Minimum Time Maximum Fault Coverage Testing of Spiking Neural Networks

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

***Abstract*—We present a novel test generation algorithm for hardware accelerators of Spiking Neural Networks (SNNs). The algorithm is based on advanced optimization tailored for the spiking domain. It adaptively crafts input samples towards high coverage of hardware-level faults. Time-consuming fault simulation during test generation is circumvented by defining loss functions targeting the maximization of fault sensitisation and fault effect propagation to the output. Comparing the proposed algorithm to the existing ones on three benchmarks, it scales up for large SNN models, and it drastically reduces the test generation runtime from days to hours and the test duration from minutes to seconds. The resultant test input shows near perfect fault coverage and has a duration equivalent to a few dataset samples, thus, besides post-manufacturing testing, it is also suited for in-field testing.**

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

I. Introduction

The brain is the most brilliant computing machine achieving impressive features, such as recognition, reasoning, learning, control, etc., with a low power budget. Neuromorphic computing aims at emulating brain-like functionality to perform a wide variety of cognitive tasks with energy advantage compared to a von Neumann computer and traditional artificial neural networks (ANNs) [1], [2]. Neurmomorphic processors implement a spiking neural network (SNN) similar to the brain structure. The design of neuromorphic processors and hardware accelerators supporting SNNs is attracting high interest nowadays [3].

Testing methodologies need to be developed in parallel for such specialized integrated circuits. Traditional testing methodologies for digital circuits may not apply as these designs may be mixed analog-digital, may not include flip-flops allowing to insert scan chains, and some use emerging non-volatile memristive technologies in a crossbar configuration [4]. Another challenge is that these designs are programmable allowing to map onto them different SNN models, thus there is a lack of specifiability necessitating a functional test approach. However, testing the functionality of a neural network implies performing inference on a large dataset, which is too time consuming to be adopted as a testing methodology in high-volume manufacturing [5]. Efficient testing of AI hardware accelerators is an emerging research field [4], but the focus so far has been more on ANNs rather than SNNs [6].

This work was funded by the ANR RE-TRUSTING project under Grant N^{o} ANR-21-CE24-0015-03 and by the European Network of Excellence dAIEDGE under Grant Agreement N^{o} 101120726.

Existing testing methodologies for SNNs can be broadly categorized into built-in self-testing and functional testing. Built-in self-testing employs extra on-chip resources to detect abnormalities down at the neuron-level and can be combined with fault mitigation strategies [7]–[16]. In contrast, functional testing assumes an SNN model mapped on hardware and aims at identifying a set of inputs that can distinguish functional from faulty devices [17]–[20].

In this work, we propose a novel functional test generation algorithm for SNNs. In previous approaches for SNNs [17]–[20][1], the inputs are either taken from the available dataset [17], [18] or are adversarial examples [17], [19] or are random [20]. As these types of inputs are not geared towards fault detection, the resultant test set ends up comprising tens to hundreds of inputs to achieve high fault coverage. In [19], [20], different test configurations, i.e., models, must be loaded onto the chip which increases further test application time due to the switching time. Test generation time is also a concern for all previous methods as during the course of the algorithm the test set is verified by repetitive fault simulations whose number is unbounded and can significantly exceed the size of the fault model, which already explodes for large networks.

Herein, we propose an algorithm that optimizes an input towards low test application time and high fault coverage. We show that a few back-to-back optimized inputs suffice to achieve near perfect fault coverage. The optimization is performed in the spiking domain using loss functions that take as argument the spike trains at the output of neurons. These loss functions allow to circumvent fault simulation during test generation. Thus, the algorithm runtime is independent of the fault model size and is governed by the SNN inference time, rendering test generation fast and scalable. A single fault simulation campaign may be performed in the end to verify the fault coverage. Finally, the compact test set can be stored on-chip, taking up a small memory space, for in-field testing.

The proposed test generation algorithm makes no assumption about the architecture of the SNN, i.e., fully connected, convolutional or recurrent, no assumption about the information coding scheme, i.e., rate coding or time-to-first-spike coding, and is generic and independent of the SNN hardware accelerator design.

The rest of the article is structured as follows. Section II provides an overview of the principle of operation of SNNs. Section III describes fault modeling and simulation

[1]Functional testing approaches for ANNs are proposed in [21]–[27].

978-3-9826741-0-0/DATE25/© 2025 EDAA

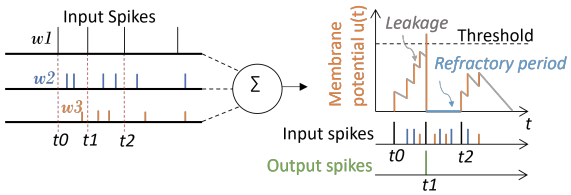


Fig. 1: LIF neuron model behavior.

for SNNs. Section IV presents the proposed test generation algorithm. Section V describes the experimental setup. Section VI describes the results. Section VII concludes the paper pointing to ideas for future work.

II. SNN OVERVIEW

SNNs use discrete events, called spikes, and exploit the timing of spikes to encode, process and transmit information between neurons. This computational paradigm is inspired by the human brain and has the advantage of being low power thanks to the event-based and asynchronous operation.

The most hardware friendly and commonly used spiking neuron model is the Leaky Integrate and Fire (LIF), illustrated in Fig. 1. It balances biological realism with computational simplicity, making it suitable for a hardware implementation. The neuron is modelled with a membrane potential $u(t)$ which increases by w every time an input spike emitted from another neuron arrives, where w is the weight of the synapse linking the two neurons. When the membrane potential reaches a predefined threshold value, the neuron emits a spike and the membrane potential is reset. The neuron has two additional functionalities. The membrane potential is continuously leaking over time and the neuron enters a refractory period after firing a spike, during which it does not integrate incoming spikes and, thereby, does not produce any output spikes.

III. SNN FAULT MODELING AND SIMULATION

Fault modeling translates hardware-level fault effects to behavioral fault models that can be used thereafter to perform fault simulation at the application level, thus speeding up the analysis compared to performing fault simulation at the circuit level [28]–[33]. This also makes test generation hardware-independent and allows implementing it on the software platform used to train the SNN. A fault injection tool orchestrates automatic sequential fault injection in the SNN model and assesses the effect of each fault on the inference accuracy [34], [35]. Main fault models for SNNs include [5], [7]–[20], [28]–[31], [34], [35]:

- Neuron faults: At the extreme a neuron can be (a) saturated, i.e., it produces non-stop output spikes even in the absence of input activity; (b) dead, i.e., it halts input spike propagation. In addition, (c) the output spike train can show timing variations that can be caused by variations in neuron parameters such as membrane potential, threshold, leakage or refractory period.
- Synapse faults: At the extreme a synapse can be (a) dead, i.e., have zero weight; (b) positively (negatively) saturated, i.e., have a very large (small) weight making it a positive (negative) outlier with respect to the weight distribution. In addition, (c) a synapse weight can have a perturbed value, for example induced by a bit-flip in the memory storing the value in digital format.

A fault is critical if it alters the top-1 prediction for at least one sample in the available dataset. However, some faults end up being benign, i.e., they have no effect, their effect is masked as it propagates through the network or they alter the output spike trains but without altering the top-1 prediction. Benign faults having no effect typically involve neurons and synapses that do not participate in the computations, as a result of training or the network architecture. An example is neurons on the perimeter of a feature map and their corresponding synapses that are not part of any receptive field during the strided convolutions.

A testing methodology is evaluated based on its capability of detecting primarily the critical faults, while benign fault detection is a bonus since they may turn out to be critical for other SNN models mapped on the same hardware platform.

IV. TEST GENERATION ALGORITHM

A. Terminology and notation

Let the SNN have L layers with N^ℓ neurons in layer ℓ, $\ell = 1, \cdots, L$, with N^L being the number of output classes.

In SNNs, the input I is spatio-temporal composed of spike trains entering the neurons in the first layer. It can be modelled with a binary tensor of dimensions $N^1 \times (T_{in} * f)$, where T_{in} is the duration of the input and f is the frequency of the internal clock. The temporal dimension is decomposed into $(T_{in} * f)$ equidistant time instances $t_j = j/f$, $j = 1, \cdots, (T_{in} * f)$. $I(i,j) = 1$ if neuron i in the first layer receives a spike at time t_j, otherwise $I(i,j) = 0$.

Let $O^{\ell i}$ denote the output of neuron i in layer ℓ. $O^{\ell i}$ is a binary vector of length $T_{inf} * f$, where T_{inf} is the inference time window. $O^{\ell i}(j) = 1$ if neuron i in layer ℓ fires a spike at time $t_j = j/f, j = 1, \cdots, T_{inf} * f$. We define $O^\ell = [O^{\ell 1}, \cdots, O^{\ell N^\ell}]$ and $O = [O^1, \cdots, O^\ell]$.

The input-output relationship can be expressed as:

$$O^L = f(I). \tag{1}$$

Let us now consider a set of N_f faults $\{f_j\}_{j=1}^{N_f}$. In the presence of a fault, we can write Eq. (1) as:

$$O^L(f_j) = f(I, f_j). \tag{2}$$

We consider that fault f_j is detected if it alters the output spike trains:

$$\|O^L - O^L(f_j)\|_1 > 0, \tag{3}$$

where $\|\cdot\|_1$ is the $L1$ norm. Denoting by $D = \{f_j : \|O^L - O^L(f_j)\|_1 > 0\}$ the set of detected faults, the fault coverage can be defined as:

$$FC = \sum_{j=1}^{N_f} \mathbb{1}_D(j)/N_f, \tag{4}$$

where $\mathbb{1}_D(j)$ is the indicator function of D, i.e., $\mathbb{1}_D(j) = 1$ if $f_j \in D$ and $\mathbb{1}_D(j) = 0$ otherwise.

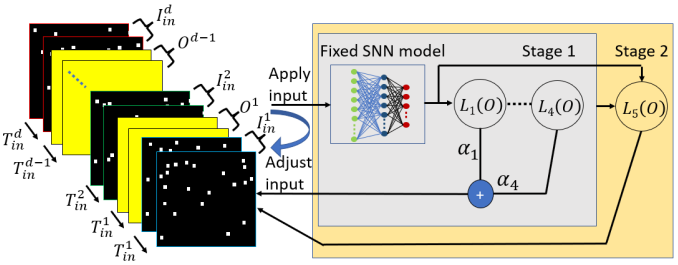


Fig. 2: Test generation algorithm flow.

B. Optimization formulation

The objective of test generation is to find an input that maximizes FC:

$$\max_{I} FC. \tag{5}$$

Using FC as the fitness of a visited input during optimization is impractical since the fault space quickly explodes with the size of the SNN model and evaluating FC for a single input can take several days. The cost of the optimization is $\mathcal{O}(M * T_{FS})$, where M is the number of algorithm iterations and T_{FS} is the fault simulation experiment time to compute FC for a given input.

To address this challenge, we propose to reformulate the problem as a multi-objective optimization problem where the computation of FC is replaced with a set of custom loss functions $\{L_i(O)\}_{i=1}^{K}$ that take as argument spike trains at the outputs of neurons and target sensitization of faults and the propagation of their effect to the output:

$$\min_{I} \sum_{i=1}^{K} \alpha_i * L_i(O), \tag{6}$$

where α_i are the weights for scalarizing the different loss functions and aggregating them into a single loss function. In this way, fault simulation is circumvented during test generation and is performed if needed only once for the final optimized test input to verify its fault coverage. As the loss functions are computed very fast, the cost of the optimization significantly reduces from $\mathcal{O}(M * T_{FS})$ to $\mathcal{O}(M + T_{FS})$.

C. Test generation algorithm

A simplified view of the test generation algorithm flow is illustrated in Fig. 2. Each j-th iteration of the algorithm is divided into two stages and produces an input I_{in}^{j} with variable duration T_{in}^{j}. In each stage, the input is optimized using different loss functions, namely four loss functions in stage 1 and one loss function in stage 2. During the input optimization the SNN model stays fixed. Let $\mathcal{N}$ and $\mathcal{N}_{\mathcal{A}}$ denote the sets of neurons and activated neurons, respectively, where initially $\mathcal{N}_{\mathcal{A}} = \emptyset$. At the end of each iteration, we record the activated neurons, i.e., $\mathcal{N}_{\mathcal{A}} \leftarrow \{n^{\ell i} : |O^{\ell i}| > 1\}$, where $n^{\ell i}$ denotes the i neuron in layer ℓ. The target set of neurons that the next iteration focuses on excludes all previously activated neurons and is given by $\mathcal{N}_{\mathcal{T}} = \mathcal{N} \setminus \mathcal{N}_{\mathcal{A}}$. The algorithm ends until all neurons are activated, i.e., $|\mathcal{N}_{\mathcal{A}}| = |\mathcal{N}|$, or until a time t_{limit} has elapsed.

As spiking neurons have a memory, before starting a new iteration, we need to reset the membrane potential of all neurons. The SNN is reset to "sleep" mode by applying a zero input $\mathbf{0}^j$ whose duration equals T_{in}^{j}.

If d is the number of inputs that have been generated at the end of the optimization, the final test is the concatenation of all inputs interleaved with zero inputs:

$$I = \{I_{in}^{1}, \mathbf{0}^1, I_{in}^{2}, \mathbf{0}^2, \cdots, \mathbf{0}^{d-1}, I_{in}^{d}\}, \tag{7}$$

The duration of the final test is given by:

$$T_{test} = \sum_{j=1}^{d-1} \left(2 * T_{in}^{j}\right) + T_{in}^{d}. \tag{8}$$

Next we describe the loss functions in each stage towards high fault coverage and the input optimization algorithm within a stage to produce one input chunk.

1) Stage 1: The first loss function ensures that all output neurons produce at least one spike during the inference window T_{inf} since intuitively this will reinforce the sensitization of the fault effect to the output:

$$L_1(O^L) = \sum_{i=1}^{N^L} \max(0, 1 - \|O^{Li}\|_1). \tag{9}$$

The second loss function ensures that all neurons are activated, i.e., applying the input makes all neurons generate spikes:

$$L_2(O) = \sum_{\ell=1}^{L} \sum_{i=1}^{N^\ell} \max(0, 1 - \|O^{\ell i}\|_1). \tag{10}$$

The rationale for this loss function is that neuron activation is the necessary condition for exposing dead and timing variation neuron faults. Furthermore, this loss function improves uniformity of spiking activity across all neurons, thus equalizing the importance of neurons during computation, which intuitively helps exposing any neuron fault.

The third loss function aims at exposing timing variation neuron faults by promoting the temporal diversity of the output spike trains of neurons. Temporal diversity $TD^{\ell i}$ of neuron i in layer ℓ is expressed as the number of times the neuron's output changes state during the inference:

$$TD^{\ell i} = \sum_{j=2}^{T_{inf} * f} |O^{\ell i}(j) - O^{\ell i}(j-1)|. \tag{11}$$

The third loss function is expressed as:

$$L_3(O) = \sum_{\ell=1}^{L} \sum_{i=1}^{N^\ell} \max(0, TD_{min} - TD^{\ell i}), \tag{12}$$

where TD_{min} is a minimum imposed temporal diversity to avoid penalizing the neuron.

The fourth loss function targets specifically the activation of synapse and dead neuron faults. When a neuron is activated, this by default propagates spikes through all synapses emanating from the neuron. However, this does not necessarily mean that all synapse faults will be activated. The reason is that incoming synapses to a neuron compete and one synapse may

overshadow the other. More specifically, a synapse that has a large weight and propagates many spikes will contribute more to the increase of the membrane potential of the post-synaptic neuron, possibly largely dominating the contribution of other weaker synapses that propagate less spikes and, thereby, masking their fault. Similarly, if a dead neuron connects with weak synapses to other neurons, it is likely that the dead fault effect will be masked. To address these issues, the fourth loss function ensures uniformity across synapses by minimizing the variance of the contributions of the non-zero weight synapses to the post-synaptic neuron:

$$L_4(O) = \sum_{\ell=2}^{L} \sum_{i=1}^{N^\ell} \underset{j}{\text{Var}} \left(w_{j,i}^{\ell-1,\ell} * |O^{\ell-1,j}| \right), \tag{13}$$

where $w_{j,i}^{\ell-1,\ell}$ is the weight of the synapse connecting neuron j in layer $\ell-1$ to neuron i in layer ℓ.

In the first stage, the input optimization is formulated as:

$$\min_{I} \sum_{i=1}^{4} \alpha_i * L_i(O). \tag{14}$$

Note that there is no loss function that targets specifically saturated neurons. This is because by default saturated neurons are self-activated and this activation is the most extreme, i.e., the neuron produces non-stop spikes.

2) Stage 2: Neuron activation does not necessarily mean that the neuron fault effect will reach the output. More specifically, the spike train of an activated neuron that gets modified due to a fault is blended with spike trains fired by other neurons in the same layer when it reaches a neuron in the next layer. The neuron in the next layer may thus be receiving a high spike rate entering a refractory period. During this time, any fault effect information carried into its input spike train is being dropped. Therefore, the fault effect information may fade out progressively as it propagates through the network and not reach the output due to the refractoriness of neurons. To help the fault effects reach the output, in a second stage, we fine-tune the generated input from the first stage aiming at nullifying the excessive spikes across the network that do not contribute to the output. Essentially, this stage reduces the amount of spikes neurons receive and, thereby, the information loss during their refractory phase. This optimization keeps the output spike trains O^L from the first stage constant and is formulated as follows:

$$\min_{I} L_5(O) \quad \text{s.t.} \quad \text{constant} \quad O^L, \tag{15}$$

where the fifth loss function is defined as follows:

$$L_5(O) = \sum_{\ell=1}^{L-1} \sum_{i=1}^{N^\ell} |O^{\ell i}|. \tag{16}$$

3) Input optimization algorithm within a stage: The input optimization in stages 1 and 2, formulated in Eqs. (14) and (15), respectively, is performed using the gradient descent-based Adam optimizer with adaptive learning rate lr. The flow of the algorithm is illustrated in Fig. 3. As the input to the SNN has a non-differentiable binary format, we start with a random

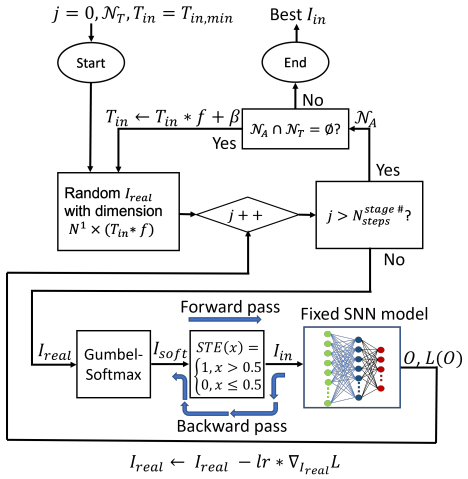


Fig. 3: Input optimization algorithm flow.

real-valued tensor I_{real} with dimension $N^1 \times T_{in,min} * f$. In a first step, we use the Gumbel-Softmax function [36], [37] to produce a real-valued tensor I_{soft} that approximates binary values and allows effective gradient descend-based optimization:

$$I_{soft} = \text{GumbelSoftmax}(I_{real}, \tau). \tag{17}$$

The temperature parameter τ controls the smoothness of the approximation. As τ approaches 0, I_{soft} becomes more discrete, closely approximating binary values. In a second step, we apply the Straight Through Estimator (STE) function [38] to I_{soft}:

$$I_{in} = \text{STE}(I_{soft}). \tag{18}$$

In the forward pass, STE binarizes I_{soft} using a threshold 0.5 to produce the spiking input tensor I_{in} that is applied to the SNN. After the forward pass, we record the spike trains O, compute the loss function L, and backpropagate the error to the input using the same backpropagation pipeline that is used during the training of the SNN. When, we reach the input, the STE function passes on the incoming gradient as if it was an identity function. The tensor I_{real} is adjusted as follows:

$$I_{real} \leftarrow I_{real} - lr * \nabla_{I_{real}} L. \tag{19}$$

Within a stage this procedure is repeated $N_{steps}^{stage\#}$ times, where the superscript $stage\#$ denotes the stage number. If no new neurons are activated after $N_{steps}^{stage\#}$ iterations, i.e., $\mathcal{N}_A \cap \mathcal{N}_T = \emptyset$, then the duration of the input is increased by β time steps to $T_{in,min} * f + \beta$ and the stage optimization is repeated. The stage optimization returns the best I_{in} that is visited.

V. Experimental Setup

A. Case studies

The proposed test generation methodology is demonstrated on three benchmark SNNs trained for the NMNIST [39], IBM DVS128 Gesture [40], and Spiking Heidelberg Digits (SHD) [41] datasets. Figs. 4, 5, and 6 show the SNN architectures and Table I provides the main characteristics. NMNIST is a spiking version of the original frame-based MNIST dataset containing $70K$ images of handwritten digits from 0 to 9. It

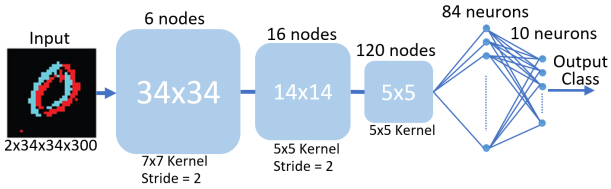


Fig. 4: SNN architecture for the NMNIST dataset.

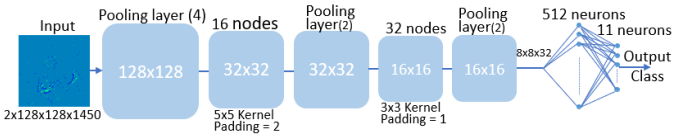


Fig. 5: SNN architecture for the IBM DVS128 Gesture dataset.

was produced by moving a Dynamic Vision Sensor (DVS) while it views MNIST images on an LCD monitor. The IBM DVS128 Gesture dataset was produced by a DVS capturing 11 different hand and arm gestures performed by 29 individuals under 3 different lighting conditions. The SHD dataset consists of 10420 audio recordings of spoken digits from 0 to 9 in German and English languages converted into spike trains.

B. SNN framework

The SNN training and test generation are performed using the SLAYER framework [42] which is built upon PyTorch [43]. The nominal and faulty SNN instances are accelerated on a NVIDIA A100 GPU. Fault injection is performed directly in SLAYER by modifying the synapse weights, neuron parameters, and neuron output spike trains.

C. Optimization parameters

The user-defined parameters of the optimization algorithm are set as follows. $T_{in,min}$ is set as the minimum input duration that produces non-zero output for all neurons in the output layer. Its value is defined by performing an initial optimization $\min_I L_1(O^L)$ starting with $T_{in,min} = 1\,ms$. TD_{min} is set to $T_{in,min}/10$. The numbers of steps within each optimization stage are set to $N^1_{steps} = 2000$ and $N^2_{steps} = N^1_{steps}/2$. The time limit termination condition is set to $t_{limit} = 3\,h$. The input duration increment β is set to $10\,ms$ and it doubles every time the input duration increases. For the temperature τ in the Gumbel-Softmax function we use an annealing schedule with maximum value 0.9. The initial learning rate lr in the Adam optimizer is set to 0.1 and adjusts based on an annealing schedule. The weight factors a_i of the loss functions in stage 1 of the optimization equal the inverse of the expected magnitude of the loss function to ensure balanced contribution to the total loss.

VI. Results

As a first step, a full fault simulation was performed so as to label faults as benign or critical and to be able to calculate the FC of the optimized test stimulus in the end. Table II shows for the three benchmarks the resultant number of critical and benign faults, as well as the total fault simulation time. Such large times in the order of days make prohibitive solving the optimization problem in a straightforward way, i.e., using

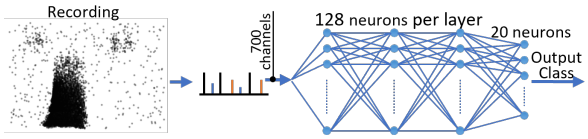


Fig. 6: SNN architecture for the SHD dataset.

TABLE I: Benchmark SNNs characteristics.

	NMNIST	IBM	SHD
Prediction accuracy	98.19%	86.36%	76.59%
# Output classes	10	11	20
# Neurons	1790	25099	404
# Synapses	61908	1059616	124928
Input spatial dimension	$2\times34\times34$	$2\times128\times128$	$700\times1\times1$
Input temporal dimension	$300\,ms$	$1.45\,s$	$1\,s$
Size training set	$60K$	1080	8332
Size testing set	$10K$	261	2088

FC as the loss function, as pointed out in Section IV-B, thus necessitating an alternative test generation strategy as the one proposed herein.

Table III shows various metrics when applying the proposed test algorithm to the three benchmarks. The test generation runtime is only a few hours and scales very well as the size of the network increases, i.e., going from the smaller NMNIST and SHD SNNs to the larger IBM SNN. In all cases, the generated test stimulus has a duration equivalent to less than a dozen original input samples from the dataset. Therefore, besides its use for fast functional post-manufacturing testing, it can be used in addition for in-field testing. The test activates a large percentage of neurons and, thereby, their outgoing synapses, achieving a near perfect coverage of critical faults. As an auxiliary benefit it detects a large percentage of benign faults too. The last row in Table III, shows the maximum drop in prediction accuracy if a critical fault goes undetected, i.e., in essence, it shows the worst case effect that a test escape can have on the SNN performance.

Figs. 7, 8 and 9 provide illustrations taking as an example the IBM SNN. Fig. 7 shows snapshots at different time stamps of the optimized test stimulus. Blue and red dots indicate spikes with positive and negative polarity, respectively. Fig. 8 compares the neuron activity when applying the optimized test input and a randomly chosen input sample from the dataset. It shows a custom grid layout of all neurons across layers, where the yellow (purple) color indicates that the neuron is activated (non-activated). As it can be seen, the optimized test input activates a much higher percentage of neurons compared to the original input sample, i.e., 82.81% as opposed to 29%. As fault coverage is proportional to neuron activation, this color map clearly illustrates the benefit of input optimization. Fig. 9 shows the fault effect propagation to the output layer, expressed as the per-class spike count difference with respect to the expected fault-free response. The per-class spike count difference distributions are shown with different color and are superimposed. For illustrating their tails, the x-axis is broken down into three parts. While a spike count difference of one suffices to detect the fault, for most faults the optimized test spreads widely their effect across the network and propagates it to produce a high output spike train corruption.

TABLE II: Fault simulation results.

	NMNIST	IBM	SHD
# Critical Neuron Faults	2922	25378	794
# Benign Neuron Faults	658	24820	14
# Critical Synapse Faults	96203	934872	311955
# Benign Synapse Faults	89521	2243976	62829
Fault Simulation Time	$\sim$ 5 days	$\sim$ 19 days	$\sim$ 8 days

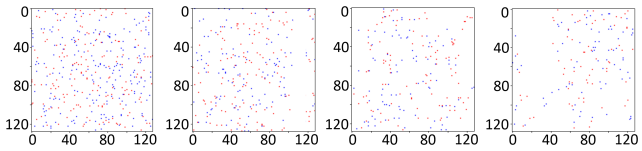
TABLE III: Test generation efficiency metrics.

Metric	NMNIST	IBM	SHD
Test generation runtime	1.5 h	2.5 h	2 h
Test duration (samples)	$\sim$ 8.76	$\sim$ 11.48	$\sim$ 7.82
Test duration (time)	4.96 sec	31.86 sec	14.64 sec
Activated neurons	98.71%	82.81%	91.33%
FC Critical neuron faults	99.97%	99.86%	98.99%
FC Critical synapse faults	96.96%	99.42%	97.25%
FC Benign neuron faults	47.26%	82.29%	21.43%
FC Benign synapse faults	78.02%	58.98%	54.40%
Maximum accuracy drop for undetected critical neuron (synapse) faults	0.1%(1.1%)	0.4%(0.9%)	0.3%(1.5%)

TABLE IV: Comparison with previous works.

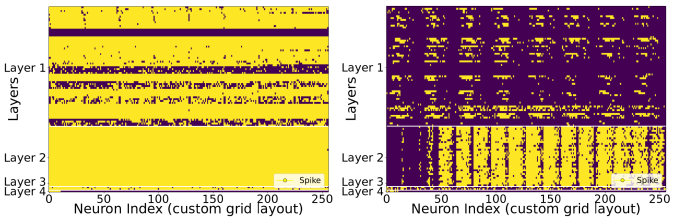
Publication	[17]	[18]	[19]	[20]	This work
NMNIST SNN model	784×128×10	Fig. 4	784 × 256 × 32 × 10	784 × 1024 × 512 × 10	Fig. 4
# Neurons	922	1790	1082	2330	1790
# Synapses	101632	61908	209216	1332224	61908
Test stimulus type	Dataset, Adversarial	Dataset	Adversarial	Random	Optimized
Test generation time	26.19 days (one synapse fault type)	10 days	-	-	1.5 h
# Test configurations	1	1	18	44	1
Test duration (samples)	302	195	662	190	~8.76
Test duration (time)	3.015 min	1.945 min	6.615 min	1.895 min	4.96 sec

Table IV provides a comparison to previous methods in terms of test duration for achieving maximum fault coverage. The comparison is done for the NMNIST SNN which is the only common benchmark. A direct quantitative comparison is possible only to [18], as we implemented this method and applied it on our NMNIST SNN of Fig. 4 and fault model. The comparison to [17], [19], [20] is qualitative as the SNN is a 3- or 4-layer fully-connected network and the fault model is somewhat different. As it can be seen, our method drastically reduces the test duration from a few hundred samples to less than ten samples or, equivalently, from a few minutes to a few seconds. The underline reason is that our method optimizes a short input for high fault coverage, whereas all other methods are based on greedy algorithms that keep adding inputs, i.e., from the dataset, adversarial or random, which are not designed for detecting faults, until maximum fault coverage is reached. The works in [19], [20] add extra test application time due to configuration switching. In terms of test generation time, our method takes 1.5 h while the method in [18] takes 10 days. For the method in [17] test generation is performed per fault type and is reported to be 26.19 days only for synapse weight perturbation faults. Test generation time is not reported in [19], [20] but it should be in the order of days too due to the greedy nature of the algorithm. Although test



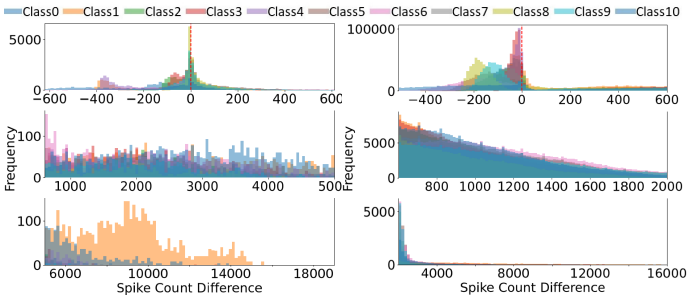
(a) 1 ms. (b) 200 ms. (c) 700 ms. (d) 1000 ms.

Fig. 7: Snapshots of the optimized test stimulus.



(a) Optimized test input. (b) Random dataset input sample.

Fig. 8: Neuron activity per layer for the optimized test input vs. a random input sample from the dataset.



(a) Neuron Faults. (b) Synaptic Faults.

Fig. 9: Per-class spike count difference distribution for detected faults.

generation time is an one-time off-line effort, in other works it depends on the fault model size which increases with the SNN model size, while in our method it is independent of the fault model size and scales very well with the SNN model size, as shown in Table III.

VII. Conclusion

We presented a novel test generation algorithm for SNNs that employs optimization in the spiking domain to craft a short duration test input that maximizes fault coverage. The test input is adjusted to minimize loss functions defined on the network's spiking activity that are inversely proportional to fault coverage. In this way, time-consuming fault simulations are avoided and the algorithm scales up very well for any SNN size. The comparison with previous algorithms on the NMNIST benchmark demonstrated a test duration reduction of over 95% and a test generation time reduction of over 99%. The algorithm is demonstrated on three benchmarks, i.e., NMNIST, IBM DVS128 Gesture, and SHD, achieving over 99% neuron critical fault coverage and over 96% critical synapse fault coverage, while the test input size is equivalent to less than a dozen dataset samples and, thereby, is also suitable for periodic online testing. Future work will focus on defining new loss functions to further improve fault coverage and reduce test duration.

REFERENCES

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

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

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

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

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

[6] H. Stratigopoulos, T. Spyrou, and S. Raptis, "Testing and reliability of spiking neural networks: A review of the state-of-the-art," in *Proc. IEEE Int. Symp. Defect Fault Toler. VLSI Nanotechnol. Syst. (DFT)*, Oct. 2023.

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

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

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

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

[11] 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.

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

[13] T. Spyrou, S. A. El-Sayed, E. Afacan, L. A. Camuñas-Mesa, B. Linares-Barranco, and H.-G. Stratigopoulos, "Neuron fault tolerance in spiking neural networks," in *Proc. Design Autom. Test Europe Conf. (DATE)*, Feb. 2021, pp. 743–748.

[14] 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.

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

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

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

[18] S. A. El-Sayed, T. Spyrou, L. A. Camuñas-Mesa, and H.-G. Stratigopoulos, "Compact functional testing for neuromorphic computing circuits," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 42, no. 7, pp. 2391–2403, 2023.

[19] I-W. Chiu, X.-P. Chen, J. S.-I. Hu, and C.-M. J. Li, "Automatic test configuration and pattern generation (atcpg) for neuromorphic chips," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Oct./Nov. 2022.

[20] X.-P. Chen, H.-Y. Huang, C.-Y. Hsiao, J. S.-I. Hu, and C.-M. J. Li, "Test compression for neuromorphic chips," in *Proc. IEEE Eur. Test Symp. (ETS)*, May 2024.

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

[22] 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.

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

[24] 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.

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

[26] A. Ruospo *et al.*, "Image test libraries for the on-line self-test of functional units in gpus running CNNs," in *Proc. IEEE Eur. Test Symp. (ETS)*, May 2023.

[27] V. Turco, A. Ruospo, G. Gavarini, E. Sanchez, and M. Sonza Reorda, "Uncovering hidden vulnerabilities in CNNs through evolutionary-based image test libraries," in *Proc. IEEE Int. Symp. Defect Fault Toler. VLSI Nanotechnol. Syst. (DFT)*, Oct. 2023.

[28] 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.

[29] 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.

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

[31] T. Spyrou, S. A. El-Sayed, E. Afacan, L. A. Camuñas-Mesa, B. Linares-Barranco, and H.-G. Stratigopoulos, "Reliability analysis of a spiking neural network hardware accelerator," in *Proc. Design Autom. Test Europe Conf. (DATE)*, Mar. 2022, pp. 370–375.

[32] A. Ruospo, E. Sanchez, L. M. Luza, L. Dilillo, M. Traiola, and A. Bosio, "A survey on deep learning resilience assessment methodologies," *Computer*, vol. 56, no. 2, pp. 57–66, Feb. 2023.

[33] M. H. Ahmadilivani, M. Taheri, J. Raik, M. Daneshtalab, and M. Jenihhin, "A systematic literature review on hardware reliability assessment methods for deep neural networks," *ACM Comput. Surv.*, vol. 56, no. 6, Jan. 2024.

[34] A. B. Gogebakan, E. Magliano, A. Carpegna, A. Ruospo, A. Savino, and S. Di Carlo, "SpikingJET: Enhancing fault injection for fully and convolutional spiking neural networks," in *Proc. IEEE Int. Symp. On-Line Test. Robust Syst. Des. (IOLTS)*, Jul. 2024.

[35] T. Spyrou, S. Hamdioui, and H.-G. Stratigopoulos, "SpikeFI: A fault injection framework for spiking neural networks," *arXiv:2412.06795*, 2024.

[36] E. Jang, S. Gu, and B. Poole, "Categorical reparameterization with gumbel-softmax," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, Feb. 2017.

[37] C. J. Maddison, A. Mnih, and Y. W. Teh, "The concrete distribution: A continuous relaxation of discrete random variables," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, Nov. 2017.

[38] Y. Bengio, N. Léonard, and A. Courville, "Estimating or propagating gradients through stochastic neurons for conditional computation," *arXiv:1308.3432*, 2013.

[39] 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.

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

[41] B. Cramer, Y. Stradmann, J. Schemmel, and F. Zenke, "The Heidelberg spiking data sets for the systematic evaluation of spiking neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 33, no. 7, pp. 2744–2757, Jul. 2022.

[42] S. B. Shrestha and G. Orchard, "SLAYER: Spike layer error reassignment in time," in *Proc. Adv. Neural Inf. Process. Syst. (NeurIPS)*, Dec. 2018, pp. 1479–1428.

[43] A. Paszke *et al.*, "PyTorch: An imperative style, high-performance deep learning library," in *Proc. Adv. Neural Inf. Process. Syst. (NeurIPS)*, Dec. 2019, pp. 8026–8037.