Testing and Reliability of Spiking Neural Networks: A Review of the State-of-the-Art

Haralampos-G. Stratigopoulos, Theofilos Spyrou, Spyridon Raptis

Testing and Reliability of Spiking Neural Networks: A Review of the State-of-the-Art

Haralampos-G. Stratigopoulos, Theofilos Spyrou, and Spyridon Raptis
Sorbonne Université, CNRS, LIP6, Paris, France

***Abstract*—Neuromorphic computing based on Spiking Neural Networks (SNNs) is an emerging computing paradigm inspired by the functionality of the biological brain. Given its potential to revolutionize the power efficiency of many Artificial Intelligence (AI) applications, heavy research is underway on algorithms, hardware implementations, and applications. This article focuses on testing and reliability of hardware implementations providing a review of the state-of-the-art.**

***Index Terms*—Neuromorphic computing, spiking neural networks, testing, reliability, dependability.**

I. Introduction

Spiking Neural Networks (SNNs) are the basis of neuromorphic computing. They enable a type of Artificial Intelligence (AI) that closely mimics the "language" of the biological brain [1], [2].

Compared to the more traditional Artificial Neural Networks (ANNs), SNNs incorporate the notion of time, as illustrated in Fig. 1. Input information is coded into spike trains that propagate through the layers of neurons via their synapse connections. A spike is a discrete event in time that emulates the firing of neurons in the brain. For example, in a computer vision application, the image is decomposed into pixels, and the pixel intensity or brightness is translated to a spike train, as illustrated in Fig. 1(b). Coding methods include rate coding, where the spike frequency is made proportional to the intensity, and time coding, where one spike per pixel is used and the spike is emitted at a time that is inversely proportional to the intensity. Another approach is to interface the SNN with a Dynamic Vision Sensor (DVS) to generate the input information in real time, as illustrated in Fig. 1(a). In this case, the pixels are sensitive to the scene dynamics and produce a spike in response to the temporal intensity changes.

Given two neurons connected with a synapse, the spike generated by the pre-synaptic neuron is weighted by the synapse's strength and reaches the post-synaptic neuron, as illustrated in Fig. 1(c). The most common and hardware-friendly spiking neuron model is the Integrate & Fire (I&F), illustrated also in Fig. 1(c). According to this model, the neuron behaves as a leaky capacitor. A spike is integrated by adding charge to the capacitor and increasing its potential. If the potential exceeds a threshold, then the neuron fires a spike of its own and the capacitor is reset to its resting potential so that the neuron can fire again. There are two additional functionalities that are often implemented, namely leakage, i.e., between two consecutive spikes the potential decays towards the reset state, and refractory period, i.e., the firing rate is suppressed by not allowing the neuron to spike unless a specific time has elapsed.

To make a decision, the spikes produced by the output neurons are translated to real values, i.e., by considering the spike firing rate, the time to first spike, or the interval between spikes. For example, for SNNs that use spike firing rate for classification, the output layer has one neuron per class and the winning class is the one whose neuron produces the larger number of spikes within a time window.

Compared to ANNs, the potential of SNNs is that they can offer low-power operation and faster inference, properties that are particularly attractive for edge computing. In particular, the spike sparsity at the SNN input makes that most neurons remain idle not consuming any power and, in addition, weight multiplications in ANNs are replaced with additions, which are less computationally intensive. Regarding speed, spikes propagate asynchronously through the network, while in ANNs the computation of one layer needs to complete before proceeding to the next layer. In fact, the first output spikes may already point to the correct decision.

However, training SNNs remains a challenge since spikes are discrete events and, thereby, back-propagation training used in ANNs that relies on gradient computation based on differential calculus is non applicable. SNNs also have more hyper-parameters to tune, i.e., neuron's threshold, leakage, and refractory period. Training approaches include ANN to SNN conversion, where an ANN model is trained and the weights are applied onto the equivalent SNN model, Spike-Timing-Dependent Plasticity (STDP), and adapting the back-propagation for SNNs [3].

Nowadays, there is an intense activity on designing efficient hardware accelerators for SNNs. We can group the existing designs into three main categories: (a) Large-scale designs for research purposes, such as SpiNNaker [4] and Loihi [5]; (b) Application-Specific Integrated Circuits (ASICs); and (c) FPGA implementations. For recent reviews, the reader is referred to [6], [7].

Moving forward, we foresee in the near future the high-volume manufacturing of Integrated Circuits (ICs) and systems comprising SNNs. To this end, in parallel to SNN hardware accelerator design efforts, we need to address their testability and reliability aspects [8]. Testing aims at proving that the intent functionality is met. It targets manufacturing faults, but for mission- and safety-critical applications it is applied also on-line in the field to detect silicon aging and radiation-induced soft errors. Reliability assessment aims at identifying the

This work was supported by the ANR RE-TRUSTING project under Grant N^{o} ANR-21-CE24-0015-03.

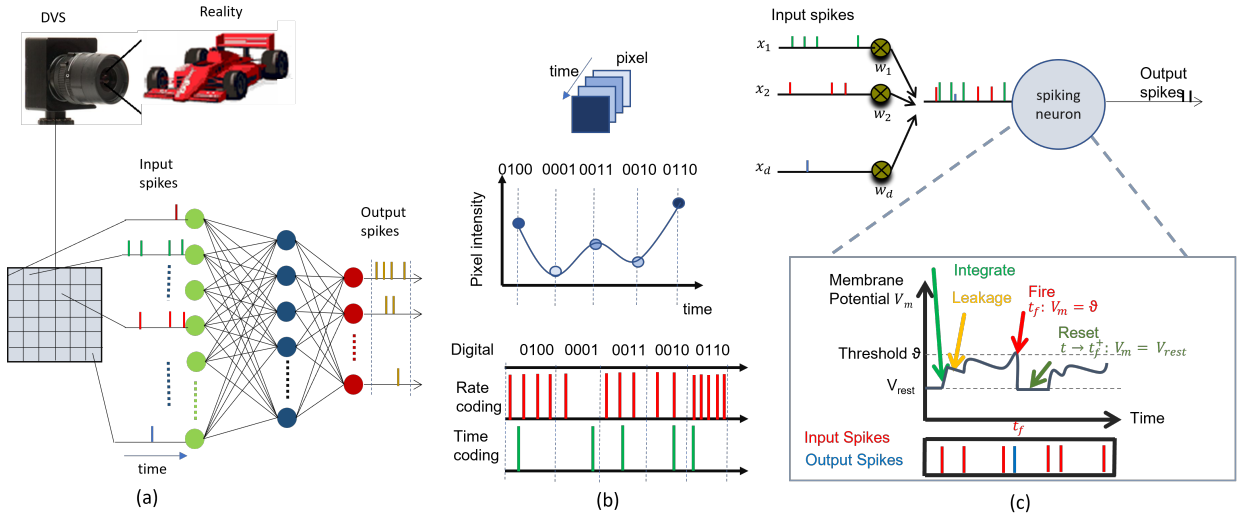

Fig. 1: Principle of operation of SNNs.

critical faults and vulnerable parts of the hardware architecture. This information is valuable for mitigating reliability concerns early in the development process through cost-effective fault tolerance strategies.

There is a belief that hardware implementations of neural networks inherit the remarkable fault tolerance of the biological brain. This is true to some extent as neural networks are highly parallel and distributed systems and thanks also to their overprovisioning, i.e., there is a large number of extra neurons in the network that are not used by the application. As a result, most hardware-level faults are benign. They have no effect locally or their effect gets masked as the information propagates through the network. Some faults, however, will be critical creating an error at the output. The most threatening type of error is the Silent Data Corruption (SDC) that leads to misleading computation but goes unnoticed. Instead, an application crush is observable and the system can enter a safe mode. While high fault rates can be overcome by performing training with the hardware in the loop, the prevalent scenario is that the training is performed in software and then the model is uploaded onto the hardware. Thus, if a critical neuron or synapse in the model is mapped onto a neuron or synapse on the hardware that suffers a fault, this can have a detrimental impact on the application.

This article provides an analysis of the state-of-the-art in reliability, testing, and fault tolerance of SNN hardware accelerators, and concludes by pointing to perspectives for future work.

II. Reliability Assessment

For neural networks, in general, a reliability assessment experiment relies on performing fault injections and assessing the fault impact on the inference accuracy. Fault injection can be performed at the software or hardware level.

A. Fault injection experiments in software

The main challenge is that fault simulation is very time-consuming at transistor level and even in higher-level hardware descriptions, i.e., gate-level, microarchitetural or RTL level. Compounded to this is that the inference accuracy needs to be evaluated on a large validation dataset. To this end, a behavioral-level fault model is often adopted, and fault simulations are performed at the software level [9]–[21]. This is suitable for neural networks as their software and hardware implementation match closely in terms of component connectivity and data flow.

A neural network is often viewed as a distributed system where neurons and synapses can fail independently. The literature in testing and reliability of SNNs [9]–[33] assumes different fault models with the most common being:

- Neuron faults: (a) variations in the timing of output spike trains that can be implementing in various ways, i.e., modifying the neuron's spike integration constant, threshold, leakage, or refractory period; (b) stuck-at responses, including neuron saturation, where firing is non-stop even in the absence of input activity, and dead neurons that never spike.
- Synapse faults: (a) weight perturbation; (b) stuck-at weight; (c) spike transmission delay.
- Spike routing faults.
- Bit-flips in network parameters, i.e., synapse weights, neuron parameters, etc., that are stored as digital words in on-chip memories, registers, and buffers.

It is important to consider behavioral-level faults that are feasible from a hardware perspective and capture hardware-level fault mechanisms. To develop such a fault model, one approach is to simulate faults at circuit level for single neurons or small-size networks [30], [34]. In [34], a neuron is simulated at transistor level after injecting in the netlist defects, i.e., short- and open-circuits, and, in addition, a Monte Carlo simulation is performed to study the effect of process variations. The aforementioned neuron faults were observed, in addition to some others, for example ghost spikes that are generated without the neuron's threshold being exceeded, that were considered to be design-specific. In [30], this bottom-up approach to extract faulty behaviors is performed for

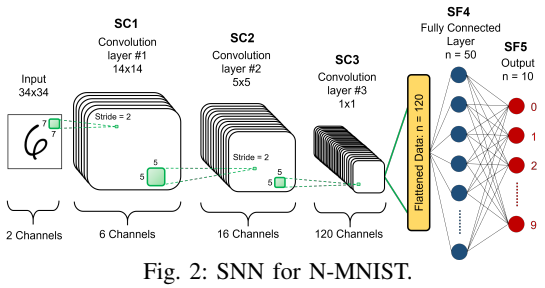

Fig. 2: SNN for N-MNIST.

a memristor-based SNN architecture. Memristors implement synapses and are organized in crossbars implementing the synaptic connections between two neuron layers [35]. This is a form of in-memory computing addressing the main memory wall challenge in AI hardware accelerator design. In [30], fault simulation experiments show that all defects, i.e., open- and short-circuits, transistor stuck-on and stuck-off, drift in memristor resistance values, lead to a variation in the time required by a neuron to finish its I&F operation. To this end, it is proposed to use functional fault models at neuron level, namely slow and fast spike integration.

However, by performing the analysis at the software level, inevitably, some parts of the hardware architecture and scheduling of network operations will be ignored. Furthermore, the reliability assessment becomes AI model-specific. Notice also that even at the software level fault injection experiments may be too time-consuming and may be exhaustive only for small-size networks, thus a fault sampling approach is often needed to focus the effort on faults that are likely to be critical.

Herein, we show results and statistics that are generated using the fault injection framework in [12]. It is built on top of the SLAYER [3] and PyTorch [36] frameworks. It supports a large library of fault models, as well as single or multiple fault injections. It accelerates fault simulation on GPU and by using "tricks" such as early stopping, i.e., simulation is stopped if the fault effect is masked in an intermediate layer, or skipping the computation of layers, i.e., for a fault in a given layer the simulation starts at the prior layer considering its golden fault-free response. Inference time is the same for the nominal and faulty instances of the network.

The case study is a convolutional SNN, shown in Fig. 2, for the classification of the N-MNIST dataset, which is a neuromorphic, i.e., spiking, version of the MNIST dataset that comprises images of handwritten arithmetic digits in gray-scale format [37].

Fig. 3 shows the inference accuracy (y-axis) in the case of dead and saturated neuron faults in different layers (x-axis). Each column may be divided into blocks of different color which points to a specific accuracy value according to the color map at the bottom of Fig. 3. The projection of a block onto the y-axis is the percentage of neurons that if faulty they result in an accuracy according to the color of the block.

Results corroborated on several SNN models [12] lead to the following observations: (a) Neuron saturation has far stronger impact compared to a dead neuron. This is because a saturated

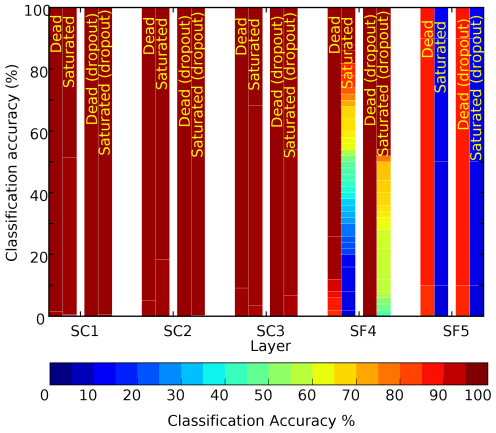

Fig. 3: Neuron fault simulation.

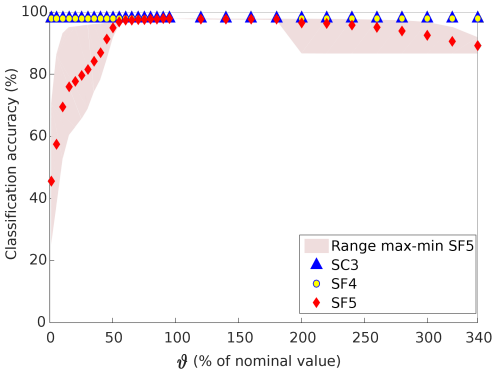

Fig. 4: Neuron threshold variation.

neuron generates an excess of spikes that propagate through the network "polluting" it and eventually altering the spike trains at its output based on which decisions are made. A dead neuron has a lesser effect because of the sparse spiking activity per neuron or because the neuron never fires anyways; (b) The last layer is the most critical one. For example, let us consider SNNs that use spike firing rate for classification. If a neuron saturates, then the same class corresponding to this neuron is always predicted and the accuracy drops to $100/N$, where N is the number of classes. If a neuron becomes dead, then the corresponding class is never predicted and the maximum accuracy that can be attained is $(N-1)\cdot 100/N$.

Fig. 4 shows the effect of neuron threshold variations. Small thresholds trigger spiking at lower values of the membrane potential and the neuron may spike more than usual. In the extreme, the neuron could end up as a saturated neuron. On the other hand, a high threshold requires higher values of the membrane potential for spiking and in the extreme the neuron could end up as a dead neuron. For this case study, only the last layer is affected by neuron threshold variations with the accuracy dropping drastically as the threshold starts reducing below 50% of the nominal value.

The number of synapses largely outnumbers the number of neurons. Thus, the synapse fault simulation time quickly explodes. Therefore, it is worth investigating whether the synapse fault space can be sampled without missing any critical synapses. To this end, in [17], an "extreme" fault

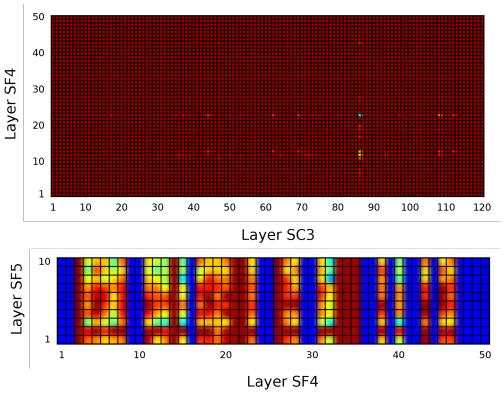

Fig. 5: Positive saturation of synapse weights.

scenario is considered where the synapse weight is saturated to a maximum value, thus likely causing the post-synaptic neuron to fire more than usual. Fault simulation results are shown in Fig. 5, where each box corresponds to a synapse connecting two neurons in two consecutive layers with each axis corresponding to one layer and the number on the axis corresponding to the neuron number in this layer. The color of the box points to the inference accuracy according to the color map at the bottom of Fig. 3. Results reveal that a large number of synapses connecting the last two layers are critical, a very small number of synapses connecting the last two hidden layers are critical, while any other synapse fault in previous layers has no impact. This observation was corroborated on other SNN models too, thus, given that the synapse fault model is "extreme", a possible fault sampling strategy is to simulate synapse faults only in the last layers. The experiment continued considering the more realistic from a hardware perspective synapse weight perturbation. It was assumed that weights are stored on on-chip memory as 8-bit integers. During fault simulation in software, real-valued weights were quantized into 8 bits, random bit-flips were performed, then the weights were reconverted to real values. The fault simulation results are shown in Fig. 6. We observe that only the first Most Significant Bit (MSB) positions are critical, which allows us to further reduce the synapse fault space.

As a final result, Fig. 7 shows the training accuracy in the presence of different fault rates. Even at a fault rate as high as 100 faults, the network is capable of learning around the faults, circumventing the faulty neurons and synapses, and achieving the baseline accuracy.

B. Fault injection experiments in hardware

In [28], an FPGA-based SNN hardware accelerator [38] is subject to fault injection. The case study is an SNN model used to classify card symbols. The hardware accelerator uses a feature map as the foundational building block. A node in a layer can represent a feature map in the case of convolutional layers, or a neuron in the case of fully-connected layers. Equivalently, a neuron could be conceptualized as a 1×1 feature map. Written in VHDL, the feature map is appropriately dimensioned and configured to represent a node of the

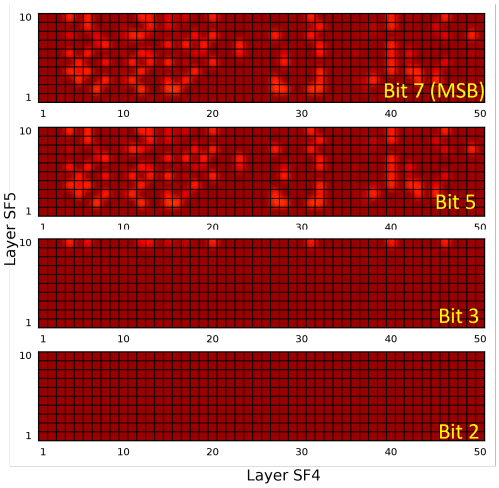

Fig. 6: Bit-flips in synapse weights.

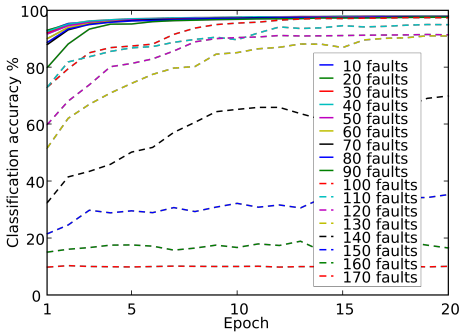

Fig. 7: Training in the presence of faults.

model. Subsequently, the nodes are interconnected in a two-dimensional mesh to construct the SNN model. The accelerator employs the Address Event Representation (AER) protocol to avoid the high number of neuron-to-neuron synapses. According to AER, a spike is an event encoded as a digital address of the neuron that generated it. The address is sent via a shared data bus to a receiver which decodes the address to identify the destination neuron where the spike should be routed. Then the spike is reconstructed at the destination neuron's input. The accelerator has local memory per node that stores the various parameters of the node as digital words, i.e., feature map size, neurons' threshold, leakage, and refractory period, synapse weights, routing parameters for the AER protocol such as the self-addresses of nodes and neurons, etc. Every parameter is represented with 8 bits.

The fault model used is bit-flips in every parameter across different bit positions, as well as random multiple bit-flips with varying Bit Error Rate (BER) probability. Fig. 8 shows example results. Figs. 8(a)-(b) show single bit-flips at given layers, while Fig. 8(c) shows random multiple bit-flips for different BER values across the entire memory. Fault injection experiments are repeated, i.e., in Fig. 8(a) a different synapse in the layer is sampled, in Fig. 8(b) a different node in the network is sampled, and in Fig. 8(c) any bit in the memory is let to flip with probability equal to the BER value. Statistics are visualized using box plots. The bottom and top edges of the box indicate the 25^{th} and 75^{th} percentile, respectively,

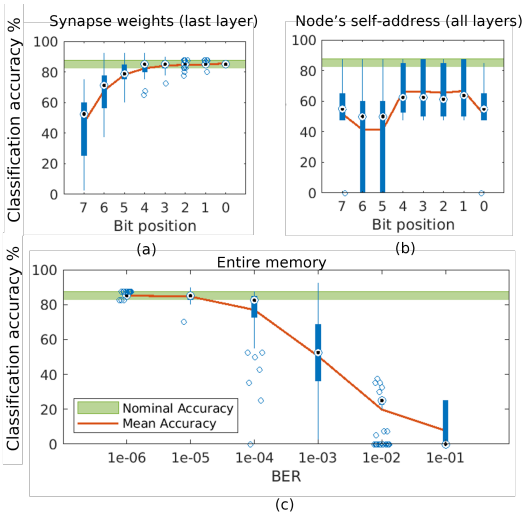

Fig. 8: Reliability analysis of a hardware accelerator.

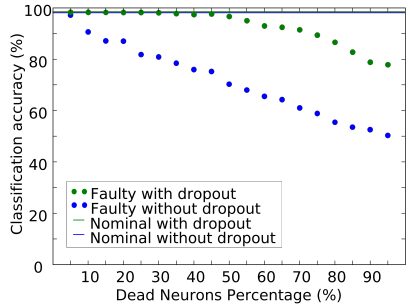

Fig. 9: Tolerated percentage of dead neurons using dropout.

the whiskers extend to the most extreme data points not considered outliers, and the outliers are plotted individually using the 'o' symbol and are not always aligned vertically for illustration purpose. Fig. 8 also shows the tolerated inference accuracy with green color and reports the median shown with a dotted circle and the average across repetitions of the same experiment shown with a red line.

Fig. 8(a) indicates that only the first 4 MSBs of synapse weights are critical. As synapse weights occupy over 60% of the memory, this means that we can save significant test and fault tolerance costs by considering only the 4 MSBs. Fig. 8(b) shows bit-flips in one of the router parameters, namely the node's self-address. All bits of routing parameters are very critical as any bit-flip corrupts the AER protocol. Finally, Fig. 8(c) shows that the network can tolerate a BER up to 10^{-5}.

III. Model-based fault tolerance

A first step towards fault tolerance is to train the model in a way that certain fault effects are suppressed by construction. This can be achieved by fault-aware training where during the training epochs faults are injected such that the model learns in the presence of faults.

In [12], it is shown that training with dropout [39] can nullify the effect of all neuron faults in all hidden layers except neuron saturation faults. Dropout removes neurons during training with some probability, along with their incoming and outgoing connections. 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. Dropout can be seen as a "natural" fault-aware training as neuron and synapses become dead with some probability during the course of training. Dropout equalizes the importance of neurons across the network, resulting in more uniform and sparse activity across the network. The third column per layer in Fig. 3 shows the inference accuracy in the presence of dead neurons after training with dropout. We observe that for layer SF4 the effect of dead neurons is suppressed. In fact, the SNN can withstand a multiple-fault scenario with high dead neuron rates, which can be up to 50% as shown in Fig. 9.

In [14], [15], [20], network parameters, i.e., synapse weights, are quantized according to their data precision on hardware, and bit-flips are induced during the course of training. This fault-aware training makes the network adapt its accuracy to different bit-flip probabilities.

In [11], it is shown that the fault tolerance characteristics of SNNs can greatly depend on the training algorithm, model, and workload. It is proposed to modify the loss function towards error resilience. In particular, for every network instance visited during training, the baseline accuracy is computed as normal and, in addition, fault injections are performed to compute the accuracy in the presence of faults. The loss function becomes a weighted sum of the baseline accuracy and the average accuracy on faulty versions of the network.

IV. Testing

AI hardware accelerators have several architectural particularities, e.g., they consist of an array of small Processing Elements (PEs) (i.e., neurons), they are memory hungry, they use in-memory computing, and there are innovative designs, i.e., based on memristor crossbar arrays [35]. These particularities make standard IC test approaches costly or require the development of novel fault models and conforming test strategies [8]. Another challenge is that functional test is very time-consuming as the tested specification is the inference accuracy on a large dataset [16]. Besides, in this way, the hardware accelerator is tested for one AI model that is likely not using all the chip resources, i.e., neurons and synapses.

A. Functional testing

One direction is to derive a compact set of inputs that can distinguish functional from faulty devices. For example, in a computer vision application, the problem boils down to generating a set of high-coverage images, as shown in Fig. 10. In [13], a test generation algorithm is proposed that starts by injecting a fault and examining if any of the available images (i.e., the training and validation sets) can detect it. If not, adversarial examples are generated aiming at finding one that detects the fault. Adversarial example generation adds a minimum amount of noise to an available image such that it is incorrectly predicted. If any image or adversarial example is

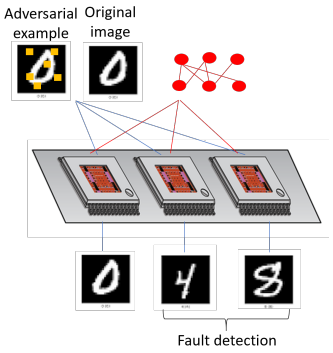

Fig. 10: Functional testing based on a compact set of images.

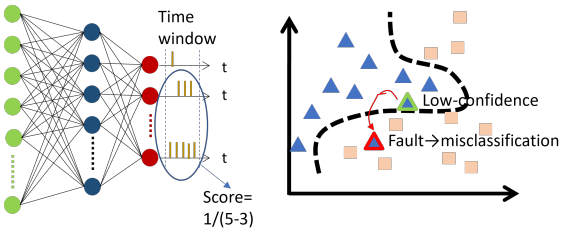

Fig. 11: Test image generation with no fault simulation.

found that detects the fault, then this successful test is tried out on all faults. It is placed in the kept list and the detected faults are dropped from the list. The algorithm reiterates targeting the next undetected fault.

In [17], test generation time is reduced by ordering the available images based on their fault coverage without performing any fault simulation. It is shown that images which are predicted correctly with low confidence have higher fault coverage. Intuitively this happens because these images are located very close to the classification boundary, thus a fault is likely to force their footprint in the feature space to jump over the boundary onto the area of another class, as shown in Fig. 11. For example, for SNNs that use spike frequency for classification, the fault coverage score is inversely proportional to the spike count difference of the top two classes. This idea was demonstrated on hardware using the accelerator in [38]. As shown in Fig. 12, with 5 images we can detect all multiple bit-flip scenarios leading to errors, while by adding one extra image in the test set we can detect all single critical bit-flips as well.

The test images can also be saved on an on-chip memory and repeated periodically or in idle times during the application to implement an on-line test.

In [40], a functional test is proposed for biologically-inspired spiking neurons, illustrated in Fig. 13. 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 deemed faulty.

In [30], a functional test is proposed considering as fault detection criterion the time a neuron requires to finish its I&F

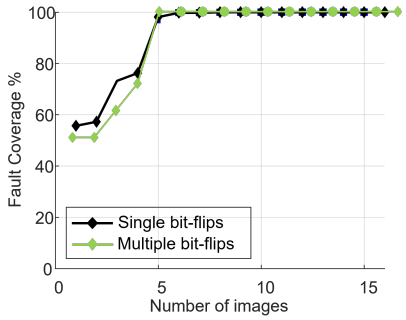

Fig. 12: Testing the accelerator in [38] with test images.

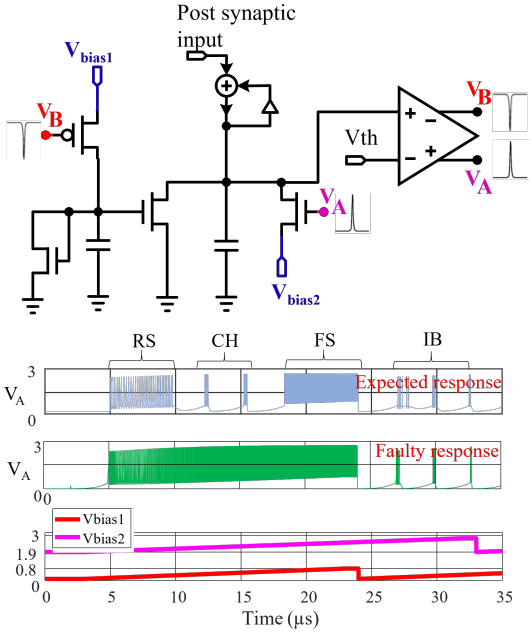

Fig. 13: Test generation for a biologically-inspired neuron.

operation. A fault is detected if the time exceeds the expected nominal range computed by a process variation simulation.

B. On-line testing based on in-situ monitors

The idea here is to add monitors into the design that detect symptoms of abnormal operation. In [12], a compact monitor, illustrated in Fig. 14, is proposed to detect neuron saturation which is the most lethal fault. It counts the number of output spikes with the counter being reset when an input spike arrives. The counter overflows and an error is flagged if many spikes are generated without any input activity. In [18], neuron saturation is detected if its membrane potential stays above the threshold for more than two clock cycles.

To reduce the overhead of adding one monitor per neuron, in [31], it is proposed to check for spike saturation at the feature map level in the case of convolutional SNNs. The cumulative spike counts at each feature map are used as new features and are subsequently mapped to a go/no-go test decision using an on-chip classification system. In essence, an AI system is used to test the AI hardware accelerator. This idea is demonstrated on hardware using the accelerator in [38], where the classification system runs on the processor of the board.

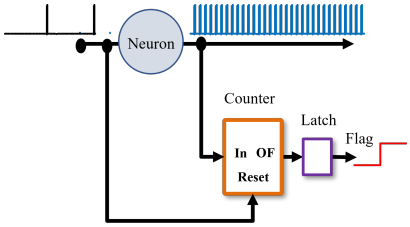

Fig. 14: Neuron saturation symptom detector.

V. Hardware-based fault tolerance

1) Redundancy-based: A classical approach is Triple Modular Redundancy (TMR) where the system is triplicated and a voting scheme is implemented to decide on the correct response, given that it is unlikely that two systems will fail simultaneously. Full-system triplication is of course very costly. However, fault injection experiments have shown that different layers have different sensitivity to faults. Therefore, we can perform selective TMR by applying it to critical layers only [12]. For example, the most critical output layer is recommended to be protected with TMR.

2) Hardening: Another classical approach is hardening of vulnerable components of the architecture, which refers to applying changes in their layout so as to tolerate exposure to radiation. In [41], a hardened spiking neuron design is proposed. In [18], it is proposed to harden only the hardware extensions that are used to enable fault tolerance.

3) Pruning: In [20], it is proposed to prune or bypass faulty PEs in the accelerator, which requires testing as a first step to determine the faulty PEs. Thereafter, fault-aware training is performed to re-train the unpruned weights while optimizing the neuron's threshold for each layer.

4) Memory fault mitigation: To mitigate memory faults, in [14] it is proposed to first derive the memory fault map using memory testing, then perform bit shuffling to prioritize placing the MSBs of the weights on the non-faulty memory cells.

5) Fault masking: Here, the idea is to mask the effect of the fault upon detection. In Section III, dropout was shown to nullify the effect of dead neuron faults. Thus, if neuron saturation is detected, for example using one of the in-situ monitors described in Section IV-B, one approach is to disable the neuron [12], [18]. In [12], this concept is called "fault hopping" and it is implemented by adding a single transistor into the neuron that cuts off its power if the flag signal in Fig. 14 is raised. For synapses, if the weight is increased or, equivalently, an abnormal current comes into the post-synaptic neuron, then one approach is to zero the weight [18], [33]. At worse this makes the post-synaptic neuron dead.

6) Astrocyte neural networks: In [22], [24]–[26], [29], it is proposed to mimic the self-repairing capability of the biological brain. Astrocytes are added into the network, where each astrocyte communicates with a set of neurons and their incoming synapses, as illustrated in Fig. 15. Astrocytes are capable of regulating the synaptic transmissions. For example, when a synapse breaks, they enhance the probability of release of the healthy synapses which can help the neuron maintain its firing frequency.

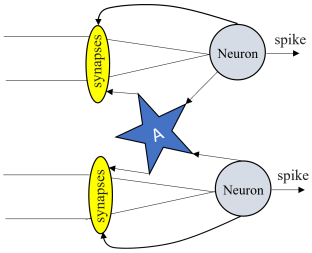

Fig. 15: Astrocyte neural network.

7) Neuron adaptation: In [23], fault tolerance is achieved by means of redundant synapses per neuron and neuron adaptation. In particular, the neuron monitors the total injected current from all synapses during a time window, and if an abrupt or abnormal variation is noticed, then the neuron's threshold or operating frequency are adjusted such that it retains the same firing rate. Vice versa, neuron's threshold variations can be compensated by adjusting the inference time steps [33].

8) Re-learning: In [9], a high-level biologically-inspired model of the cortical structure of the brain is developed. The model is trained using Hebbian learning with repeated exposure to input samples. For dead neurons, the network is capable of re-learning as their functionality is taken over by neighboring neurons. On the other hand, saturated 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 by forming a TMR voting scheme using the existing redundancy.

9) Memristor crossbar arrays: Current memristor technologies are known to have low yield and endurance. The lifetime of the memristor depends on many factors, including: (a) its programmed resistance state according to the model mapping onto the crossbar; (b) the current that flows though it which depends on its position on the crossbar (i.e., the current in the lower left cell is higher than this of the upper right cell since the path is shorter and the voltage drop is smaller); and (c) the criticality of the synapse which depends on the workload. To this end, in [27], [32] frameworks are proposed aiming at finding an endurance-aware mapping of synapses onto the hardware, such that critical synapses are implemented on memristors with high endurance.

VI. Conclusions and Perspectives

We analyzed the emerging trends in the test and reliability of SNN hardware accelerators. Moving forward, fault models may need to expand to cover additional failure modes in modern accelerators. An automated, versatile, and extendable fault injection framework is yet to be made open source, which can become a useful tool in the hands of the designers for accelerating reliability analysis. Viewing an accelerator as a black-box, one interesting problem is to generate a single "golden" test image capable of detecting any fault. Model-based fault tolerance has the advantage that it does not require any hardware modifications, thus withstanding the largest number of faults at this stage is very attractive as it reduces the

on-chip provisions needed for hardware-based fault tolerance. Developing low-overhead in-situ self-test monitors and fault mitigation strategies is another interesting research direction. Finally, memristor crossbar array architectures require new test and reliability strategies which are of vital importance given the fragility of memristors.

REFERENCES

[1] K. Roy, A. Jaiswal, and P. Panda, "Towards spike-based machine intelligence with neuromorphic computing," *Nature*, vol. 575, no. 7784, pp. 607–617, Nov. 2019.

[2] C. D. Schuman, S. R. Kulkarni, M. Parsa, J. P. Mitchell, P. Date, and B. Kay, "Opportunities for neuromorphic computing algorithms and applications," *Nat. Comput. Sci.*, vol. 2, no. 1, pp. 10–19, Jan. 2022.

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

[4] 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.

[5] 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.

[6] M. Bouvier *et al.*, "Spiking neural networks hardware implementations and challenges: A survey," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 15, no. 2, Apr. 2019.

[7] A. Basu, L. Deng, C. Frenkel, and X. Zhang, "Spiking neural network integrated circuits: A review of trends and future directions," in *Proc. IEEE Cust. Integr. Circuits Conf. (CICC)*, Apr. 2022.

[8] F. Su, C. Liu, and H.-G. Stratigopoulos, "Testability and dependability of AI hardware: Survey, trends, challenges, and perspectives," *IEEE Des. Test*, vol. 40, no. 2, pp. 8–58, Apr. 2023.

[9] 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.

[10] 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.

[11] 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.

[12] 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, pp. 743–748.

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

[14] 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.

[15] 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.

[16] Y.-Z. Hsieh, H.-Y. Tseng, I. Chiu, and J. C. M. Li, "Fault modeling and testing of spiking neural network chips," in *Proc. IEEE Int. Test Conf. Asia (ITC-Asia)*, Aug. 2021.

[17] 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.

[18] 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.

[19] A. Colucci, A. Steininger, and M. Shafique, "enpheeph: A fault injection framework for spiking and compressed deep neural networks," in *IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Oct. 2022, pp. 5155–5162.

[20] A. Siddique and K. A. Hoque, "Improving reliability of spiking neural networks through fault aware threshold voltage optimization," in *Proc. Design Autom. Test Europe Conf. (DATE)*, Apr. 2023.

[21] J. Li, B. Sun, and X. Xie, "A reliability assessment approach for a LIF neurons based spiking neural network circuit," in *Proc. Int. Conf. Thermal Mech. Multi-Phys. Simulation Exp. Microelectron. Microsyst. (EuroSimE)*, Apr. 2023.

[22] S. Karim *et al.*, "Assessing self-repair on FPGAs with biologically realistic astrocyte-neuron networks," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI (ISVLSI)*, Jul. 2017, pp. 421–426.

[23] 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.

[24] J. Liu, J. Harkin, L. P. Maguire, L. J. McDaid, and J. J. Wade, "SPANNER: A self-repairing spiking neural network hardware architecture," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 4, pp. 1287–1300, Apr. 2018.

[25] S. Karim *et al.*, "FPGA-based fault-injection and data acquisition of self-repairing spiking neural network hardware," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2018.

[26] S. Karim, J. Harkin, L. McDaid, B. Gardiner, and J. Liu, "AstroByte: Multi-FPGA architecture for accelerated simulations of spiking astrocyte neural networks," in *Proc. Design Autom. Test Europe Conf. (DATE)*, Mar. 2020, pp. 1568–1573.

[27] T. Titirsha *et al.*, "Endurance-aware mapping of spiking neural networks to neuromorphic hardware," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 2, pp. 288–301, Feb. 2022.

[28] 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, pp. 370–375.

[29] M. Isik, A. Paul, M. L. Varshika, and A. Das, "A design methodology for fault-tolerant computing using astrocyte neural networks," in *Proc. 19th ACM Int. Conf. Comput. Frontiers (CF)*, May 2022, p. 169–172.

[30] K.-W. Hou, H.-H. Cheng, C. Tung, C.-W. Wu, and J.-M. Lu, "Fault modeling and testing of memristor-based spiking neural networks," in *Proc. IEEE Int. Test Conf. (ITC)*, Sep 2022, pp. 92–99.

[31] T. Spyrou and H.-G. Stratigopoulos, "On-line testing of neuromorphic hardware," in *Proc. IEEE Eur. Test Symp. (ETS)*, May 2023.

[32] A. Paul, S. Song, T. Titirsha, and A. Das, "On the mitigation of read disturbances in neuromorphic inference hardware," *IEEE Des. Test*, vol. 40, no. 2, pp. 100–108, Apr. 2023.

[33] A. Saha, C. Amarnath, and A. Chatterjee, "A resilience framework for synapse weight errors and firing threshold perturbations in RRAM spiking neural networks," in *Proc. IEEE Eur. Test Symp. (ETS)*, May 2023.

[34] 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.

[35] 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.

[36] 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.

[37] G. Orchard, A. Jayawant, G. K. Cohen, and N. Thakor, "Converting static image datasets to spiking neuromorphic datasets using saccades," *Front. Neurosci.*, vol. 9, Nov. 2015, Article 437.

[38] 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.

[39] 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.

[40] 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.

[41] 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, Jun. 2021.