q. Wang, H. Li, X. Li, and K.T. Cheng, "HyCA: A hybrid computing architecture for fault tolerant deep learning," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 41, no. 10, pp. 3400–3413, Oct. 2022.

[256] Z. Gao *et al.*, "Soft error tolerant convolutional neural networks on fpgas with ensemble learning," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 30, no. 3, pp. 291–302, Mar. 2022.

[257] B. Reagen *et al.*, "Minerva: Enabling low-power, highly-accurate deep neural network accelerators," in *Proc. ACM/IEEE Annual Int. Symp. Comput. Archit. (ISCA)*, Jun. 2016, pp. 267–278.

[258] W. Li *et al.*, "Soft error mitigation for deep convolution neural network on FPGA accelerators," in *Proc. IEEE Int. Conf. Artif. Intell. Circuits Syst. (AICAS)*, 2020.

[259] S. Burel, A. Evans, and L. Anghel, "MOZART+: Masking outputs with zeros for improved architectural robustness and testing of DNN accelerators," *IEEE Trans. Device Mater. Reliab.*, vol. 22, no. 2, pp. 120–128, Jun. 2022.

[260] S. Burel, A. Evans, and L. Anghel, "Zero-overhead protection for cnn weights," in *Proc. IEEE Int. Symp. Defect Fault Toler. VLSI Nanotechnol. Syst. (DFT)*, Oct. 2021.

[261] R. V. W. Putra, M. A. Hanif, and M. Shafique, "SoftSNN: Low-cost fault tolerance for spiking neural network accelerators under soft errors," in *Proc. 59th Design Autom. Conf. (DAC)*, Jul. 2022, p. 151–156.

[262] A. Agrawal, C. Lee, and K. Roy, "X-CHANGR: changing memristive crossbar mapping for mitigating line-resistance induced accuracy degradation in deep neural networks," *CoRR*, vol. abs/1907.00285, Jun. 2019.

[263] T. Titirsha, S. Song, A. Das, J. Krichmar, N. D. Dutt, N. Kandasamy, and F. Catthoor, "Endurance-aware mapping of spiking neural networks to neuromorphic hardware," *IEEE Trans. Parallel Distrib. Syst.*, Mar. 2021.

[264] S.-S. Lee and J.-S. Yang, "Value-aware parity insertion ECC for fault-tolerant deep neural network," in *Proc. Design Autom. Test Europe Conf. (DATE)*, Mar. 2022, pp. 724–729.

[265] C. Schorn, A. Guntoro, and G. Ascheid, "Efficient on-line error detection and mitigation for deep neural network accelerators," in *Proc. Int. Conf. Comput. Safety Rel. Secur. (SAFECOMP)*, Sep. 2018, pp. 205–219.

[266] S. Kim, P. Howe, T. Moreau, A. Alaghi, L. Ceze, and V. Sathe, "MATIC: Learning around errors for efficient low-voltage neural network accelerators," in *Proc. Design Autom. Test Europe Conf. (DATE)*, Mar. 2018.

[267] P. N. Whatmough, S. K. Lee, H. Lee, S. Rama, D. Brooks, and G.-Y. Wei, "A 28nm SoC with a 1.2GHz 568nJ/prediction sparse deep-neural-network engine with >0.1 timing error rate tolerance for IoT applications," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, Feb. 2017, pp. 242–243.

[268] J. Zhang, K. Rangineni, Z. Ghodsi, and S. Garg, "ThUnderVolt: Enabling aggressive voltage underscaling and timing error resilience for energy efficient deep learning accelerators," in *Proc. Design Autom. Conf. (DAC)*, Jun. 2018.

[269] A. P. Johnson *et al.*, "Homeostatic fault tolerance in spiking neural networks: A dynamic hardware perspective," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 65, no. 2, pp. 687–699, 2018.

[270] L. Xia, M. Liu, X. Ning, K. Chakrabarty, and Y. Wang, "Fault-tolerant training enabled by on-line fault detection for RRAM-based neural computing systems," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 38, no. 9, pp. 1611–1624, Sep. 2019.
[271] S. Zhang, G. L. Zhang, B. Li, H. H. Li, and U. Schlichtmann, "Aging-aware lifetime enhancement for memristor-based neuromorphic computing," in *Proc. Design Autom. Test Europe Conf. (DATE)*, Mar. 2019.
[272] P. I. Vaz, P. Girard, A. Virazel, and H. Aziza, "Improving TID radiation robustness of a CMOS OxRAM-based neuron circuit by using enclosed layout transistors," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 29, no. 6, pp. 1122–1131, 2021.
[273] M. V. Beigi and G. Memik, "Thermal-aware optimizations of ReRAM-based neuromorphic computing systems," in *Proc. Design Autom. Conf. (DAC)*, Jun. 2018.
[274] H. Shin, M. Kang, and L.-S. Kim, "A thermal-aware optimization framework for ReRAM-based deep neural network acceleration," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Nov. 2020.
[275] J. Meng *et al.*, "Temperature-resilient RRAM-based in-memory computing for DNN inference," *IEEE Micro*, vol. 42, no. 1, pp. 89–98, Jan./Feb. 2022.
[276] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, Jun. 2014.
[277] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, Apr. 2017.
[278] D. Ernst *et al.*, "Razor: circuit-level correction of timing errors for low-power operation," *IEEE Micro*, vol. 24, no. 6, pp. 10–20, Nov./Dec. 2004.
[279] K.-H. Huang and J. A. Abraham, "Algorithm-based fault tolerance for matrix operations," *IEEE Trans. Comput.*, vol. C-33, no. 6, pp. 518–528, Jun. 1984.

Fei Su is a DFX and Telemetry architect at Intel Corporation. His research interests include testability and dependability of semiconductor circuits/chiplets, AI/ML hardware, cyber-physical systems, and edge/cloud computing. Su has a PhD from Duke University, USA. He is a Senior Member of IEEE.

Chunsheng Liu is the leader of DFT team at Alibaba inc. His research interests include test infrastructure for high-performance processors, FPGA and machine learning accelerators, as well as high dependability of cloud computing hardware. He has a PhD from Duke University, USA. He is a Senior Member of IEEE.

Haralampos-G. Stratigopoulos is a Research Director of the French National Center for Scientific Research (CNRS) at the LIP6 Laboratory of Sorbonne Université, Paris, France. His research interests include neuromorphic computing, hardware security, and design-for-test of integrated circuits and systems. Stratigopoulos has a PhD from Yale University, New Haven, USA. He is a Member of IEEE.