Compact Functional Testing for Neuromorphic Computing Circuits

Sarah A El-Sayed, Theofilos Spyrou, Luis A Camuñas-Mesa, Haralampos-G. Stratigopoulos

Compact Functional Testing for Neuromorphic Computing Circuits

Sarah A. El-Sayed, Theofilos Spyrou, Luis A. Camuñas-Mesa,
and Haralampos-G. Stratigopoulos, *Member, IEEE*

***Abstract*—We address the problem of testing Artificial Intelligence (AI) hardware accelerators implementing Spiking Neural Networks (SNNs). We define a metric to quickly rank available samples for training and testing based on their fault detection capability. The metric measures the inter-class spike count difference of a sample for the fault-free design. In particular, each sample is assigned a score equal to the spike count difference between the first two top classes. The hypothesis is that samples with small scores achieve high fault coverage because they are prone to misclassification, i.e., a small perturbation in the network due to a fault will result in these samples being misclassifiifed with high probability. We show that the proposed metric correlates with the per-sample fault coverage and that retaining a set of high-ranked samples in the order of ten achieves near perfect fault coverage for critical faults that affect the SNN accuracy. The proposed test generation approach is demonstrated on two SNNs modelled in Python and on actual neuromorphic hardware. We discuss fault modeling and perform an analysis to reduce the fault space so as to speed up test generation time.**

***Index Terms*—Neuromorphic computing, spiking neural networks, testing, fault modeling, fault simulation.**

I. Introduction

The design of Artificial Intelligence (AI) hardware accelerators has attracted significant attention in recent years. On one hand, AI hardware accelerators are essential for efficient processing of high-dimensional AI workloads, i.e., Deep Neural Networks (DNNs). On the other hand, technologies such as edge computing are witnessing growing interest with several new use cases. Designing lightweight AI hardware accelerators, i.e., having low energy consumption and form factor, for performing inference directly on edge devices can offer significant advantages compared to performing the computation centrally in the cloud. In particular, performing computations closer to the user's side allows autonomous operation, saves energy and bandwidth by reducing data transfer, circumvents cloud service latency, and assures data privacy.

High-volume manufacturing of Application-Specific Integrated Circuit (ASIC) AI hardware accelerators is foreseen in the near future. On-die neural networks have been explored in the past for building an on-die "test brain" that classifies chips as functional or faulty [1]. The "inverse" problem, i.e., how to efficiently test AI hardware accelerators, however, is an emerging problem [2]–[4].

In general, existing and proven test methods for traditional computing devices can be portable to AI hardware accelerators. Nevertheless, the unique architectural features of AI hardware accelerators make these test methods less efficient and give rise to new test challenges. For instance, AI hardware accelerators usually consist of multiple identical cores, e.g., the Multiply-and-Accumulate (MAC) units (also referred to as Processing Elements (PEs)), which are too small to implement traditional Design-for-Test (DfT) techniques, e.g., scan test, with reasonable overhead. Another characteristic of AI hardware accelerators is that they are memory-hungry, with the memory storage being dominated by the synapse weights which can be in the order of millions. Testing large embedded memories with today's Memory Built-In Self-Test (MBIST) tools can pose large Power, Performance, and Area (PPA) penalties.

Traditional fault models, such as stuck-at, delay, and cell-aware fault models, can be reused as well in the context of AI hardware accelerator testing, but emerging in-memory computing architectures based on memristive crossbars [5] or Spiking Neural Networks (SNNs) [6]–[8] for example, require new fault models. Moreover, fault models for AI hardware accelerators could be defined in software at a higher-abstraction behavioral-level, i.e., variations in neuron outputs and synapse weight values [9], aiming at speeding up test generation. This is because software and hardware implementations of neural networks closely match together [10]. Recent fault injection experiments in AI hardware accelerators [10]–[16] have shown that there is some inherent fault tolerance since many faults are benign, i.e., they are masked before their effect propagates to the output layer or they produce an output change that is tolerable. This is expected based on the analogy with biological neural networks which have remarkable fault tolerance capabilities. Some faults, though, are still critical and evidently impact the performance, and test efforts can focus on these critical faults to reduce test time [4].

Test methods that take into account the architectural par-

Manuscript received July 22, 2022; revised October 31, 2022; accepted November 6, 2022. This work was supported by the ANR RE-TRUSTING project under Grant ANR-21-CE24-0015-03 and by Junta de Andalucía under contract US-1260118 (Neuro-Radio). The work of T. Spyrou was supported by the Sorbonne Center for Artificial Intelligence (SCAI) through Fellowship. L. A. Camuñas-Mesa was funded by the VI PPIT through the Universidad de Sevilla. This article was recommended by Associate Editor G. Di Natale. (Corresponding author: Haralampos-G. Stratigopoulos.)

Sarah A. El-Sayed was with Sorbonne Université, Centre National de la Recherche Scientifique (CNRS), LIP6, 75005 Paris, France. She is now with Minia University, 61519 Minia, Egypt (e-mail: sarah.elsayed@lip6.fr).

T. Spyrou and Haralampos-G. Stratigopoulos are with the Sorbonne Université, CNRS, LIP6, 75005 Paris, France (e-mail: theofilos.spyrou@lip6.fr; haralampos.stratigopoulos@lip6.fr).

Luis A. Camuñas-Mesa is with the Instituto de Microelectrónica de Sevilla (IMSE-CNM), Consejo Superior de Investigaciones Científicas (CSIC), Universidad de Sevilla, 41092 Sevilla, Spain (e-mail: camunas@imse-cnm.csic.es).

Digital Object Identifier 10.1109/TCAD.2023.XXXXXXX

ticularities of AI hardware accelerators have started surfacing recently. A brief overview of the state-of-the-art is provided in Section II. In this work, we focus on SNN hardware accelerators for which the literature on fault modeling, fault injection experiments, test and reliability methods is still fairly limited, as it will also be discussed in Section II. More specifically, we propose a method for generating compact functional test sets. A functional test is an input, i.e., an image for Convolutional Neural Networks (CNNs) used in computer vision, that aims at detecting the presence of a maximum number of hardware-level faults and their combination thereof, such that a compact set of few functional tests can detect all faults. In the context of an AI hardware accelerator device, a fault is detected if the predictions of the nominal fault-free and faulty devices differ. The prior art on functional test generation methods for AI hardware accelerators will be described in detail in Section VIII.

This article makes the following contributions:

- We use available samples from the training and testing sets as functional tests. We propose a fault-agnostic metric that assesses the fault detection capability of each sample based on how close the sample is to being misclassified. The underlying hypothesis is that such corner samples lying close to the classification hyper-boundary will be classified differently if a fault occurs, thus producing a distinguishing output compared to the nominal fault-free case.
- We confirm experimentally that the fault detection capability of a sample is directly correlated with the ranking of the sample based on the chosen metric.
- Based on this metric, we cumulatively add test samples according to their ranking and demonstrate that we can generate compact functional test sets with the size of a few tens of tests that achieve 100% fault coverage of critical faults. A good percentage of benign faults can also be detected, which is beneficial since they are considered as reliability hazards. Such compact functional test sets can be used not only for fast post-manufacturing testing, but they can also be stored on-chip to be re-used for periodical on-line test in idle times.
- We discuss fault modeling and we perform fault injection experiments for different fault models identifying critical fault types and fault locations. These findings, corroborated on various SNN models, can be used to reduce the effective fault space and, thereby, the fault simulation time for computing fault coverage.
- The proposed compact functional test set generation method is demonstrated for SNNs at behavioral-level and on actual neuromorphic hardware.

The rest of the article is structured as follows. In Section II, we discuss the state-of-art on testing AI hardware accelerators including SNNs. In Section III, we provide a brief overview of SNNs. In Section IV, we introduce the proposed functional test generation algorithm. In Section V, we present the SNN case studies. In Section VI, we discuss fault modeling, fault simulation frameworks, and fault space reduction. Experimental results are presented in Section VII. In Section VIII, we discuss, in more detail, the prior art on functional test generation for AI hardware accelerators and provide comparisons to our work. In Section IX, we discuss general properties of the proposed functional test generation approach and future work directions. Finally, Section X concludes this article.

II. Prior art on testing AI hardware accelerators

DfT methods suited for AI hardware accelerators based on large arrays of small PEs are proposed in [2], [17]. Test generation algorithms aiming at creating a compact set of functional tests that can detect the presence of faults are proposed in [18], [19], [20]. Symptom detectors that detect some anomaly in intermediate nodes, i.e., high neuron activation, are proposed in [11], [21]–[23]. Selective Triple Modular Redundancy (TMR) applied to the most critical neural network layers is proposed in [22], [24], [25]. Algorithmic-based error detection and correction methods using checksum arithmetic are discussed in [13], [26]–[29]. On-line test methods are proposed in [30] based on Software Test Libraries (STL), in [31] based on a simplified metric of dynamic power consumption, and in [32] based on encrypting weights in the memory with an encryption algorithm that spreads single bit-flips extending them to multiple bit-flips and checking if the padding bytes used for the encryption to work properly are correctly decrypted.

Specifically now for SNNs, behavioral-level fault modeling is discussed in [33], [34]. Fault injection experiments at behavioral-level are described in [22], [23], [33], [35], [36], while fault injection experiments onto actual neuromorphic hardware are described in [16]. In [22], [23], symptom detectors are designed for the two main catastrophic fault mechanisms in SNNs. In particular, for synaptic faults it is proposed to perform weight bounding [23]. If the weight drifts to a value 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 an output spike train is observed while there is no input activity, then spike generation is disabled and the neuron is silenced [22]. A similar condition is proposed in [23] where spike generation is disabled if the membrane voltage stays above the threshold for more than two clock cycles. A BIST technique for biological spiking neurons that checks for the appearance of all expected firing patterns is proposed in [37]. Inherent fault resilience of SNNs when trained with different algorithms is studied in [35], in addition to showing how to modify a training algorithm to improve fault tolerance. Fault-tolerance schemes to mitigate memory failures in SNNs are proposed in [36], [38]. And finally, a functional test generation method for SNNs is proposed in [20], which will be discussed in more detail in Section VIII.

III. SNNs

Unlike traditional Artificial Neural Networks (ANNs), information in SNNs flows as spike trains propagating across network layers asynchronously, which resembles the biological brain operation. SNNs form the basis of neuromorphic computing pioneered by Carver Mead in the 1980s [39]. One of the least computationally complex, and hence hardware-friendly,

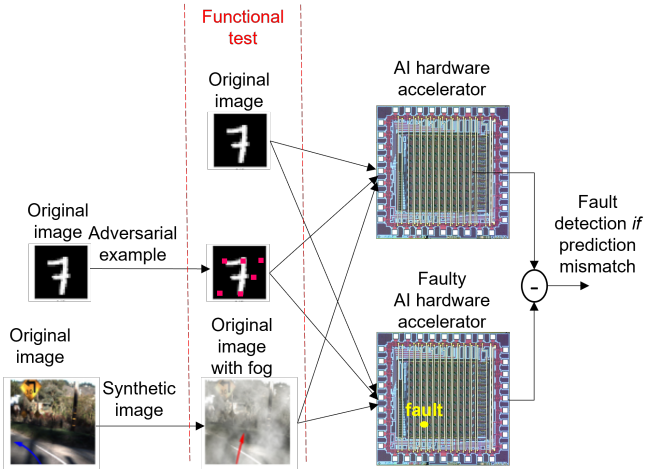


Fig. 1: Functional test generation. The street images are from [41]. The chip image is from [1].

neuron model used to implement SNNs is the Integrate & Fire (I&F) model [40]. The I&F neuron integrates spikes from incoming synapses by increasing its membrane's potential, and when the potential 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 I&F neuron can be equipped with two additional brain-inspired functionalities, namely 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 in SNNs also resembles the biological synapse operation where a synapse receives spikes and, in turn, stimulates the membrane potential of post-synaptic neurons via a current that is proportional to the synapse weight. As for decision making methods at the output of SNNs, the most common method is based on evaluating the firing rate of output neurons, i.e., number of spikes over an observation period. Other coding schemes have been used, such as the time-to-first-spike. From a hardware perspective, there is a belief that SNNs offer faster inference and lower energy consumption compared to ANNs. However, there are still challenges facing SNNs such as the complexity of training. 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. For an extensive discussion on SNNs and their hardware implementation, the readers are referred to [6]–[8].

IV. Proposed Functional Test Generation Algorithm for SNNs

Functional test generation aims at either identifying or generating new input samples 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. As shown in Fig. 1 using as an example an image recognition cognitive task, these samples could be original images from training and testing sets [18], [19], adversarial examples generated from original images [20], or synthetic images generated from original images [41], [42]. These related works will be described in more detail in Section VIII. The proposed algorithm for SNNs selects tests from the set of available samples in the training and testing sets.

Let us consider an SNN employed for an N-class classification cognitive task. The SNN has N neurons in the output layer each corresponding to one class. We consider that the SNN uses the firing rate as classification criterion, i.e., the winning class is the one whose corresponding neuron produces the largest number of spikes within a given duration interval.

Let us consider also a set of input samples of cardinality M. For a given sample t_i, $i = 1, \cdots, M$, let n_i^j denote the spike count for output neuron j, $j = 1, \cdots N$. We rank the values n_i^j from high to low, resulting in the ordered set $\{n_i^{(1)}, n_i^{(2)}, n_i^{(3)}, \cdots\}$, i.e., the neuron that produces $n_i^{(1)}$ spikes corresponds to the top-1 class, the neuron that produces $n_i^{(2)}$ spikes corresponds to the top-2 class, and so forth.

Then, we define the *margin*

$$X_i = n_i^{(1)} - n_i^{(2)}, \tag{1}$$

i.e., X_i is the difference in spike count between neurons corresponding to the top-1 and top-2 classes. The *quality metric* of sample t_i is defined by the score

$$q_i = \frac{1}{X_i} \tag{2}$$

The rationale is that samples with high scores (or, equivalently, small margins) are likely to be *distinguishing samples*, i.e., they are prone to producing different top-1 class predictions for the nominal and faulty networks. This is based on the intuition that when the first top classes are close in terms of firing rate, the network has low confidence in its decision and it is likely that a fault will alter the class ranking for this particular sample. In other words, this sample lies close to the multi-class classification hyper-boundary and is likely to be misclassified when a fault occurs.

Let us consider now the ranking of samples based on their scores from high to low, resulting in the ordered set $\{t^{(1)}, t^{(2)}, \cdots, t^{(i)}, \cdots, t^{(M)}\}$, where $t^{(1)}$ is the sample with the highest score, and so forth. A functional test set of size T can be generated by considering the first T higher-score samples in this ordered set.

As we will see in our experimental results, the score in Eq. (2) directly correlates with the per-sample fault coverage, i.e., the samples fault coverage is shown to increase linearly with the samples score. To this end, the proposed algorithm first performs M inferences, i.e., one inference per available sample, on the nominal network, then ranks the samples according to their score. Next, starting from the top ranked sample and sequentially adding the next top ranked samples, it evaluates the cumulative fault coverage. In each step, the detected faults are dropped from the fault list. Let $N_{uf}(i)$ denote the number of undetected faults at the beginning of iteration i where the i-th ranked sample is examined, i.e., $N_{uf}(1) = K$ for a fault model of size K. The algorithm stops adding samples when the fault coverage saturates. Our experimental results show that the size of the resultant test

set needs to be in the order of few tens of samples for reaching 100% fault coverage for critical faults. For a test set of cardinality T, the total number of inferences which dominates the test generation time is

$$N_{inf} = M + \sum_{i=1}^{T} N_{uf}(i) \tag{3}$$

where the first term corresponds to test generation and the second term to fault coverage evaluation.

Finally, for a fault model of size K, let F_k denote fault k. We define the following indicator function for test t_i

$$I^{t_i}(F_k) = \begin{cases} 1 & : \quad F_k \text{ is detected} \\ 0 & : \quad \text{otherwise} \end{cases} \tag{4}$$

where detection means that the responses of the nominal fault-free network and the faulty network with fault F_k injected differ, i.e., a different class is predicted.

The fault coverage of test t_i indicates the percentage of faults detected by this particular test. It is defined as

$$FC(t_i) = \frac{\sum_{k=1}^{K} I^{t_i}(F_k)}{K} \tag{5}$$

Considering a test set of cardinality T denoted by $\{t_1, \cdots, t_T\}$, its global fault coverage is defined as

$$FC = \frac{\sum_{k=1}^{K} \min(1, \sum_{i=1}^{T} I^{t_i}(F_k))}{K} \tag{6}$$

V. SNN Case Studies

As case studies, we use three convolutional SNNs performing three different cognitive tasks, namely an SNN trained to classify the N-MNIST dataset [43], an SNN trained to classify IBM's DVS gesture dataset [44], and an SNN trained to classify the 4 poker card symbols [45]. Their architectures are shown in Figs. 2, 3, and 4, respectively. In each case, the winning class is declared based on the most triggered neuron at the output layer.

In CNNs, a convolutional layer is composed of several feature maps each forming a channel [46]. 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 may be 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 Fully Connected (FC) network. The last convolutional or sub-subling layer is flattened, i.e., each feature map is essentially a single neuron. Thereafter, there is a number of FC layers to perform the

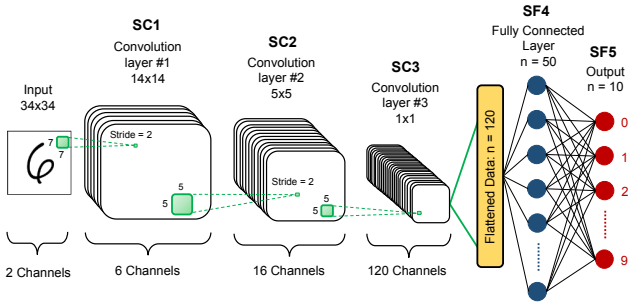


Fig. 2: Architecture of the N-MNIST SNN.

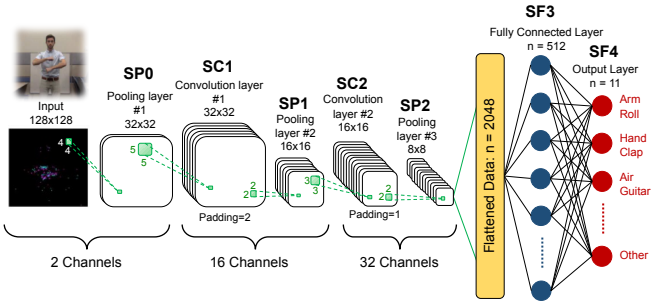


Fig. 3: Architecture of the IBM DVS Gesture SNN.

classification. In FC networks, the neurons of a new layer are connected via synapses to the outputs of all neurons in the prior layer.

The N-MNIST and IBM's DVS gesture SNNs are designed and simulated in Python, while the poker card-symbols SNN is running on an FPGA-based neuromorphic hardware platform.

A. SNN models in Python

The SNN models designed in Python are based on the open-source Spike LAYer Error Reassignment (SLAYER) [47] and PyTorch [48] frameworks, and run on a Graphics Processing Unit (GPU) accelerator. They are trained using batch learning with a variation of back-propagation, incorporating the dropout technique [49]. The SLAYER framework uses the Spike Response Model (SRM) [50] to model spiking neurons, which is a generalized form of the ubiquitous I&F model.

1) N-MNIST SNN: The N-MNIST dataset [43] is a neuromorphic version of the handwritten digits MNIST dataset. The SNN achieves a 98.08% classification accuracy on the testing set, which is comparable to the performance of state-of-the-art level-based ANNs.

2) IBM DVS Gesture SNN: The IBM's DVS gesture dataset [44] is created directly in spiking form using a 128×128 Dynamic Vision Sensor (DVS). It consists of 29 individuals performing 11 hand and arm gestures in front of the DVS, such as hand waving and air guitar. The SNN performs with an 82.5% accuracy on the testing set.

B. SNN hardware implementation

The poker card symbols dataset is created by presenting a deck of poker cards in front of a DVS sensor. The events are recorded and processed to extract a 32×32 pixel window

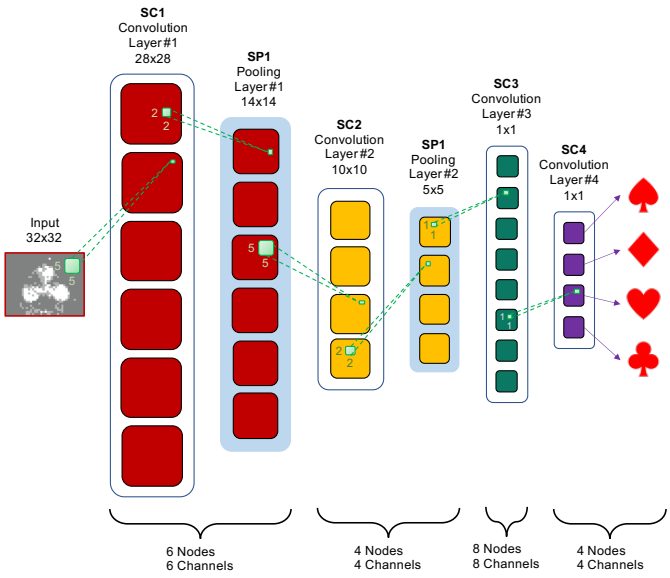


Fig. 4: Architecture of the poker card-symbols SNN.

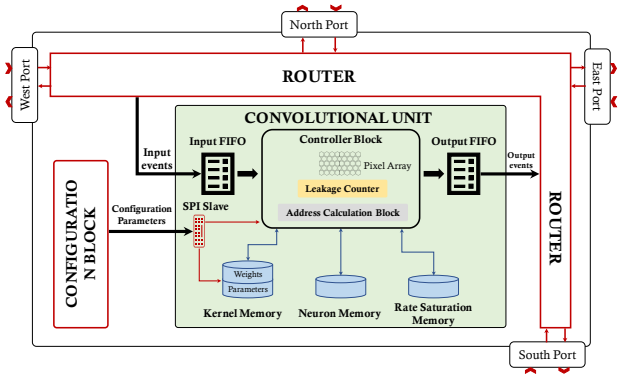


Fig. 5: Configurable convolutional node.

that shows only the centered symbol. The hardware implementation of the poker card-symbols SNN uses as a fundamental building block a generic event-driven configurable *convolutional node* that constitutes one feature map [45]. As shown in Fig. 5, each node consists of three main blocks, namely a *convolutional unit* that encompasses a 128×128 I&F neuron (pixel) array and is responsible for the convolution calculations, an internal *configuration block* that receives and interprets the configuration data of the convolutional node, and a *router* responsible for handling the transmission of the events from their origin to their destination according to a predefined routing scheme. Using this building block, the SNN of Fig. 4 is designed in VHDL and implemented on a Zynq®UltraScale+™ MPSoC ZCU104 FPGA board.

VI. Fault Modeling

Fault models can be defined either at behavioral-level using an abstract representation of the neural network, e.g., a Python model, or at hardware-level. Performing test generation at behavioral-level rather than at hardware-level, is faster and the derived tests are still actual samples, i.e., images, which are directly portable to the hardware. The disadvantage is that the resultant fault coverage may deviate from the true one computed on the actual hardware implementation. However, as it will be discussed in Section VI-C, for both options the fault space quickly explodes even for small-size neural networks, making fault sampling strategies essential. In this regard, performing test generation at the hardware-level yields approximative fault coverage. Similar to testing of traditional chips, fault coverage metrics should be reported by specifying the used underlying fault model.

A. Behavioral-level fault modeling

For the purpose of this analysis, faults are inserted in the main operators that support the neural computational task, i.e., neurons and synapses, assuming that neurons and synapses can fail independently [51].

1) Neurons: In [34], detailed transistor-level fault simulations were performed on a spiking neuron so as to deduce the possible output faulty behaviors. Main behavioral-level faults include: (i) *dead neuron* faults, i.e., the neuron halts spiking even in the presence of input activity; (ii) *saturated neuron* faults, i.e., the neuron fires spikes non-stop even at the absence of input activity; and (iii) timing variations, i.e., variations in the firing rate and the time-to-first-spike. Such neuron behavioral-level fault types are in agreement with recent works [20], [22], [23], [33] and encompass other fault types that have been considered in the past, such as stuck-at neurons and errors in the activation values [9].

2) Synapses: Common synapse faults considered in the literature include disabled synapses, stuck-weight synapses or saturated synapse weights [20], [35]. However, this synapse fault model is not realistic from a hardware perspective since real-weighted synapses after model training in software are quantized and stored as digital words in an on-die memory. Herein, we use a hardware-aware synapse fault model where we quantize the weight value given the data type in hardware, i.e., 8-bit integer in the neuromorphic hardware used in this work, we perform bit-flips across bit positions and then we map back the binary value to a real value. This fault model essentially corresponds to synapse weight perturbation that could be major if one or more Most Significant Bits (MSBs) are flipped.

3) Fault injection framework: For the neuron behavioral-level faults we use a fault injection framework described in [22], which is built on top of the SLAYER [47] and PyTorch [48] frameworks. Neuron faults are inserted by modifying the SRM model or by customizing the flow of computations in the frameworks. Synapse faults are inserted by modifying the weight values using as root-cause bit-flips as described above. Fault simulation is accelerated by creating a faulty SNN instance, then mapping it on a GPU to perform inference.

B. Hardware-level fault modeling

The neuromorphic hardware used in this work is reconfigurable storing all the main SNN parameters in on-die memories, including: (i) synapse weights; (ii) feature maps size and center-shift; (iii) neuron threshold, leakage, and refractory period; (iv) router parameters that configure

the architecture and direct spiking events; and (v) splitter parameters that define the input event copies to be sent to the first layer nodes. Each parameter has an 8-bit integer representation. We consider as fault model (i) single bit-flips for all parameters and across all bit positions and (ii) multiple bit-flips across the entire memory with different Bit Error Rate (BER) probabilities.

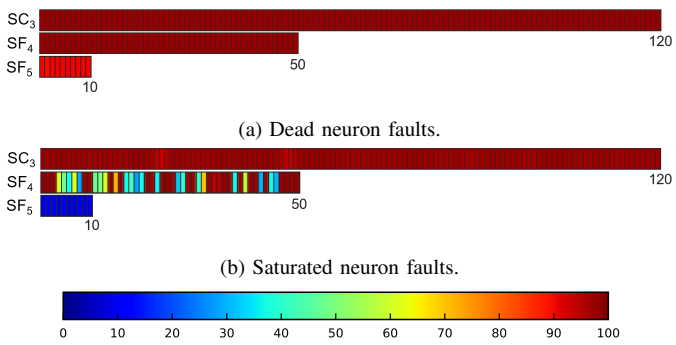
C. Fault space reduction

The common conclusion of several published fault injection and reliability experiments for ANNs [10]–[15] and SNNs [16], [20], [22], [23], [33], [35], [36] is that not all faults are equal. A large number of faults are either completely masked or they induce a negligible drop in the network classification accuracy. Such benign faults can be excluded to reduce the fault space and speed up fault simulation that is invoked several times during test generation. Including all faults can quickly make fault simulation intractable, even for small-size networks. For example, for the N-MNIST and IBM DVS Gesture SNNs the number of neurons is 1756 and 25099, respectively, and the number of synapses is 57488 and 1059616, respectively. Consequently, a prudent elimination of benign faults is required so as to avoid inadvertently excluding critical faults.

In the context of this work, we define benign and critical faults in a more strict fashion. A fault is considered benign if the response of the nominal network and the network in the presence of the fault match on a sample-by-sample basis with a certain tolerance, e.g., a 0% tolerance requires an exact match between both responses. If the number of mismatched samples between the nominal network response and that of the faulty network is above the tolerance, the fault is considered critical.

1) Behavioral-level fault model: For SNNs, in [22], it is shown that training with dropout [49] offers proactive fault resilience against dead neuron faults and neuron timing variations. Dropout was originally proposed to prevent over-fitting and reduce the generalization error on unseen data. It has its roots on the observation that model combination, a.k.a. ensemble learning, nearly always improves performance. In this regard, it temporarily removes neurons and their associated synapse connections during training with some probability that could vary from one layer to another, which is equivalent to combining many "thinned" scaled-down models during one training session. Dropout achieves fault resilience because it equalizes the importance of neurons and distributes the neuron activity across the network. Thus, if a neuron becomes dead or it presents timing variations, the impact is inherently tolerated [22]. In contrast, a neuron saturation fault is not compensated because a neuron constantly firing is likely to severely perturb the propagating spike trains [22], [23].

As an example, Fig. 6 shows for the N-MNIST SNN the effect of neuron faults in the last 3 layers on the network classification accuracy. Each row corresponds to a layer and each rectangle within a row corresponds to a neuron in this layer. The color of the rectangle shows the accuracy when this particular neuron is faulty based on the color map shown at the bottom of Fig. 6. As it can be seen, dead neuron faults are benign except in the last layer. Layers SC1 and SC2 are not shown due to their high neuron count. Similarly, timing variations are proven benign apart from the last layer [22]. The same findings were obtained for the IBM DVS Gesture SNN.



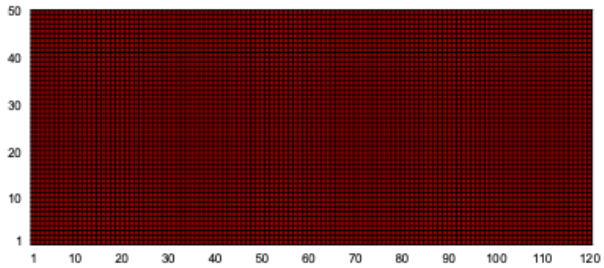
(a) Dead neuron faults.

(b) Saturated neuron faults.

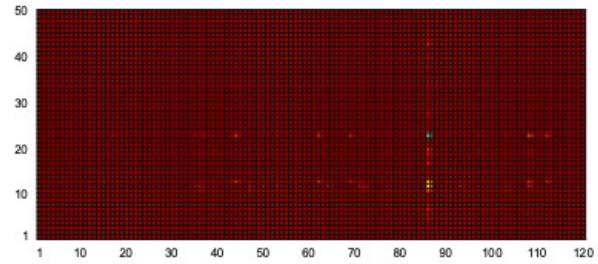
Fig. 6: Effect of neuron faults on the classification accuracy of the N-MNIST SNN.

As for synapse faults, fault space reduction becomes of utmost importance since the number of synapses can be in the order of several millions. In [20], only the last layer synapse faults were considered arbitrarily. In this work, we performed an analysis for the N-MNIST SNN to validate this hypothesis. The result is shown in Fig. 7. We assumed extreme synapse faults, namely dead synapses and positively saturated weights. Each box in Fig. 7 corresponds to one synapse connecting two neurons in two subsequent layers $j-1$ and j, with the neuron numbers for layers $j-1$ and j shown in the row and columns, respectively. The classification accuracy in the presence of a synapse fault is shown with the box color according to the color map at the bottom of Fig. 7. As it can be seen, only positively saturated weights appear to be critical, and this holds primarily for the synapses connecting the last two layers SF4-SF5, while few such critical synapse faults are observed for layers SC3-SF4. Synapse faults in previous layers have no impact and are excluded from Fig. 7. The reason behind this observation is that positive saturated weights could cause the post-synaptic neuron to always fire, i.e, saturate. In contrast, dead synapses just reduce the firing activity of the post-synaptic neuron, while we know that with dropout dead neuron faults are benign. These findings were corroborated on the IBM DVS Gesture SNN as well. Note that such extreme faults are not realistic from a hardware perspective as explained in Section VI-A2. Using the hardware-aware synapse fault model, i.e., bit-flips on quantized weights resulting in weight perturbation, has a smaller impact as shown in Fig. 8 for the IBM DVS Gesture SNN considering synapses connecting the last two layers SF3-SF4. We observe that only bit flips in the first two MSBs 7 and 6 can be critical, while for bit 5 the baseline accuracy is observed. Bit-flips for bit positions 0-4 are benign and are not shown in Fig. 8.

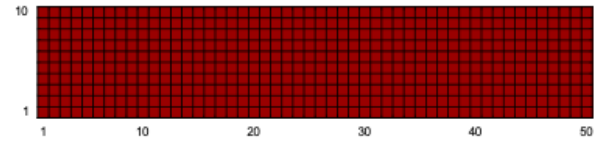
From these fault injection experiments we can conclude about the criticality of fault types occurring at different locations in the network. Critical faults most likely can only be neuron saturation faults at any point in the network, dead



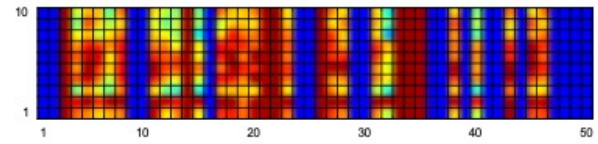
(a) Dead synapses between SC3-SF4.



(b) Positively saturated synapses between SC3-SF4.



(c) Dead synapses between SF4-SF5.



(d) Positively saturated synapses between SF4-SF5.

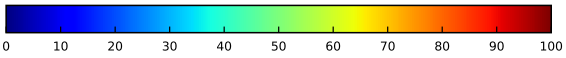


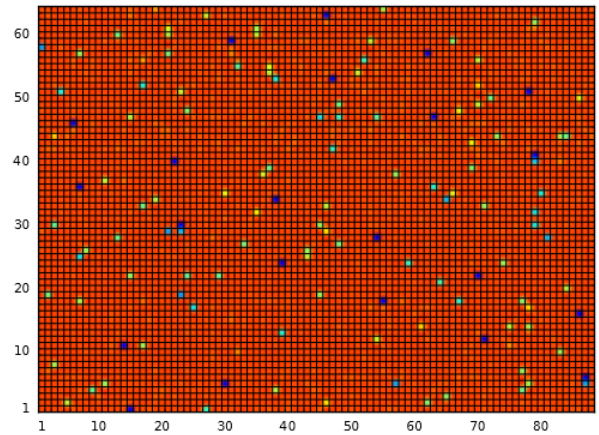
Fig. 7: Effect of synapse faults on the classification accuracy of the N-MNIST SNN.

neuron faults in the last two layers, and synapse faults in the last two layers. Still, many of these faults will end up being benign. The rest of the faults are benign with a very high probability and could be excluded from fault simulation.

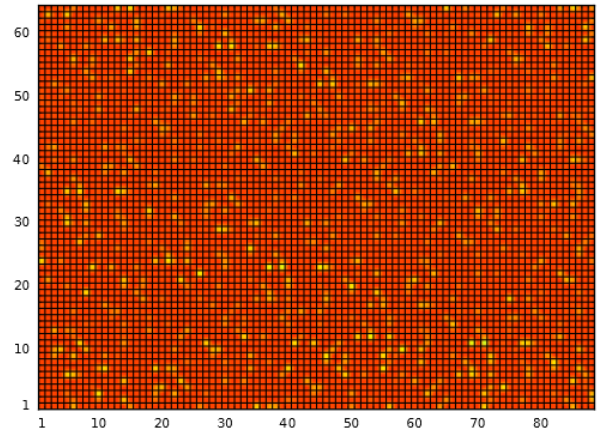
2) Hardware-level fault model: A detailed reliability analysis for the poker card symbols SNN implemented on neuromorphic hardware was presented in [16]. The used fault model is the one described in Section VI-B and fault injection was performed on the actual hardware. Critical bit-flip faults were located across different network parameters and bit positions. For example, it was shown that bit-flips in the 4 Least Significant Bits (LSBs) of synapse weights are benign which can help to significantly reduce the fault space by nearly 50% since synapse weights occupy most of the memory size.

VII. Results

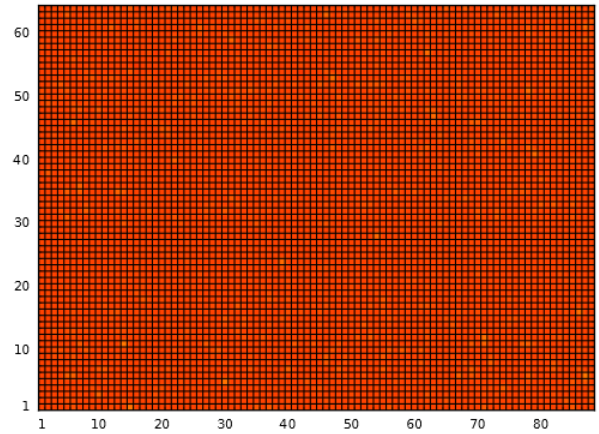
Thanks to the acceleration of fault injection on hardware, i.e., using a GPU for the N-MNIST and IBM DVS Gesture SNNs and the FPGA-based SNN hardware accelerator for the



(a) Bit 7 (MSB).



(b) Bit 6.



(c) Bit 5.

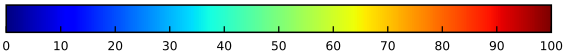
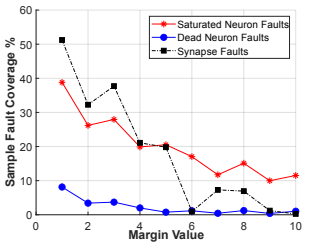
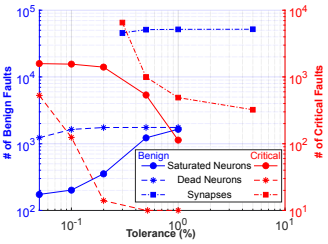


Fig. 8: Effect of bit-flip synapse faults between SF3-SF4 for the IBM DVS Gesture SNN.

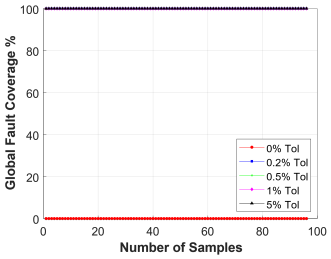
poker card symbols SNN, we considered a conservative fault space reduction only for the synaptic faults and only in the N-MNIST and IBM DVS Gesture SNNs. In particular, we considered only the synapse faults in the last two layers. The results are grouped per SNN in Figs. 9-11.



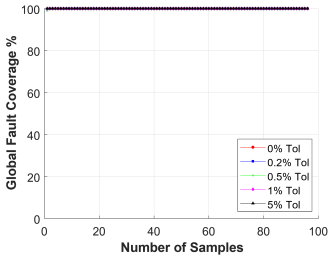
(a) Relationship between the margin of a sample and the percentage of faults it covers.



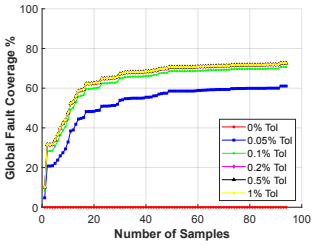
(b) Change in number of critical & benign faults with the tolerance.



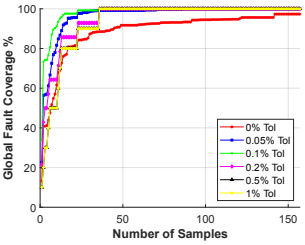
(c) Cumulative fault coverage of benign synaptic faults.



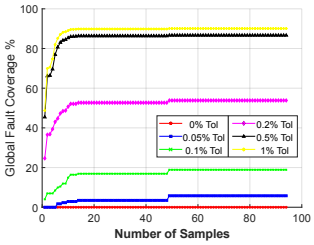
(d) Cumulative fault coverage of critical synaptic faults.



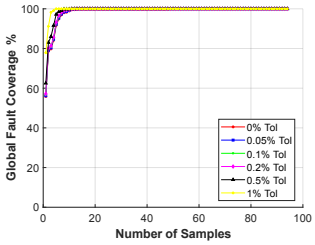
(e) Cumulative fault coverage of benign dead neuron faults.



(f) Cumulative fault coverage of critical dead neuron faults.

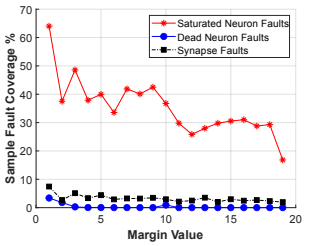


(g) Cumulative fault coverage of benign saturated neuron faults.

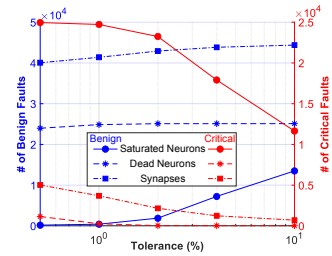


(h) Cumulative fault coverage of critical saturated neuron faults.

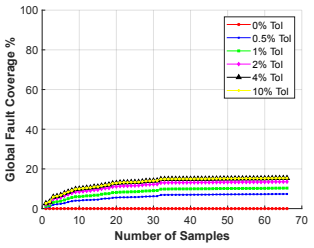
Fig. 9: N-MNIST SNN.



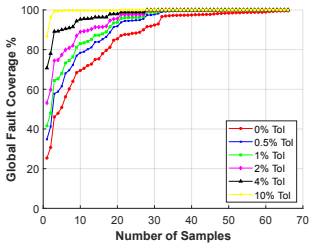
(a) Relationship between the margin of a sample and the percentage of faults it covers.



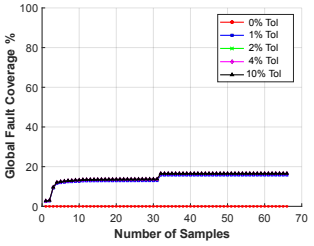
(b) Change in number of critical & benign faults with the tolerance.



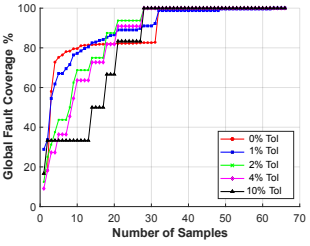
(c) Cumulative fault coverage of benign synaptic faults.



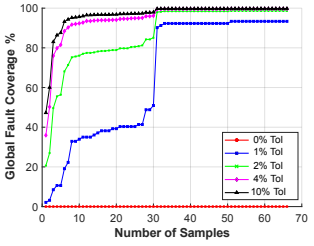
(d) Cumulative fault coverage of critical synaptic faults.



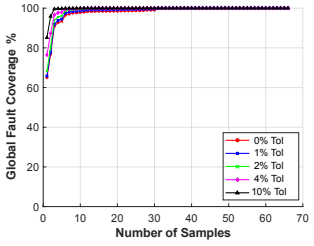
(e) Cumulative fault coverage of benign dead neuron faults.



(f) Cumulative fault coverage of critical dead neuron faults.

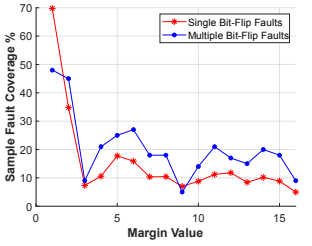


(g) Cumulative fault coverage of benign saturated neuron faults.

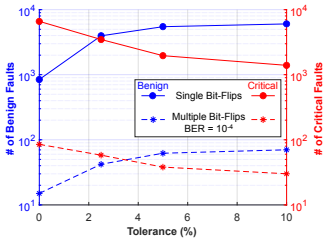


(h) Cumulative fault coverage of critical saturated neuron faults.

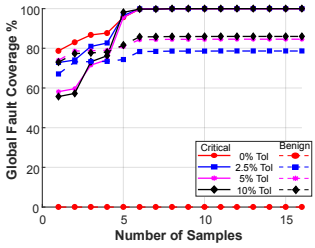
Fig. 10: IBM DVS gesture SNN.



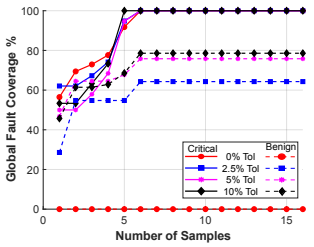
(a) Relationship between the margin of a sample and the percentage of faults it covers.



(b) Change in number of critical & benign faults with the tolerance.



(c) Cumulative fault coverage of single bit-flip faults.



(d) Cumulative fault coverage of multiple bit-flip faults at BER=10^{-4}.

Fig. 11: Poker card symbols SNN.

Figs. 9(a), 10(a), and 11(a) show the average per-sample fault coverage as a function of the margin value. Samples with identical margin values are grouped. For each sample, the average fault coverage is computed across all faults separately for each fault type, i.e., dead neuron, saturated neuron, and synapse faults for the N-MNIST and IBM DVS Gesture SNNs, and single bit-flip and multiple bit-flip faults with BER=10^{-4} for the poker card symbols SNN. Then, fault coverage values are averaged again across all samples within the same group. A clear trend is observed, i.e., samples with low margin, or equivalently with high score, tend to achieve a higher fault coverage. This proves the suitability of the chosen fault-agnostic metric for ranking samples according to their fault detection capability. We also observe that the fault coverage rapidly increases when the margin decreases for small margin values, while for large margin values, fault coverage values are flattened or show a small fluctuation. This means that input samples with relatively large margin values may show small deviation in their fault detection capability, whereas for input samples with small margin values, the deviation can be significant. Note also that a flat fault coverage curve does not necessarily imply that adding more samples with larger margins to the test set is meaningless since they may be achieving the same fault coverage, but at the same time they may be detecting a different set of faults compared to the faults already detected by samples of higher ranking.

Figs. 9(b), 10(b), and 11(b) show for the different fault types the number of benign and critical faults by varying the error tolerance. By increasing the tolerance from 0%, some critical faults are labelled as benign, thus the number of benign faults increases and the number of critical faults drops. The number of truly benign faults, i.e., none of the samples is misclassifed when these faults occur, is shown in Figs. 9(b), 10(b), and 11(b) for tolerance 0%. Fault type criticality for a given tolerance value can be assessed by examining the percentages of benign and critical faults. For tolerance 0% we observe that for all fault types, faults turn out to be more critical than benign, while for dead neuron faults, as expected, the critical and benign fault populations are more balanced. For all fault types, we observe that there is a tolerance value for which the critical and benign curves cross, which means that there is a certain tolerance value for which benign faults start outnumbering critical faults.

In Figs. 9(c)-(h), 10(c)-(h), and 11(c)-(d) we rank the samples in an ascending order according to their scores, and we show the global cumulative fault coverage as we add samples in the test set for the different fault types. Two general conclusions can be drawn here. First, for the critical faults, the global cumulative fault coverage curves quickly reach 100%, while the convergence speed in general increases with the tolerance. This is because higher tolerance means less critical faults and, thereby, smaller test effort to reach 100% fault coverage. The number of samples required to achieve 100% coverage varies from one SNN to another and from one fault type to another. Second, for the benign faults, the global cumulative fault coverage is not expected to reach 100% since many faults are truly benign. For tolerance 0%, the benign fault coverage is by definition zero. We also observe that the global cumulative fault coverage curves quickly saturate. The number of samples at the saturation point corresponds to the number of samples needed for detecting all benign faults that are not truly benign.

% Tol	Neuron Faults				Synapse Faults	
	Dead		Saturated		Benign	Critical
	Benign	Critical	Benign	Critical		
0	-	150	-	33	-	1
0.2	92	36	33	33	1	1
0.5	92	36	17	9	1	1

(a) NMNIST SNN.

% Tol	Neuron Faults				Synapse Faults	
	Dead		Saturated		Benign	Critical
	Benign	Critical	Benign	Critical		
0	-	32	-	25	-	66
2	13	28	31	19	35	35
4	13	28	31	15	32	28

(b) IBM DVS gesture SNN.

% Tol	Single Bit-Flips		Multiple Bit-Flips	
	Benign	Critical	Benign	Critical
0	-	6	-	6
2.5	6	6	6	6
5	6	6	6	6

(c) Poker card symbols SNN.

TABLE I: Number of samples needed to reach the maximum fault coverage for different fault types at different tolerance values.

Table I extracts some representative results from Figs. 9(c)-(h), 10(c)-(h), and 11(c)-(d), summarizing for each SNN and for three representative tolerance values the two most interesting quantities, i.e., the number of samples required to reach 100% fault coverage for critical faults and the number of samples required to detect all benign faults that are not truly benign. For example, for the N-MNIST SNN, we observe that the most highly ranked sample alone can detect all critical synapse faults for any tolerance value. To reach 100% fault coverage for critical dead neuron and saturated neuron faults for 0% tolerance, 150 and 33 samples are required, respectively. The required number of samples drops as the tolerance increases, i.e., for 0.2% tolerance 36 and 33 samples are required and for tolerance 0.5% 36 and 9 samples are required. Also from the cumulative benign fault coverage curves, we observe that, in general, for any tolerance value, the saturation point is reached before the maximum number of 150 samples required for detecting all critical faults at 0% tolerance, i.e., for 0.2% tolerance saturation is observed from 92, 33, and 1 samples onwards for benign dead neuron, saturated neuron, and synapse faults, respectively. For the IBM DVS gesture SNN, the number of samples required to achieve 100% critical fault coverage is 66, 35, and 28, for tolerance values 0%, 2%, and 4%, respectively. For these test sets, maximum coverage for not truly benign faults is achieved, except for tolerance 4% where an additional 31-28=3 samples from the ordered list need to be included. For the poker card symbols SNN, 6 samples suffice to detect all critical and not truly benign faults for any fault type.

Overall, results show that in all cases a compact functional test set is found that is capable of detecting all critical faults for any tolerance value, and as an auxiliary benefit it detects all benign faults that result in even the smallest accuracy drop, i.e. one sample is misclassified. Taking as an example the N-MNIST SNN for which $70,000$ samples are available from

the training and testing sets, the functional test set required to achieve 100% critical fault tolerance for all fault types is compacted to $150/(7 \cdot 10^4) = 0.214\%$ and $36/(7 \cdot 10^4) = 0.051\%$ of the combined size of the training and testing sets for tolerance 0% and tolerance $> 0\%$, respectively, where the maximum number of required samples across the three fault types is used.

VIII. Related work on functional test generation for AI hardware accelerators and comparison

The DeepXplore [42] and DeepTest [41] algorithms generate error-inducing corner test cases, which are then used for retraining the ANN aiming at improving classification accuracy. The criterion is maximizing neuron coverage, such that ideally for each neuron there is a test that makes it highly active. In the end, the generated synthetic samples represent real-world samples. For example, [41] starts with a subset of the testing set, called "seeds", then performs realistic transformations of seed images, such as changing brightness, changing contrast, rotation, blurring, fog effect, rain effect, etc. These test generation algorithms target improving classification performance and not hardware-level fault detection. However, it would be interesting to investigate whether synthetic samples generated in this fashion can also achieve high fault coverage.

In [18], a methodology to derive a diminutive set of functional test patterns for systolic array-based ANN accelerators is proposed. Two functional test pattern generation algorithms are proposed, namely an ANN model-agnostic algorithm and an ANN model-aided confidence-based algorithm. The ANN model-agnostic algorithm rates samples in the testing set based on their similarity to other samples belonging to different output classes. The similarity metric used is average pixel intensity. The ANN model-aided confidence-based algorithm searches for samples in the testing set that have been predicted correctly but with least confidence score. The proposed method in our work concerns SNNs and chooses samples that are prone to misclassification based on a different SNN-specific similarity metric defined based on the output spike trains.

In [19], it is proposed to rank and select a small subset of samples from the training set making the hypothesis that samples that require more neural network parameter tuning during training than others will be more sensitive to changes in neural network parameters due to faults. Tuning effort per sample is approximated with the change in the loss function in each training step. In the case where the model is pre-trained, a black-box approach is proposed to rank samples based on the difference in the loss function of a randomly initialized neural network instance and the pre-trained neural network. The methodology is demonstrated for memristive crossbar-based ANN hardware accelerators.

A functional test generation algorithm for SNNs is proposed in [20]. The algorithm produces a test set containing a mixture of available samples and adversarial examples. An adversarial example is generated by perturbing available samples by adding a minimum amount of noise such that the predictions of the nominal and faulty SNNs are differentiated. The algorithm starts by injecting a fault and examining if any of the available samples detects it. If not, up to D adversarial examples are generated, where D is a user-defined variable, aiming at finding one that detects the fault. If any available sample or adversarial example is 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. Since the number of synapse faults is too high, to solve the scalability issue only the last layer synapse faults are considered. In this work, we performed an experiment in Section VI-C to justify fault space reduction. Furthermore, the algorithm in [20] follows a greedy approach where tests are repeatedly evaluated on the undetected faults, combining test generation with fault coverage estimation. Thus, the number of inferences required, i.e., the test generation time, is a summation over tests and faults. On the contrary, the proposed algorithm in this paper dissociates test generation with fault simulation. Tests are first ranked by performing inference on the fault-free network, then fault coverage is computed only for the highly ranked tests. Thanks to the fault model agnostic ranking, the proposed method reduces dramatically the number of inferences. Finally, in the algorithm in [20], adversarial examples are useful for detecting benign faults since all critical faults are covered by original samples before entering the adversarial example generation loop. Adversarial example generation can be also seemingly added to the proposed algorithm in our work as a second step to boost benign fault detection.

IX. Discussion

A. Generality

The proposed functional test generation method is generic, treating the SNN architecture as a black-box. In this regard, the method is virtually applicable to any SNN hardware accelerator and neuromorphic computing hardware platform, including for example the SpiNNaker [52], TrueNorth [53], Loihi [54], BrainScaleS [55], Neurogrid [56], FPGA-based implementations [45], and application-specific small-scale chips [57]–[62].

B. Test generation effort

The proposed test generation algorithm can be decomposed into two steps: (a) the ranking of available input samples according to their fault coverage ability; and (b) the fault coverage assessment of a test set composed of highly ranked input samples.

Step (a) is agnostic to the fault model, using only inference data, i.e., number of spikes per output class for each sample in the training and testing sets from a pre-trained SNN model. This information is already available from the training phase, thus step (a) can be completed very fast independently of the SNN and dataset sizes.

Step (b) requires fault simulation, thus the effort is proportional to the SNN size. The number of fault locations increases with the increase in the SNN size and, thereby, the number of functional tests required to achieve the desired fault coverage is likely to increase as well. For large SNNs, fault simulation effort may rapidly explode as explained in Section VI-C. For

this reason, the fault space needs to be conservatively reduced by considering the impactful fault locations, i.e., neurons in last layer, saturation neuron faults, and synaptic connections in the last layers. Note, however, that the highest ranked input sample will detect a very high number of faults and in each step, by adding the next highest ranked input sample, the cumulative global fault coverage rises quickly. This behavior was observed in the results in Section VII. In each step, we exclude from the simulation the faults already detected by previous input samples in the ranked order. This way, fault simulation is not exhaustively repeated for every input sample and it can become tractable even for large size SNNs. Note also that step (b) is an one-time effort and can be significant for traditional computing chips as well. Once the functional test set is generated, it will be used as a fixed test program in high-volume production. As a final observation, the effort in step (b) is not related to the dataset size. A large dataset size means more input samples, thus a larger pool of functional tests to choose from. In fact, a larger dataset size may lead to a more compact functional test set.

C. Other metrics for grading functional tests

The metric used for grading functional tests is based on spike-count difference for the first two top classes, i.e., the only relevant aspect of a spike train is its total number of spikes. In principle, any metric that quantifies the distance between two spike trains can be used. Several such metrics have been proposed in the past generalizing the distance measure to include the temporal structure in the spike trains [63]–[67]. For example, in the Victor-Purpura metric [63], the distance is defined as the minimum cost required to transform one spike train into the other via a path of elementary steps. The cost equals the sum of the costs assigned to each of the allowed elementary steps. In our context, the cost is inversely proportional to the score of the functional test. There are two kinds of elementary steps: (a) adding or deleting a spike which is assigned a cost of 1; and (b) shifting in time the occurrence of a single spike by an amount Δt which is assigned a cost of $q * |\Delta t|$, where q is a parameter that expresses the relative sensitivity of the metric to precise timing of spikes. The two extreme cases are $q = 0$ and $q = \infty$ (or very large q). By setting $q = 0$, we recover the spike-count difference metric used in this work. This is because for two spike trains with number of spikes n_1 and $n_2 > n_1$, we can align the first n_1 spikes with zero cost, then transform the second spike train to the first by deleting its last $n_2 - n_1$ spikes with cost $n_2 - n_1$. In the limit $q = \infty$, it is less costly to add or delete spikes than to shift a spike and so the distance between two spike trains becomes the total number of non-synchronous spikes.

X. Conclusions

We presented a method to generate compact functional test sets for SNN hardware accelerators. A fault-agnostic metric is proposed to rank the available samples based on their fault coverage capability without performing any fault injection experiments. The functional test set is generated by performing inference only once for each sample on the nominal network and recording the output neuron spiking activity, i.e., an effort that is spent already during training. Thereafter, fault injection experiments are performed using only the highly ranked samples to compute fault coverage given a fault model. Results collected from three SNNs, one of which is mapped to neuromorphic hardware implemented on an FPGA, show that the proposed method generated highly compact functional test sets that can detect all faults resulting in even the smallest accuracy drop, i.e., one sample is misclassified.

References

[1] 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.

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

[3] 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.

[4] 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.

[5] 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.

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

[7] 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.

[8] 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.

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

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[31] 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.

[32] 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.

[33] 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.

[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] 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.

[36] 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.

[37] 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.

[38] 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.

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

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

[41] 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.

[42] 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.

[43] 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.

[44] A. Amir *et al.*, "A low power, fully event-based gesture recognition system," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017.

[45] 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.

[46] 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.

[47] 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.

[48] 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.

[49] 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.

[50] W. Gerstner, "Time structure of the activity in neural network models," *Phys. Rev. E*, vol. 51, pp. 738–758, 1995.

[51] 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.

[52] 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.

[53] 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.

[54] 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.

[55] 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.

[56] 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.

[57] R. Serrano-Gotarredona, T. Serrano-Gotarredona, A. Acosta-Jimenez, and B. Linares-Barranco, "A neuromorphic cortical-layer microchip for spike-based event processing vision systems," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 53, no. 12, pp. 2548–2566, Dec. 2006.

[58] N. Qiao, H. Mostafa, F. Corradi, M. Osswald, F. Stefanini, D. Sumislawska, and G. Indiveri, "A reconfigurable on-line learning spiking neuromorphic processor comprising 256 neurons and 128K synapses," *Front. Neurosci.*, vol. 9, Apr. 2015, Article 141.

[59] 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.

[60] D. Ma *et al.*, "Darwin: A neuromorphic hardware co-processor based on spiking neural networks," *J. Syst. Archit.*, vol. 77, pp. 43–51, Jun. 2017.

[61] 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.

[62] 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.

[63] J. D. Victor and K. P. Purpura, "Nature and precision of temporal coding in visual cortex: a metric-space analysis," *J. Neurophysiol.*, vol. 76, no. 2, pp. 1310–1326, Aug. 1996.

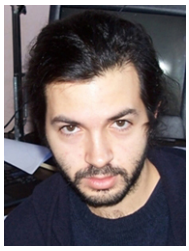
[64] M. C. W. van Rossum, "A novel spike distance," *Neural Comput.*, vol. 13, no. 4, pp. 751–763, Apr. 2001.

[65] T. Kreuz, J. S. Haas, A. Morelli, H. D.I. Abarbanel, and A. Politi, "Measuring spike train synchrony," *J. Neurosci. Methods*, vol. 165, no. 1, pp. 151–161, Sep. 2007.
[66] T. Kreuz, D. Chicharro, C. Houghton, R. G. Andrzejak, and F. Mormann, "Monitoring spike train synchrony," *J. Neurophysiol.*, vol. 109, no. 5, pp. 1457–1472, Mar. 2013.
[67] E. Satuvuori *et al.*, "Measures of spike train synchrony for data with multiple time scales," *J. Neurosci. Methods*, vol. 287, pp. 25–38, Aug. 2017.

Sarah A. El-Sayed received her B.Sc. Degree in Electrical and Electronic Engineering from Minia University, Egypt, in 2009. From 2010 to 2017, she was a teaching/research assistant at the faculty of Engineering at Minia University, Egypt, where she also got her M.Sc. degree in microelectronics in 2015, in collaboration with the National Telecommunications Institute, NTI, in Cairo, Egypt. She received her Ph.D. degree from Sorbonne Université in Paris, France, in 2021. And in 2022, she was a post-doctoral researcher with the Laboratoire d'Informatiques de Paris 6, LIP6, at Sorbonne Université in Paris, France. She is now an associate professor at the faculty of Engineering, Minia University, Egypt. Her research interests include bio-inspired computing, spiking neural networks and analog circuit testing.

Theofilos Spyrou received his diploma of Electrical and Computer Engineering from the Faculty of Engineering of Aristotle University of Thessaloniki, Thessaloniki, Greece, in 2019. He is a PhD candidate at the Laboratoire d'Informatique de Paris 6 (LIP6) at Sorbonne Université. His work is supported by the Sorbonne Center of Artificial Intelligence (SCAI) through fellowship. His PhD dissertation is on the reliability and functional safety of AI Hardware and his research interests include spiking neural networks, neuromorphic computing, and AI hardware accelerators, among others.

Luis A. Camuñas-Mesa received the B.S. degree in electrical engineering, the M.S. degree in microelectronics, and the Ph.D. degree in event-based vision systems from the University of Sevilla, Seville, Spain, in 2003, 2005, and 2010, respectively. From 2004 to 2010, he was with the Sevilla Microelectronics Institute (IMSE-CNM), Spanish Research Council (CSIC), Seville, where he was involved in the VLSI implementation of event-driven AER vision processing systems. In 2006, he was a Visiting Student with the Institute of Neuroinformatics, Zurich, Switzerland. From 2010 to 2013, he was an Associate Postdoctoral Researcher with the Centre for Systems Neuroscience, University of Leicester, Leicester, U.K., where he was involved in olfactory sensing and processing, spike detection and sorting, and simulation of extracellular recordings. Since 2013, he has been with IMSE-CNM, CSIC, where he received the "Juan de la Cierva" Postdoctoral Fellowship. His current research interests include bioinspired circuits and systems, real-time event-based vision sensing and processing chips, neuromorphic stereo vision, and nanoscale memristor-based AER circuits for STDP learning. He is an Associate Editor of the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—II: BRIEF PAPERS.



Haralampos-G. Stratigopoulos (Member, IEEE) received the Diploma degree in electrical and computer engineering from the National Technical University of Athens, Athens, Greece, in 2001, and the Ph.D. degree in electrical engineering from Yale University, New Haven, CT, USA, in 2006. Since 2020 he is Research Director with the French National Center for Scientific Research (CNRS) at the LIP6 Laboratory, Sorbonne Université, Paris, France. Before he was a Researcher with the CNRS at TIMA Laboratory, Université Grenoble Alpes, Grenoble, France, from 2008 to 2015, and at LIP6 Laboratory, Sorbonne Université, Paris, France, from 2015 to 2020. His main research interests include hardware security, neuromorphic computing, and design-for-test for analog, mixed-signal, and RF circuits and systems. He was the General Chair of the 2015 IEEE International Mixed-Signal Testing Workshop (IMSTW) and the 2021 and 2022 AI Hardware: Test, Reliability and Security (AI-TREATS) Workshop and the Program Chair of the 2017 IEEE European Test Symposium (ETS). He has served on the Technical Program Committees for the Design, Automation, and Test in Europe Conference (DATE), Design Automation Conference (DAC), IEEE International Conference on Computer-Aided Design (ICCAD), IEEE European Test Symposium (ETS), IEEE International Test Conference (ITC), IEEE VLSI Test Symposium (VTS), and several others international conferences. He has also served as an Associate Editor for IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS, IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—I:REGULAR PAPERS, IEEE DESIGN AND TEST, and Journal of Electronic Testing: Theory and Applications (Springer).