2025 30th IEEE European Test Symposium (ETS)
- Special Session Paper -

On the Trustworthiness of Spiking Neural Networks and Neuromorphic Systems

Theofilos Spyrou*, Haralampos-G. Stratigopoulos†, Ihsen Alouani§‡, Said Hamdioui*, Anteneh Gebregiorgis*
*Computer Engineering Lab, Delft University of Technology, Delft, The Netherlands
†Sorbonne Université, CNRS, LIP6, Paris, France
‡Queen's University, Belfast, United Kingdom
§CNRS-IEMN, UPHF, INSA Hauts-De-France, France

2025 IEEE European Test Symposium (ETS) | 979-8-3315-9450-3/25/$31.00 ©2025 IEEE | DOI: 10.1109/ETS63895.2025.11049613

***Abstract*—Neuromorphic computing offers a promising solution for realizing energy-efficient and compact Artificial Intelligence (AI) systems. Implemented with Spiking Neural Networks (SNNs), neuromorphic systems can benefit from SNN characteristics, such as event-driven computation, event sparsity, biological plausibility, etc., to achieve high performance and energy efficiency, an aspect vital for the realization of AI at the edge. Although SNNs are biology-inspired structures, their use in mission- and safety-critical applications raises multiple concerns around the trustworthiness of neuromorphic hardware due to various intrinsic and extrinsic reliability and security issues. Hence, adequately studying the dependability of SNNs and neuromorphic hardware accelerators becomes of utmost importance, in order to expose and harden against potential vulnerabilities, so that a reliable and secure operation is ensured. This paper presents an analysis of the dependability and trustworthiness aspects of SNNs and neuromorphic hardware. It outlines potential mitigation and countermeasure strategies to improve the reliability, testability, and security aspects of SNN hardware and ensure its trustworthy deployment in critical application domains.**

***Index Terms*—Neuromorphic Computing, Spiking Neural Networks, Artificial Intelligence, Reliability, Test, Security.**

I. Introduction

Deep Neural Networks (DNNs), that is, the backbone of modern Artificial Intelligence (AI) applications, have reached a tremendous degree of complexity, typically comprising a multitude of layers [1]. Computationally speaking, the vast amount of operations performed by DNN models translates into a high demand for hardware and energy resources [2]. A great challenge therefore arises for resource-limited edge AI applications, where the widely used general-purpose AI hardware accelerators, such as GPUs, are a direct no-go [3].

In an effort to increase energy efficiency, an alternative is to employ Application-Specific Integrated Circuits (ASICs) [4], [5], which can holistically improve the cost-energy-performance trade-off. A notable example of very low-power ASIC AI hardware acceleration is neuromorphic computing. Implemented with Spiking Neural Networks (SNNs), neuromorphic accelerators [6]–[10] can benefit from the biology-inspired characteristics of SNNs, such as asynchronous operation and event-driven processing, to achieve significant enhancements in energy consumption while maintaining high performance.

The widespread adoption of AI, though, comes with concerns about its trustworthiness from both a reliability and security perspective. SNNs are no exception to this, although they are known to inherit to an extent the fault tolerance capabilities of the human brain, thus, their promising properties require a proactive analysis to ensure proper functionality. From a hardware perspective, neuromorphic architectures remain prone to hardware-level faults and sophisticated attacks targeting these systems, posing a threat to network performance and putting safety and privacy at risk.

In this work, we explore the dependability characteristics of SNNs and neuromorphic systems. Knowing what can go wrong is the first step towards a trustworthy neuromorphic system. There are several factors that can hinder operation, such as aging and over-stressing of a chip, cosmic radiation, Single Event Upsets (SEUs), etc. Creating the corresponding fault models enables large-scale fault injection experiments to assess the reliability of neuromorphic implementations. The derived results are then used to expose vulnerable parts of the system and contribute to the design of testing mechanisms to monitor these regions for malfunctions. In the event of a faulty operation, fault mitigation mechanisms are triggered, which contributes in this way to enhanced fault tolerance.

Regarding security and privacy aspects of SNNs, their unique computational properties—such as event-driven processing, temporal dynamics, and non-differentiable activation functions—suggest that they may offer inherent defenses against certain classes of attacks. We conduct an empirical study to systematically evaluate the adversarial robustness and privacy leakage risks of SNNs in comparison to their ANN counterparts. We assess the behavior of both models under common adversarial attacks such as FGSM and PGD and evaluate their exposure to Membership Inference Attacks (MIAs) across a range of benchmark datasets and architectures. Our results suggest that neuromorphic models may contribute not only to energy efficiency but also to the design of inherently more secure and privacy-preserving machine learning systems.

This work was partially funded by: the European Network of Excellence dAIEDGE under Grant Agreement N^{o} 101120726, the Dutch Organization for Scientific Research (NWO) under grant number KICH1.ST04.22.021 for the project Self-Healing Neuromorphic Systems (SNS), and the EdgeAI project, supported by the Chips Joint Undertaking and its members under grant agreement No 101097300.

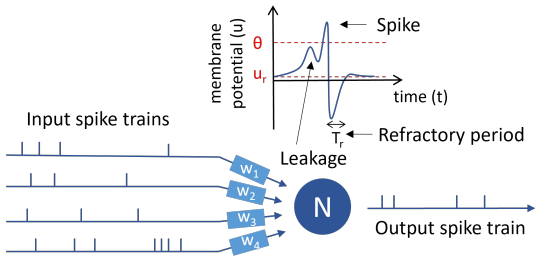

Fig. 1. Operation of a Leaky Integrate and Fire spiking neuron.

The remainder of the paper is structured as follows. Section II introduces the theoretical background on SNNs and neuromorphic hardware. Section III discusses the reliability assessment of neuromorphic hardware accelerators. Section IV presents neuromorphic chip testing techniques. Section V describes fault tolerance strategies employed in neuromorphic designs followed by the discussion on the security and privacy aspects of SNNs in Section VI. Finally, Section VII concludes the paper.

II. Background Information

A. Spiking Neural Networks

The undeniable computational and energy-efficient superiority of biological neural systems sets the foundations of the third generation of ANNs, namely Spiking Neural Networks (SNNs) [11]. Resembling their biological counterparts, SNNs incorporate time within their calculations and information flows in the form of *spikes*, i.e., electrical pulses. Spiking neurons use *spike trains*, i.e., sequences of spikes, to encode and propagate temporal information among each other.

Throughout the development of SNNs, a variety of spiking neuron models have been proposed [12]–[14]. One of the least computationally complex spiking neuron models, and hence hardware-friendly [15], is the ubiquitous Integrate and Fire (I&F) neuron. In addition, a generalization of I&F is the Spike Response Model (SRM), which describes the spiking activity as the neuron's responses to the incoming spiking activity and the previously fired spikes [16].

Fig. 1 shows the operation of a Leaky I&F (LI&F) neuron; a variation of I&F. At any given moment t, the *membrane potential* $u(t)$ of the neuron denotes its current state, which is equal to u_r while at rest. The input spike trains that arrive through neuron's incoming *synapses* stimulate the membrane potential proportionally to the strength of each *synaptic weight* w_i connecting it to neuron i of the previous layer. Once the membrane potential reaches a predefined *threshold* θ, the neuron *fires* a spike at its output, which is propagated to subsequent neurons. The neuron is then reset to its resting state and pauses integrating new input spikes for a time equal to the *refractory period* T_r. During input inactivity, the *leakage* mechanism is periodically applied and gradually reduces the membrane potential, pulling it towards the resting state.

The non-differentiable nature of spikes, i.e., delta functions, makes training large SNN models a challenge [17]–[19], as traditional learning techniques, such as the backpropagation algorithm, are not directly applicable. To overcome this challenge, several learning schemes have been developed [20], [21], including shadow training, that is, training an ANN and converting it to SNN, evolutional algorithms, and spike-based adaptations of backpropagation. There also exist biology-inspired techniques, such as Hebbian learning and Spike Timing-Dependent Plasticity (STDP), which are aligned with biological plasticity mechanisms and enable temporal learning.

Regarding the input of SNNs, it needs to be in a spiking form as well, meaning that the network is fed with continuous time series instead of static frames of data. A way to achieve this is to encode the original static information in the frequency or in the exact timings of spikes. Another way is to capture information with neuromorphic devices, such as the Dynamic Vision Sensor (DVS). A DVS resembles the retina of the human eye and generates spikes based on brightness changes in its pixel neurons.

B. Neuromorphic hardware

From a hardware perspective, SNNs are highly distributed systems that operate asynchronously on an event-driven basis. Spiking activity tends to be sparse, as neurons remain silent for most of the time and fire only when necessary, in contrast to the cycle-by-cycle activation in conventional ANNs. Considering modern neuromorphic architectures that incorporate millions of neurons and synapses [6]–[10], the above characteristics of SNNs make neuromorphic computing an attractive technology, whose main advantages are as follows:

- *Energy efficiency:* Spiking neurons are mostly idle and are activated only when a spike reaches their input, which, in combination with the sparsity of spikes throughout the network, leads to great power savings.
- *Computation speed:* SNNs are capable of processing a substantial amount of data using a relatively small number of spikes. The first output spikes start emerging as the input events still flow in. Event-driven sensing and processing allow input and output event flows to be (in practice) simultaneous [22].
- *Noise robustness:* The shape of a spike carries no information and its scope exists only up to the next layer's neurons, which fire their own spikes. Therefore, there is a regeneration of signals in every neuron [23], making SNNs inherently robust to noise.

Within the highly parallel and distributed architectures of neuromorphic systems, connectivity and communication among neurons become an issue. To deal with this and overcome potential physical limitations, e.g., finite synaptic connections per neuron, communication protocols, such as Address Event Representation (AER), are employed. The AER protocol treats spikes as events with information on the timing of firing and the location of the neuron of origin or destination. This eliminates the need for strongly defined synaptic connections and allows efficient allocation of resources when and where needed.

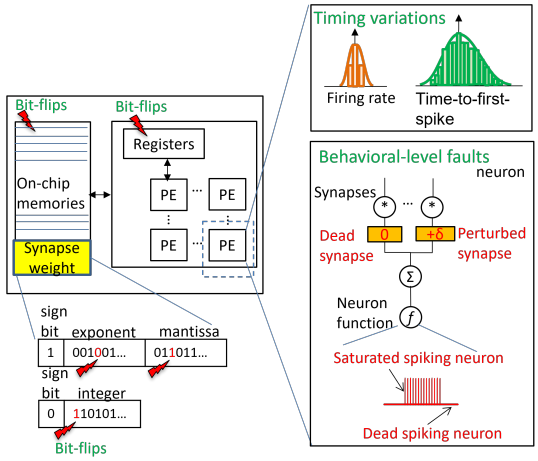

Fig. 2. Fault modeling.

III. Reliability Assessment of Neuromorphic Hardware

Reliability failure is the inability of a device to perform its proper functionality throughout its intended lifetime. Such failure can have significant consequences depending on the criticality of the targeted application. SNN hardware reliability failure can be rooted in different sources or failure mechanisms such as process variation, deviation of device parameters from intended values due to manufacturing imperfections, device degradation/wear-out, and environmental factors. Reliability failure mechanisms can be broadly classified as time-zero and time-dependent mechanisms.

Time-zero reliability mechanisms: Time-zero reliability failures are issues that occur at time $t = 0$. These issues are caused by imperfections in the fabrication process. Due to these varying conditions during fabrication, the fabricated device/chip parameters differ from their intended design specifications. Time-zero reliability failures such as local process variation, line edge roughness, and dopant fluctuations can be either local, i.e., within single die/chip, or global, i.e., inter-die (die-to-die or chip-to-chip).

Time-dependent reliability mechanisms: Time-dependent mechanisms include those that occur during the operational lifetime of the device, i.e., $t > 0$. They may be intrinsic, i.e., caused by mechanisms within the device itself, or extrinsic, i.e., caused by external environmental factors.

In order to address time-zero and time-dependent reliability issues in SNN neuromorphic systems, application-aware fault modeling and simulation are crucial. The remainder of this section presents the fault modeling and fault injection experiments for SNN hardware along with potential mitigation strategies.

A. Fault modeling

Fault models for SNN at behavioral level focus on the neuron and synapse functionality, assuming that neurons and synapses can fail independently. These models aim to represent the impact of low-level hardware faults on the network's mathematical behavior. An illustration of main fault models is shown in Fig. 2.

1) Neurons: In [24], a spiking neuron was subjected to transistor-level fault simulation, accounting for defects such as short- and open-circuits, as well as process variations through a Monte Carlo simulation. The primary faulty behaviors identified include:

- *Saturated neuron*: The neuron continuously generates spikes, even when no excitatory input spikes are present.
- *Dead neuron*: The neuron fails to produce output spikes, even with incoming excitatory spikes at its input.
- *Spike timing variations*: The neuron fires, but the timing of spikes is altered, affecting parameters such as time-to-first-spike or firing frequency.

This neuron fault model is widely used in studies on SNN testing and fault tolerance [25], often with variations to capture intermediate fault behaviors, such as hard-to-spike and easy-to-spike states [26]. For SNNs trained with spike-timing-dependent plasticity (STDP), additional neuron fault models can be applied, as these networks exhibit distinct architectural and operational features, such as local learning rules [27].

2) Synapses: Synapse faults are modeled as changes in the synaptic weight:

- *Dead synapse*: The synapse becomes stuck-off, failing to process any input spikes.
- *Perturbed weight*: Weights are typically stored as digital words in on-chip memory. A hardware-related fault may cause bit-flips, leading to weight perturbations that can range from small variations to significant deviations beyond the weight distribution observed after training. At the extreme the synapse may become stuck-on, i.e., for a positive (negative) saturated synapse an incoming spike translates to a large positive (negative) jump in the membrane potential of a post-synaptic neuron which causes the neuron to immediately (never) fire. For memristor-based implementations, where a memristor represents a synapse, the fault can be modeled as a memristor resistance fault, i.e., a memristor being stuck in either a high or low resistance state [28].

These faults are incorporated into the computation flow of the SNN software framework. For instance, neuron faults can be introduced by directly altering their output spike train or by modifying internal parameters like the membrane potential threshold. Synaptic faults can be simulated by directly modifying the weights, with bit-flip errors being handled through quantization, bit-flipping, and a backward conversion to real values. Nowadays, two open-source automated fault injection frameworks exist, namely *SpikingJet* [29] built on top of SnnTorch [20] and *SpikeFI* [30] built on top of SLAYER [21].

B. Fault injection experiments in software and hardware

Fault simulations in hardware descriptions of neuromorphic architectures, such as at the gate level and RTL, can be a very time-consuming procedure. Moreover, assessing the fault impact on the inference accuracy of the network requires evaluation on large validation sets, thus, simulation time can easily explode. For this, performing fault injection experiments

at the software level with the aforementioned frameworks based on behavioral-level fault models reflecting hardware properties can minimize simulation time, while assessing many different fault scenarios [30].

The work in [31] applies this technique to assess the reliability of SNNs in the presence of neuron faults measured on the impact on the classification accuracy. The experiments carried out examine cases of single neurons failing under various faults after training the network or multiple ones before training. Results show that saturated neurons have a lethal impact on the network's performance, while dead neurons and spike timing variations have a less severe effect. Moreover, convolutional layers present a higher degree of resilience than fully connected ones, while the output layer is always the most vulnerable, drastically degrading performance when faulty. In the case of faults occurring prior to training, SNNs are capable of tolerating high fault rates.

Regarding robustness under synapse faults, the FI experiments conducted in [32] report a less severe impact compared to neuron faults. Dead and negatively saturated synapse faults affect performance slightly, as they suppress the output of the pre-synaptic neuron towards the post-synaptic one, making it harder for the latter's membrane potential to reach the firing threshold. Positively saturated synapses, on the other hand, cause post-synaptic neurons to fire more easily and, therefore, create a denser spiking activity that propagates the effect. Besides these extreme faults, the impact of memory bit-flips occurring in synaptic weights is studied in [30]. Results indicate that the Most Significant Bits (MSBs) have the most detrimental effect on performance, as they cause a great disturbance in the value of the defective weight.

Although experimenting on higher abstraction levels allows flexibility and saves time, the particularities of the system orchestrating the neuromorphic architecture are not fully taken into account. In [33], fault injection experiments are conducted on actual neuromorphic hardware implemented on an FPGA board configured by an embedded processor. The fault model is composed of bit-flips in the memories storing the different network parameters, i.e., kernel weights, routing, neuron's threshold and leakage, etc. The results of the analysis pinpoint the critical parts of the accelerator, which may be other than neurons and synapses. The network performance is hindered by faults occurring at other components as well, e.g., routing, which may lead to a completely different network structure if faulty.

C. Memristor non-ideality impact analysis and mitigation for CIM-based SNN accelerators

Memristor-based Computing-In-Memory (CIM) is a computing paradigm in which the computation (i.e. execution) of an operation is performed within the memory where the data reside. Implemented with emerging memristive non-volatile devices, such as Resistive Random Access Memory (RRAM), memristor-based CIM architectures can circumvent the costly data movement of Von-Neumann systems, thus significantly improving energy efficiency and latency while

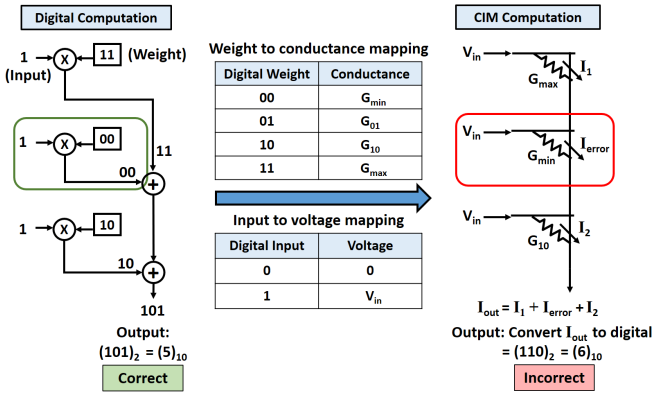

Fig. 3. Illustration of non-zero G_{min} error.

offering superior performance [34]. The structure of CIM crossbar arrays makes them ideal for Multiply & Accumulate (MAC) operations, which are fundamental in SNN. Despite the numerous advantages of memristor-based CIM architectures, their practical implementation often suffers from non-idealities and manufacturing defects, such as conductance drift, non-zero G_{min}, read disturb, etc. [35], [36].

1) Impact of non-zero G_{min} error on CIM accuracy: In the digital domain, multiplying any non-zero input with a zero weight must result in a zero output. However, when such computation is mapped to memristors as shown in Fig. 3, a non-zero output current is produced when a non-zero input voltage is applied to a memristor with G_{min} conductance which represents digital zero. This phenomenon is known as non-zero G_{min} error which causes a functional mismatch between the expected digital value and the actual memristive computation result [37].

2) Mitigating the impact of non-zero G_{min}: Mapping SNN to CIM crossbar often involves division of synaptic weights into smaller chunks, called slices, which are mapped to conductances. During map operation, the column current resulting from the interaction of inputs with sliced conductances is an output that represents the weighted synaptic accumulation of an SNN neuron. However, such weighted synaptic accumulation suffers from non-zero minimum conductance (G_{min}) error, which degrades the accuracy of the CIM-based SNN.

In [37], an Unbalanced Bit-Slicing (UBS) scheme is proposed that changes the way in which synaptic weights are mapped to the CIM crossbar to mitigate the impact of non-zero G_{min} error. The proposed UBS scheme provides high sensing margin for the MSBs by using a memristor with n-bit maximum capacity as an m-bit memory-cell (slice), where $m<n$. This provides sufficient sensing margin to the MSB column output and makes them immune to non-zero G_{min} error as shown in Fig. 4. The improvement of UBS over conventional bit slicing is illustrated by the increase in the sensing margin at the output of the Analog-to-Digital Converter (ADC). For the balanced bit-slicing approach, since the MSB stores two bits, the result is expressed with three bits, while for UBS only two bits are needed at the output. This is illustrated by the output sensing margins shown in Fig. 4.

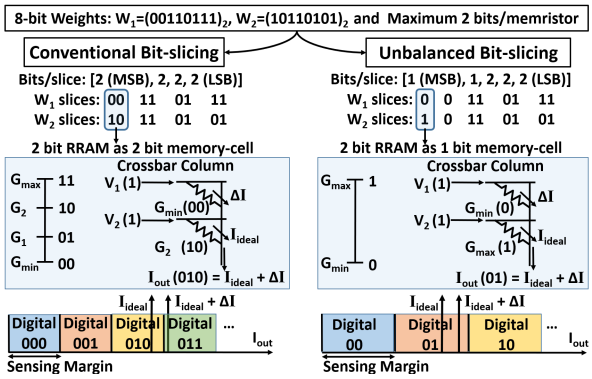

Fig. 4. Impact of sensing margin on bit-slicing schemes [37].

This suffices for good accuracy due to the robustness of neural networks to minor computational fluctuations, i.e. errors in the Least Significant Bits (LSBs). The simulation results show that the unbalanced bit-slicing scheme achieves up to 8.8× and 1.8× accuracy compared to state-of-the-art CIM architectures for single-bit memristors and two-bit memristors, respectively, at reasonable energy overheads arising due to extra columns.

IV. Neuromorphic Chip Testing

Testing is conducted to identify hardware failures immediately after manufacturing (known as post-manufacturing testing) and during operation in the field (referred to as periodic online testing or concurrent error detection).

Neuromorphic chips and, in general, AI hardware accelerators, demand specialized testing methodologies that go beyond traditional structural and functional tests [38]. The reason is that they have a complex computation and data flow, they feature extensive on-chip memory and complex interconnect networks for data movement, and include dynamic reconfiguration capabilities. Additionally, an application running on the chip exhibits a high level of inherent fault tolerance. Many hardware faults turn out to be benign, either because they impact unused resources or because the computation can tolerate a certain degree of approximation.

This suggests the need for a more functional testing approach. A simple method would be to load an SNN model onto the hardware and perform a full inference. However, this approach presents two challenges. First, full inference is time-consuming, whereas chip testing in high-volume production must be completed within a few seconds. Second, since the SNN model does not utilize all hardware resources, it does not ensure that an end-user deploying a different SNN model will experience an error-free operation.

For years, chip testing has leveraged fault models to improve efficiency. These models, which accurately represent hardware failures at the behavioral level, enable the generation of compact input test patterns that achieve nearly 100% fault coverage. However, for AI chips, fault injection should be approached differently, aligning more closely with AI workloads.

Furthermore, to demonstrate the fault coverage of test patterns, a dedicated fault simulation framework is required. This framework typically operates on top of an SNN software framework, used for designing, simulating, and training SNNs, enabling faster analysis compared to hardware-based fault simulation, which is excessively time-consuming [29], [30] .

A. Automatic test pattern generation

Taking as an example a vision application, the input to an SNN can be interpreted as a time series of frames, each of size $W \times H$, arriving at intervals of T_f. Each frame is composed of pixels that may or may not generate a spike. Consequently, at each pixel location, a spike train propagates over time, feeding into a neuron in the input layer. The pixel state at each time step can be represented as a binary event $\{0, 1\}$, where 1 indicates a spike. Thus, the input can be modeled as a binary tensor of dimensions $W \times H \times (N_f \times T_f)$, where N_f represents the number of frames and $N_f \times T_f$ denotes the total input duration.

Automatic test pattern generation (ATPG) is the process of identifying a set of input patterns that achieve maximum fault coverage. This means generating input patterns that activate all possible faults and propagate their effects to the output, ensuring that the response of a faulty network can be distinguished from the fault-free nominal response.

In [32], it is suggested to use samples from the dataset that are correctly predicted but with low confidence. The rationale is that the footprint of these samples lies close to the hyper-dimensional classification boundary, and a fault is likely to cause this footprint to jump over the boundary and change the classification label, thus the fault will be detected. The algorithm starts by ranking the input samples based on the confidence in their prediction. For example, considering rate coding and a classification task, the confidence is measured by the output spike count difference of the two neurons producing the largest number of spikes, i.e., the neurons corresponding to the two top classes. Then, starting from the top of the list, fault simulation is performed adding the input to the test list if it detects previously undetected faults and dropping the newly detected faults from the fault list.

In [26], it is suggested to enhance the test pattern set with adversarial examples, which are generated from dataset samples by adding small perturbations such that the network's decision is fooled. Adversarial examples also lie close to the classification boundary, thus they are prone to misclassification if a fault occurs and, in this way, they can detect it. The test generation algorithm starts by picking a fault from the fault list and examining if any of the available inputs in the dataset is capable of detecting it. If no such input is found, 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 input or adversarial example is found that detects the fault, then this successful test is tried out on all faults. It is placed in the test list and the detected faults by this test are dropped from the fault list. The algorithm reiterates targeting the next undetected fault.

In [39], the configurability of the hardware is leveraged to map various models onto it. Multiple test configurations are explored, each corresponding to a different model. The

test algorithm starts by picking a fault and training a random test configuration with random inputs such that there is a maximum difference between the outputs of the nominal and faulty models. Note that in this case the random inputs do not need to have target labels. If this step succeeds, then in the next step untried faults are picked and an adversarial sample is generated to detect them. Adversarial samples that succeed are added to the test list together with their corresponding test configuration and detected faults are dropped from the fault list. The algorithm reiterates starting with a new random test configuration and inputs until all faults are detected.

In [40], the test generation algorithm applies random combinations of test configurations and inputs. Test configuration and inputs pairs that detect a previously undetected fault are marked as candidate pairs. The algorithm iterates until all faults are detected. Then, test compaction is performed. Starting from the last selected candidate pair, a full fault simulation is performed dropping candidate pairs if fault coverage does not improve.

A common feature of the methods in [26], [32], [39], [40] is that the resulting test set consists of tens to hundreds of tests to ensure high fault coverage, with each test taking as long as a single inference. In [39], [40], different test configurations, i.e., models, must be loaded onto the chip, further increasing test application time. Additionally, test generation time is a concern, as the test set is verified by repetitive fault simulations. For large networks, the fault model size grows significantly, potentially limiting the scalability of these methods.

A different approach is proposed in [41], where the input is trained to achieve high fault coverage. The training process utilizes backpropagation in the spike domain using the SNN software and is shown to converge rapidly, even for large networks, to a short-duration input comparable to a few dataset samples, while ensuring high fault coverage. Unlike the previous methods, fault simulation is circumvented during test pattern generation. The fault coverage metric is replaced with objective functions driving the optimization, focusing on activating all neurons and enhancing the likelihood that modified local spiking activity, influenced by faults, propagates effectively to the output, ultimately leading to fault detection.

As a final note, a compact test set with high fault coverage can be stored in on-chip memory and periodically replayed in idle times for online testing.

B. Regression-based testing

In [42], a regression-based approach is proposed. A subset of dataset samples are used as test set. A signature is collected after performing inference for the test set. For example, for the considered time-to-first-spike encoded SNN, the signature is defined as the vector of time stamps of first spikes at the output neurons, where if a neuron does not spike it is assigned the maximum time allocated for inference. The regression model is trained using SNN instances with perturbed weights to map the signature to the accuracy that would have been measured if inference on the complete dataset was performed. During test time, the same subset of dataset samples is used and the regression model is used to predict the accuracy. The method works only for weight perturbation and other faults should be ruled out before applying it, otherwise the predicted accuracy is likely to be misleading.

C. Built-in self-test

Built-in self-test (BIST) involves integrating test signal generators and/or monitors on-chip to simplify testing, allowing for reuse in the field for online testing and concurrent error detection.

The initial approach involved a self-testable implementation of a spiking neuron circuit [43]. The neuron model, adapted from [44], replicates the behavior of a biological neuron, generating multiple classes of spiking firing patterns. The BIST approach consists of a ramp generator block that controls the bias voltages of the neuron to produce all different firing patterns at the output in the fault-free case, and a digital block that checks if any of these patterns is missing, in which case it flags a fault detection.

In [31], [45], BIST monitors are proposed to detect the neuron saturation fault, which is identified as the most critical for network operation. In [31], the monitor consists of a small counter that tracks the number of output neuron spikes, resetting whenever a new input spike arrives. If excessive spikes are generated without input activity, the counter overflows, triggering an error flag. In [45], neuron saturation is detected when the membrane potential remains above the threshold for more than two clock cycles.

In [46], a classifier is added on-chip and is trained to map a signature of the SNN to a direct pass or fail decision, where the signature is the vector of the spike counts at the neuron level or accumulated at the feature map level for a convolutional SNN. The classifier is one-class trained with a dataset that is generated on the nominal SNN by presenting the available dataset.

V. Designing for Fault Tolerance

1) Fault-aware training: One approach to enhancing fault tolerance is fault-aware training, where the training process is adapted to improve the network's robustness against faults that may occur during the chip's lifetime. As noted in [31], dropout [47] inherently serves as a fault-tolerance mechanism. During training with dropout, certain neurons are randomly deactivated in each forward pass setting the weights of their associated synapses to zero. This prevents the network from relying too much on specific neurons, encouraging the model to learn more robust and general features. The study in [31] demonstrated that dropout naturally trains the network to cope with dead neuron faults, as it essentially simulates the presence of dead neurons during training.

While dropout offers a natural form of fault tolerance, a more intrusive approach involves quantization-aware training combined with random bit-flips in synaptic weights [48]–[50]. This technique exposes the network to bit-level faults during training, allowing it to adapt and build resilience against

potential bit-flips caused by radiation, voltage variations, or temperature fluctuations over the chip's lifetime.

A third approach introduces fault tolerance directly into the training objective by modifying the loss function [51]. Specifically, during training, each network instance undergoes multiple fault injections, and the model's accuracy is evaluated across these faulty variants. The resulting loss is then computed as a weighted combination of the baseline (fault-free) accuracy and the average accuracy under faults, guiding the network to maintain performance even in the presence of errors.

2) Component hardening: A traditional strategy at hardware design time is to reinforce vulnerable hardware components by altering their physical layout to improve resilience against radiation-induced faults. In [52], a radiation-hardened design for spiking neurons is proposed to enhance their fault-tolerance characteristics.

3) Triple modular redundancy: Triple Modular Redundancy (TMR) is a traditional method where the system is replicated three times, and a voting mechanism determines the correct response, under the assumption that no two subsystems can fail at the same time. While triplicating the entire SNN hardware accelerator leads to significant cost, selective hardening of critical layers—such as the last layer, which has been shown to be the most critical and requires robust fault-tolerance protection [31]—can be a more cost-effective approach.

4) Memory fault mitigation: The memory storing synaptic weights can be safeguarded using Error Correcting Code (ECC). However, the significant memory size in AI hardware accelerators adds to the cost of implementing ECC, primarily due to increased memory usage, higher inference latency, added hardware complexity, and greater power consumption.

An alternative approach is presented in [48], where memory testing is initially conducted to generate a memory fault map. Subsequently, bit shuffling is applied to prioritize placing the Most Significant Bits (MSBs) of the weights in the non-faulty memory cells.

5) Fault masking: Since dropout can mitigate the impact of dead neurons and saturated faults are the most critical, [31] suggests using the BIST neuron saturation monitor described in Section IV-C. When a neuron saturation is detected, the neuron can be simply disabled, as this will have little to no effect on accuracy due to the training with dropout. For synapses, it has been observed that large positive values can significantly affect accuracy. Therefore, if the weight increases or an abnormal current flows into the post-synaptic neuron, one approach is to set the weight to zero [45], [53]. In the worst case, this would make the post-synaptic neuron dead.

6) Pruning: One potential approach involves testing to detect faulty processing elements, e.g. neurons, followed by pruning these neurons and their associated synapses from the model, retraining it, and deploying an updated version of the model that avoids utilizing the defective resources [50].

7) Astrocyte neural networks: In [54]–[58], 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. 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.

8) Neuron adaptation: In [59], fault tolerance is implemented through redundant synapses per neuron combined with adaptive neuron behavior. Specifically, each neuron tracks the total input current from all its synapses over a given time window. If a sudden or abnormal change is detected, the neuron adjusts its threshold or operating frequency to maintain a consistent firing rate. On the other hand, variations in the neuron's threshold due to faults can be mitigated by modifying the number of inference time steps [53].

9) Re-learning: In [60], a high-level, biologically-inspired model of the brain's cortical structure is proposed, designed to carry out feed-forward sensory processing and automatic abstraction of visual inputs. The model is trained through Hebbian learning by repeatedly presenting input samples. When neurons become dead, their roles can be assumed by neighboring neurons, enabling the network to re-learn and recover functionality. In contrast, saturated neurons can significantly impair performance. Once identified, these neurons are disabled, and the network undergoes re-training. Detection involves interrupting operation and re-evaluating the response using a TMR voting scheme that leverages built-in redundancy.

10) Memristor crossbar arrays: An appealing hardware implementation involves executing neural network layers on a memristor crossbar, where each memristor represents a synapse. However, current memristor technologies suffer from limited yield and endurance. The lifespan of a memristor is influenced by several factors, including: (a) its programmed resistance state, determined by the model mapped onto the crossbar; (b) the amount of current it carries during inference, which varies with its crossbar location and the computational workload; and (c) the importance of the corresponding synapse to the model's performance. To address this, the works in [61], [62] propose frameworks for endurance-aware synapse-to-memristor mapping, ensuring that critical synapses are assigned to memristors with higher endurance.

VI. Security and Privacy Aspects of SNNs

As SNNs emerge as a promising paradigm for energy-efficient and biologically inspired computation, understanding their security and privacy properties has become increasingly important. While significant research has focused on the performance and efficiency of SNNs, relatively little attention has been given to how these models withstand adversarial threats and protect sensitive information. Given the growing deployment of machine learning systems in critical domains such as healthcare, finance, and autonomous systems, it is essential to assess the vulnerabilities and resilience of SNNs to both adversarial attacks and privacy breaches. This section explores the current understanding of the security and privacy

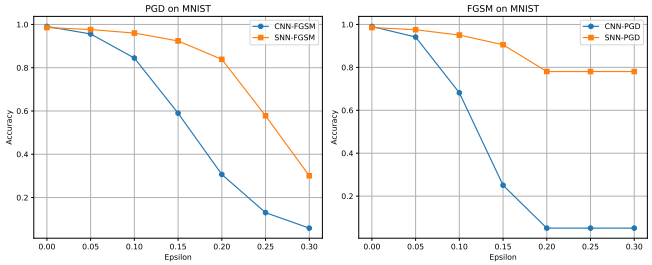

Fig. 5. Accuracy under FGSM and PGD attacks for ANN and SNN on MNIST dataset.

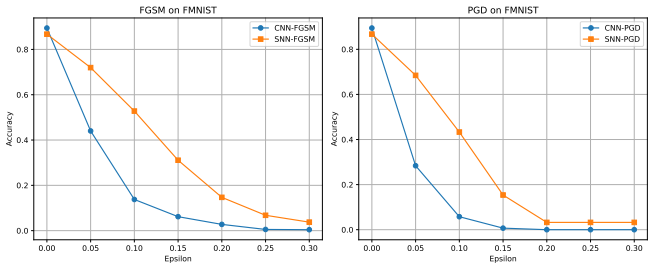

Fig. 6. Accuracy under FGSM and PGD attacks for ANN and SNN on Fashion-MNIST dataset.

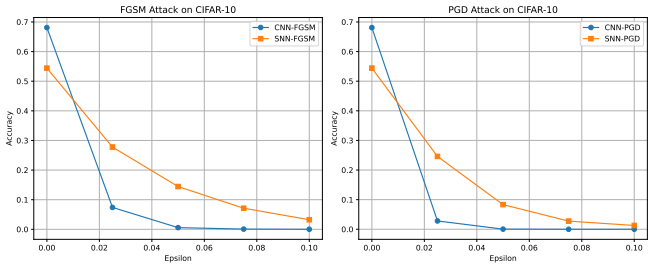

Fig. 7. Accuracy under FGSM and PGD attacks for ANN and SNN on CIFAR-10 dataset.

characteristics of SNNs, highlighting recent empirical findings that differentiate them from their ANN counterparts.

A. Robustness to adversarial noise

The trustworthiness of Machine Learning (ML) models is threatened by adversarial attacks [63]–[65], where a malicious actor can jeopardize models' integrity by leveraging a low magnitude imperceptible input perturbation. In safety-critical applications, these attacks can cause catastrophic consequences.

Experimental Results. We conducted experiments evaluating adversarial robustness of ANNs and SNNs across three standard image classification benchmarks: MNIST, Fashion-MNIST (FMNIST), and CIFAR-10. For each dataset, a convolutional ANN and an equivalent convolutional SNN using LI&F neurons with surrogate gradient learning were trained. Adversarial robustness was assessed using Fast Gradient Sign Method (FGSM) and Projected Gradient Descent (PGD) attacks, with varying perturbation strength (epsilon). Figs. 5, 6, and 7 illustrate accuracy degradation under these attacks.

Across all datasets, we observed that while expected accuracy decreases progressively as the attack strength (epsilon) increases, SNNs consistently demonstrated superior robustness compared to the ANN, maintaining higher accuracy under both FGSM and PGD attacks. This robustness gap is particularly pronounced for more challenging datasets, such as CIFAR-10, suggesting inherent resilience in spiking neuron architectures to adversarial perturbations. These results highlight the potential of SNNs in developing secure machine learning systems resilient against adversarial attacks.

B. Inherent privacy-preserving properties of SNNs?

As ML systems become more sophisticated and widespread, individuals are increasingly relying on these systems, entrusting them with personal and professional data. Consequently, the risk of sensitive information exposure is growing significantly in multiple sectors. It is particularly alarming in fields such as healthcare, where the confidentiality of patient data is extremely sensitive, as a breach could result in severe personal and financial implications, which can affect patient care and institutional credibility [66], [67]. This has led to the development of various privacy attacks targeting ML models to extract sensitive information, including Membership Inference Attacks (MIAs) [68]. In MIAs, an adversary seeks to ascertain if a specific data point was part of the dataset used to train the model. This intrusion risks exposing classified information about individuals in the training dataset, potentially compromising personal data confidentiality [69].

Several works have been arguing the potential of inherent privacy-preserving properties in SNNs, particularly with regards to their ANNs' counterparts [70]. The intuition behind this hypothesis is based on two key aspects:

- First, the discrete event-based data representation and the non-differentiable nature of SNNs' operators such as I&F and LI&F may weaken the correlation between the model and individual data points. This makes it more challenging for a potential adversary to identify the membership of a particular data point in the training set.
- Secondly, the unique encoding mechanisms employed by SNNs introduce an additional layer of stochasticity [71] and variability to the data representation. This added complexity can make it more difficult for an attacker to infer unique characteristics of individual data points, thereby making them more indistinguishable.

Experimental Setup. To assess the privacy-preserving properties of SNNs compared to ANNs, we trained both network types on a diverse set of benchmark datasets: MNIST, Fashion-MNIST (FMNIST), CIFAR-10, CIFAR-100, Iris, and Breast Cancer. For each dataset, we used three representative architectures: a lightweight Baseline CNN, ResNet-18, and VGG-16. All networks were trained using standard cross-entropy loss until convergence, and SNNs were implemented using LI&F neurons with surrogate gradient descent for backpropagation. Spiking models used a fixed time window (T=10) for temporal integration. Identical training pipelines and hyperparameters (e.g., optimizer, learning rate, batch size) were used for both ANN and SNN variants to ensure a fair comparison.

Privacy Evaluation and Discussion. The MIA attack performance is measured using the AUC metric, where higher

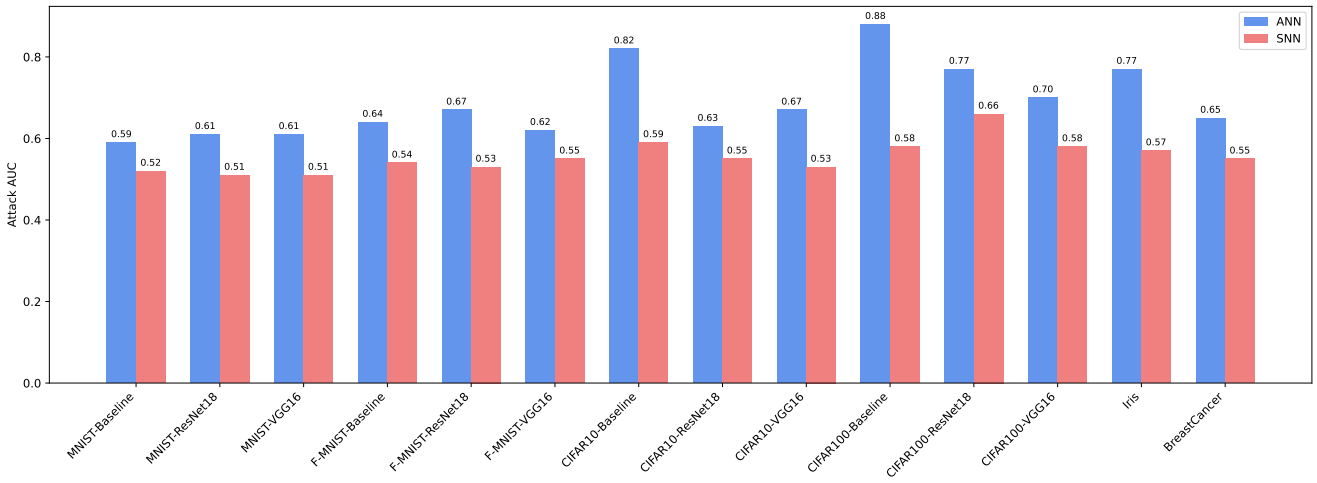

Fig. 8. Comparison of ANN and SNN Resilience to MIA Across Datasets and Architectures.

values indicate stronger attack success and, thus, lower model privacy. The results are summarized in Fig. 8.

SNNs consistently achieve lower attack AUC scores across all datasets and model architectures compared to their ANN counterparts. This trend is especially evident in complex datasets such as CIFAR-10 and CIFAR-100, where the gap between ANN and SNN attack AUC reaches up to 0.25 in some configurations. For example, while ANN models on CIFAR-100 (Baseline) reach an attack AUC of 0.88, their SNN counterparts reduce it significantly to 0.58. Similar trends are observed across other architectures and datasets, including FMNIST and MNIST.

These findings empirically reinforce the hypothesis that the inherent spiking behavior and stochastic representations in SNNs contribute to making individual training samples less distinguishable, thereby mitigating privacy risks. Importantly, this suggests that SNNs are not only competitive in terms of robustness but also provide promising advantages in defending against privacy leakage, making them a compelling choice for privacy-sensitive applications.

Discussion. Our experimental results across multiple datasets and threat models demonstrate that SNNs, as a class of neuromorphic models, offer tangible advantages over traditional ANNs in both adversarial robustness and data privacy. From a security perspective, SNNs exhibit significantly reduced accuracy degradation under attacks such as FGSM and PGD, likely due to their sparse, temporal, and non-linear dynamics that dampen the effectiveness of perturbations. In terms of privacy, SNNs consistently achieve lower membership inference attack success rates across all tested architectures and datasets. This supports the idea that the discontinuous and event-driven computation in SNNs limit the influence of individual data points, thereby reducing their memorization and leakage risks.

These observed advantages point to fundamental architectural properties in SNNs that serve as natural defenses in sensitive and adversarial environments. The combination of non-differentiable neuron behavior and encoding-induced variability makes SNNs particularly suitable for deployment in privacy-critical and resource-constrained settings such as edge computing and neuromorphic hardware. However, while these results are promising, SNNs are not immune to more advanced or adaptive adversarial attack and MIA strategies. On the other hand, it is important to consider additional threats such as model stealing via side-channels [72], backdoor attacks [73], hardware Trojan attacks [74], and fault injection attacks [75]. Future work should explore the vulnerability of SNNs to white-box attacks, encoding-aware adversaries, and potential timing-based side channels. Moreover, practical adoption of SNNs remains limited by the relative immaturity of their training algorithms and ecosystem support compared to ANNs. Bridging these gaps through advances in training methods, hardware-software co-design, and standardized evaluation frameworks will be essential to unlock the full potential of neuromorphic models in building secure and privacy-preserving AI systems.

VII. Conclusions

Neuromorphic computing shows great promise for the realization of very low-power AI, increasing performance and energy efficiency. While SNNs originate from mimicking biological neural systems known for their remarkable fault tolerance capabilities, their proper functionality is threatened by various trustworthiness issues, such as hardware-related faults and security vulnerabilities. Thus, assessing the dependability of neuromorphic hardware accelerators and providing solutions, as outlined in this paper, is vital to ensure trustworthy deployment of SNNs and neuromorphic engines for critical AI applications, such as in healthcare, automotive, and space.

References

[1] Y. LeCun et al., "Deep learning," *Nature*, 2015.
[2] L. Alzubaidi et al., "Review of deep learning: concepts, CNN architectures, challenges, applications, future directions," *J. of Big Data*, 2021.
[3] Z. Zhou et al., "Edge intelligence: Paving the last mile of artificial intelligence with edge computing," *Proceedings of the IEEE*, 2019.
[4] N. P. Jouppi et al., "In-datacenter performance analysis of a tensor processing unit," in *ISCA*, 2017.
[5] C. Zhang et al., "Benchmarking and in-depth performance study of large language models on Habana Gaudi processors," in *SC-W*, 2023.

[6] P. A. Merolla et al., "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science*, 2014.
[7] M. Davies et al., "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, 2018.
[8] E. Painkras et al., "SpiNNaker: A 1-W 18-core system-on-chip for massively-parallel neural network simulation," *JSSC*, 2013.
[9] N. Qiao et al., "A reconfigurable on-line learning spiking neuromorphic processor comprising 256 neurons and 128K synapses," *Front. Neurosci.*, 2015.
[10] F. Akopyan et al., "TrueNorth: Design and tool flow of a 65 mW 1 million neuron programmable neurosynaptic chip," *TCAD*, 2015.
[11] W. Maass, "Networks of spiking neurons: The third generation of neural network models," *Neural Netw.*, 1997.
[12] A. L. Hodgkin and A. F. Huxley, "Currents carried by sodium and potassium ions through the membrane of the giant squid axon of loligo," *J. of Physiology*, 1952.
[13] J. Nagumo et al., "An active pulse transmission line simulating nerve axon," *Proceedings of the IRE*, 1962.
[14] W. Gerstner, "Time structure of the activity in neural network models," *Phys. Rev. E*, 1995.
[15] G. Indiveri et al., "Neuromorphic silicon neuron circuits," *Front. Neurosci.*, 2011.
[16] W. Gerstner et al., *Neuronal dynamics: From single neurons to networks and models of cognition*, Cambridge University Press, 2014.
[17] A. Tavanaei et al., "Deep learning in spiking neural networks," *Neural Netw.*, 2019.
[18] K. Roy et al., "Towards spike-based machine intelligence with neuromorphic computing," *Nature*, 2019.
[19] C. D. Schuman et al., "Opportunities for neuromorphic computing algorithms and applications," *Nat. Comput. Sci.*, 2022.
[20] J. K. Eshraghian et al., "Training spiking neural networks using lessons from deep learning," *Proc. IEEE*, 2023.
[21] S. B. Shrestha and G. Orchard, "SLAYER: Spike layer error reassignment in time," in *Proc. Adv. Neural Inf. Process. Syst. (NeurIPS)*, 2018.
[22] L. A. Camuñas-Mesa et al., "Event-driven sensing and processing for high-speed robotic vision," in *BioCAS*, 2014.
[23] A. Joubert et al., "Hardware spiking neurons design: Analog or digital?," in *IJCNN*, 2012.
[24] S. A. El-Sayed et al., "Spiking neuron hardware-level fault modeling," in *IOLTS*, 2020.
[25] H. G. Stratigopoulos et al., "Testing and reliability of spiking neural networks: A review of the state-of-the-art," in *DFT*, 2023.
[26] H.-Y. Tseng et al., "Machine learning-based test pattern generation for neuromorphic chips," in *ICCAD*, 2021.
[27] E. Vatajelu et al., "Special session: Reliability of hardware-implemented spiking neural networks (SNN)," in *VTS*, 2019.
[28] K.-W. Hou et al., "Fault modeling and testing of memristor-based spiking neural networks," in *ITC*, 2022.
[29] A. B. Gogebakan et al., "SpikingJET: Enhancing fault injection for fully and convolutional spiking neural networks," in *IOLTS*, 2024.
[30] T. Spyrou et al., "SpikeFI: A fault injection framework for spiking neural networks," *arXiv:2412.06795*, 2024.
[31] T. Spyrou et al., "Neuron fault tolerance in spiking neural networks," in *DATE*, 2021.
[32] S. A. El-Sayed et al., "Compact functional testing for neuromorphic computing circuits," *TCAD*, 2023.
[33] T. Spyrou et al., "Reliability analysis of a spiking neural network hardware accelerator," in *DATE*, 2022.
[34] A. Gebregiorgis et al., "Tutorial on memristor-based computing for smart edge applications," *Mem.-Mater., Devices, Circ. and Syst.*, 2023.
[35] H. Aziza et al., "On the reliability of rram-based neural networks," in *VLSI-SoC*, 2023.
[36] T. Spyrou et al., "Fault tolerant design for memristor-based ai accelerators," in *DTTIS*, 2024.
[37] S. Diware et al., "Unbalanced bit-slicing scheme for accurate memristor-based neural network architecture," in *AICAS*, 2021.
[38] F. Su et al., "Testability and dependability of AI hardware: Survey, trends, challenges, and perspectives," *Des. Test*, 2023.
[39] I. W. Chiu et al., "Automatic test configuration and pattern generation (atcpg) for neuromorphic chips," in *ICCAD*, 2022.
[40] X. P. Chen et al., "Test compression for neuromorphic chips," in *ETS*, 2024.
[41] S. Raptis and H. G. Stratigopoulos, "Minimum time maximum fault coverage testing of spiking neural networks," in *DATE*, 2025.
[42] A. Saha et al., "Post-manufacture criticality-aware gain tuning of timing encoded spiking neural networks for yield recovery," in *ETS*, 2024.
[43] S. A. El-Sayed et al., "Self-testing analog spiking neuron circuit," in *SMACD*, 2019.
[44] J. H. B. Wijekoon and P. Dudek, "Compact silicon neuron circuit with spiking and burtsing behaviour," *Neural Netw.*, 2008.
[45] R. V. W. Putra et al., "SoftSNN: Low-cost fault tolerance for spiking neural network accelerators under soft errors," in *DAC*, 2022.
[46] T. Spyrou and H. G. Stratigopoulos, "On-line testing of neuromorphic hardware," in *ETS*, 2023.
[47] N. Srivastava et al., "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, 2014.
[48] R. V. W. Putra et al., "ReSpawn: Energy-efficient fault-tolerance for spiking neural networks considering unreliable memories," in *ICCAD*, 2021.
[49] R. V. W. Putra et al., "SparkXD: A framework for resilient and energy-efficient spiking neural network inference using approximate DRAM," in *DAC*, 2021.
[50] A. Siddique and K. A. Hoque, "Improving reliability of spiking neural networks through fault aware threshold voltage optimization," in *DATE*, 2023.
[51] C. D. Schuman et al., "Resilience and robustness of spiking neural networks for neuromorphic systems," in *IJCNN*, 2020.
[52] P. I. Vaz et al., "Improving TID radiation robustness of a CMOS OxRAM-based neuron circuit by using enclosed layout transistors," *TVLSI*, 2021.
[53] A. Saha et al., "A resilience framework for synapse weight errors and firing threshold perturbations in RRAM spiking neural networks," in *ETS*, 2023.
[54] S. Karim et al., "Assessing self-repair on FPGAs with biologically realistic astrocyte-neuron networks," in *ISVLSI*, 2017.
[55] J. Liu et al., "SPANNER: A self-repairing spiking neural network hardware architecture," *Trans. Neural Netw. Learn. Syst.*, 2018.
[56] S. Karim et al., "FPGA-based fault-injection and data acquisition of self-repairing spiking neural network hardware," in *ISCAS*, 2018.
[57] S. Karim et al., "AstroByte: Multi-FPGA architecture for accelerated simulations of spiking astrocyte neural networks," in *DATE*, 2020.
[58] M. Isik et al., "A design methodology for fault-tolerant computing using astrocyte neural networks," in *Comput. Frontiers*, 2022.
[59] A. P. Johnson et al., "Homeostatic fault tolerance in spiking neural networks: A dynamic hardware perspective," *TCAS-I*, 2018.
[60] A. Hashmi et al., "Automatic abstraction and fault tolerance in cortical microachitectures," in *ISCA*, 2011.
[61] T. Titirsha et al., "Endurance-aware mapping of spiking neural networks to neuromorphic hardware," *Trans. Parallel Distrib. Syst.*, 2022.
[62] A. Paul et al., "On the mitigation of read disturbances in neuromorphic inference hardware," *Des. Test*, 2023.
[63] A. Madry et al., "Towards deep learning models resistant to adversarial attacks," in *ICLR*, 2018.
[64] V. Venceslai et al., "Neuroattack: Undermining spiking neural networks security through externally triggered bit-flips," in *IJCNN*, 2020.
[65] S. Raptis and H. G. Stratigopoulos, "Input-specific and universal adversarial attack generation for spiking neural networks in the spiking domain," in *IJCNN*, 2025.
[66] U.S. Department of Health & Human Services, "Health information privacy," 2024, Accessed: 2024-06-03.
[67] A. Arous et al., "Exploring machine learning privacy/utility trade-off from a hyperparameters lens," in *IJCNN*, 2023.
[68] V. Shejwalkar et al., "Membership inference attacks against NLP classification models," in *NeurIPS Workshop Priv. in Mach. Lear.*, 2021.
[69] R. Shokri et al., "Membership inference attacks against machine learning models," in *IEEE SP*, 2017.
[70] A. Moshruba et al., "Are neuromorphic architectures inherently privacy-preserving? an exploratory study," *PoPETs*, 2025.
[71] W. Olin-Ammentorp et al., "Stochasticity and robustness in spiking neural networks," *Neurocomputing*, 2021.
[72] K. Nagarajan et al., "SCANN: Side channel analysis of spiking neural networks," *Cryptography*, 2023.
[73] G. Abad et al., "Sneaky spikes: Uncovering stealthy backdoor attacks in spiking neural networks with neuromorphic data," in *NDSS*, 2024.
[74] S. Raptis et al., "Input-triggered hardware trojan attack on spiking neural networks," in *HOST*, 2025.
[75] K. Nagarajan et al., "Analysis of power-oriented fault injection attacks on spiking neural networks," in *DATE*, 2022.