Post-Manufacture Criticality-Aware Gain Tuning of Timing Encoded Spiking Neural Networks for Yield Recovery

Anurup Saha, Kwondo Ma, Chandramouli Amarnath and Abhijit Chatterjee
School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta GA 30332
Email: {asaha74, kma64, chandamarnath}@gatech.edu, abhijit.chatterjee@ece.gatech.edu

2024 IEEE European Test Symposium (ETS) | 979-8-3503-4932-0/24/$31.00 ©2024 IEEE | DOI: 10.1109/ETS61313.2024.10567479

***Abstract*—Time-to-first-spike (TTFS) encoded spiking neural networks (SNNs), implemented using memristive crossbar arrays (MCA), achieve higher inference speed and energy efficiency compared to artificial neural networks (ANNs) and rate encoded SNNs. However, memristive crossbar arrays are vulnerable to conductance variations in the embedded memristor cells. These degrade the performance of TTFS encoded SNNs, namely their classification accuracy, with adverse impact on the yield of manufactured chips. To combat this yield loss, we propose a post-manufacture testing and tuning framework for these SNNs. In the testing phase, a timing encoded signature of the SNN, which is statistically correlated to the SNN performace, is extracted. In the tuning phase, this signature is mapped to optimal values of the tuning knobs (gain parameters), one parameter per layer, using a trained regressor, allowing very fast tuning (about 150ms). To further reduce the tuning overhead, we rank order hidden layer neurons based on their criticality and show that adding gain programmability only to 50% of the neurons is sufficient for performance recovery. Experiments show that the proposed framework can improve yield by up to 34% and average accuracy of memristive SNNs by up to 9%.**

***Index Terms*—Spiking Neural Networks, Yield Recovery, Process Variations, Neuromorphic Computing**

I. Introduction

Spiking neural networks (SNN) achieve orders of energy efficiency improvement compared to artificial neural networks (ANNs). [1]. Compute-in-memory (CiM) along with the use of sparsity has been extensively explored to further reduce energy consumption of neuromorphic computing. Time-to-first-spike (TTFS) encoding can achieve higher sparsity compared to the widely used rate encoding [2] and can be efficiently mapped to custom accelerators [3]. Memristive crossbar arrays (MCAs), implemented with emerging non-volatile memories such as resistive random access memory (RRAM) and magnetoresistive random access memory (MRAM), are the primary enabler of analog CiM architectures used for machine learning accelerators. [4]. While recent research has used MCAs for low-latency and energy-efficient SNN inference [5], MCA based SNNs are vulnerable to severe process variation induced performance (measured by classification accuracy of a chip) degradation. After fabrication, SNN chips with unacceptable performance degradation need to be discarded, *resulting in lower manufacturing yield*.

Prior works have addressed testing of SNNs in presence of weight errors and neuron threshold variations [6], [7]. However, *performance recovery of memristive SNNs in presence of process variation* has not received much attention. In this work, we propose a fast post-manufacture testing and tuning framework for MCA based TTFS SNNs that relies on reprogramming one tuning knob (gain) corresponding to each SNN layer. The proposed framework relies on a *timing encoded signature* of the device under test (DUT) and isolates out-of-spec DUTs using a signature assisted testing framework, called *timing encoded signature test (TEST)*. The optimal values of the tuning knobs are predicted from the DUT signature using a trained regressor, allowing fast tuning of SNNs during manufacturing. Finally, to reduce the tuning overhead, we propose a *criticality aware gain tuning* framework, where gain tuning is applied only to critical neurons in hidden layers. Neuron criticality is evaluated using a novel sensitivity score metric. In summary, the main contributions of our work are:
(a) We propose a post-manufacture testing and tuning framework for memristive implementations of TTFS SNNs to compensate for accuracy degradation and manufacturing yield loss caused by realistic process variations during manufacturing memristive crossbar arrays.
(b) We propose criticality aware gain tuning, which involves rank ordering the hidden layer neurons according to their criticality and adding gain tunability only to a subset of these neurons. We show that adding gain tunability to only 50% of the hidden layer neurons is sufficient for yield recovery.

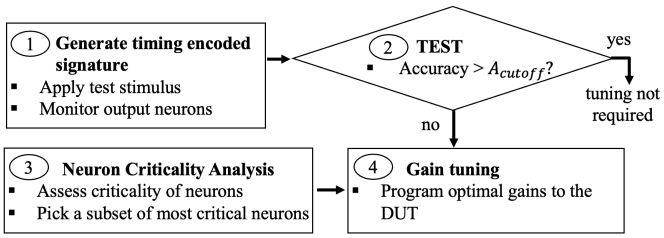

Figure 1: Overview of the testing and tuning framework

II. Approach Overview

Fig. 1 explains the overall tuning framework. The first step, as shown in Fig. 1, involves applying a compact test stimulus (consisting of $\sim$ 50 images) to each manufactured SNN chip or device under test (DUT) to obtain a signature. The timing encoded response of the DUT to the test stimulus is referred as the *timing encoded signature*. The second step determines whether the DUT requires tuning because of variability induced accuracy degradation. The timing encoded signature is passed to a regressor (called the testing regressor), which predicts the accuracy of the DUT. If the predicted accuracy is below a predefined cutoff A_{cutoff}, the DUT undergoes tuning. The testing process is referred to as *timing encoded signature test (TEST)*. For tuning, we introduce a gain parameter within the spiking neuron circuit. All neurons in a layer share the same gain. Before tuning (step 3 of Fig. 1), we rank-order the neurons of each hidden layer by assigning a *sensitivity*

979-8-3503-4932-0/24/$31.00 ©2024 IEEE

score to each neuron. The gain parameter, used for tuning, can be added to all neurons (gain tuning) or a subset of neurons with highest sensitivity scores (criticality aware gain tuning). During post manufacture tuning (step 4 of Fig. 1), the optimal layer-wise gains of the DUT are predicted using a trained regressor and the optimal gains are reprogrammed into the hardware for performance recovery.

III. Preliminaries

A. SNN Hardware Overview

We use time-to-first-spike (TTFS) encoded SNNs for image classification tasks. Following [8], each pixel value p_i is converted to a spike at time $t_i = \frac{p_{max}-p_i}{p_{max}} \times t_{max}$, where t_{max} is the maximum time allocated for SNN inference, and p_{max} is the value of the brightest pixel in the image classification dataset. In the output layer, the neuron emitting the first spike determines the label of an image. If an integrate and fire (IF) neuron j is connected to N input neurons, whose outputs at timestep t are $\{x_i(t)\}_{i=1}^{N}$, then the membrane potential of the j-th neuron can be written as $v_j(t) = \sum_{i=1}^{N} w_{ij} \sum_{\tau=1}^{t} x_i(\tau) = v_j(t-1) + i_j(t)$, where w_{ij} refers to the synaptic weight between neurons i and j and $i_j(t)$ is the input current to the j-th neuron. The neuron output is $y_j(t) = \mathbb{I}[v_j(t) > Th] \times (1 - \sum_{\tau=1}^{t-1} y_j(\tau))$. Here, Th is the spiking threshold and the term $(1 - \sum_{\tau=1}^{t-1} y_j(\tau))$ ensures that a neuron can fire at most once. In memristive crossbar array (MCA) based implementations of SNNs, each weight of an SNN w_{ij} is mapped to an equivalent conductance g_{ij} and the memristor cells are laid out in a rectangular grid. The currents from the cells along each column add up according to Kirchhoff's Current Law (KCL) and generate input currents to spiking neurons.

B. Variability Injection Framework

RRAM devices suffer device-to-device conductance variation, which can manifest in the form of inter-chip (systematic) and intra-chip variation [9], [10]. Further, intra-chip variation constitutes of spatially correlated variation and independent random variation [11]. Because of conductance variation, the weights represented by the crossbar can be different from pretrained model weights. To quantify the impact of process variation, we replace the weights of a pretrained model with equivalent non-ideal weights, i.e., each weight w_{ij} is replaced with a non-ideal weight:

$$w_{ij}^{'} = w_{ij}(1+\epsilon) = w_{ij}(1+\epsilon^{sys}+\epsilon_{ij}^{cor}+\epsilon_{ij}^{rand})$$

We call ϵ_{ij} the non-ideal coefficient. We refer to $\epsilon^{sys}, \epsilon_{ij}^{cor}$, and ϵ_{ij}^{rand} as systematic, spatially correlated and random non-ideal coefficients respectively and use them to model systematic, spatially correlated and random components of variability. By definition, systematic variability is the same across all memristor cells of a DUT. Hence, ϵ^{sys} is sampled only once for each DUT from a zero mean normal distribution with variance σ_{sys}^2. The random non-ideal coefficient ϵ_{ij}^{rand} corresponding to each weight is sampled independently from a zero mean normal distribution with variance σ_{rand}^2. Following [12], we compute the spatially correlated non-ideal coefficients for the weights in a weight matrix $W \in \mathbb{R}^{M\times N}$ using the following steps: ① We refer to the i-th row and j-th column of the weight matrix as the l-th weight, where $l = (i-1)N+j$. Our goal is to compute $\Psi = [\psi_1 \ \psi_2 \ \cdots \ \psi_{MN}]^T$ and set $\epsilon_{ij}^{cor} = \psi_{(i-1)\times N+j}$ ② We generate a correlation matrix $\Omega \in \mathbb{R}^{(MN)\times(MN)}$, where the (l,m)-th element of Ω, Ω_{lm}, represents the correlation between l-th and m-th weights of W. Memristor cells are placed in a rectangular grid in two dimensional crossbars. If the $l(m)$-th weight refers to $i_1(i_2)$-th row and $j_1(j_2)$-th column of W, the physical distance of the memristor cells storing the l and m-th weights of W is $v_{l,m} \propto \sqrt{(i_1-i_2)^2+(j_1-j_2)^2} = c_2\sqrt{(i_1-i_2)^2+(j_1-j_2)^2}$. Following [12], we assume $\Omega_{lm} = \exp(-c_1 \cdot v_{l,m}) = \exp(-\lambda\sqrt{(i_1-i_2)^2+(j_1-j_2)^2}$. Here c_1, c_2 and λ are proportionality constants and $\lambda = c_1c_2$ determines the magnitude of spatial correlation. ③ We calculate a corresponding covariance matrix $\Sigma = \Omega \cdot \sigma_{cor}^2$, where σ_{cor}^2 refers to the variance of spatially correlated variability. Using Cholesky decomposition, we factorize, $\Sigma = LL^T$, where L is a lower triangular matrix. We sample MN standard normal variables $\Phi = [\phi_1 \ \phi_2 \ \cdots \ \phi_{MN}]^T$, where $\phi_l \sim \mathcal{N}(0,1)$. Finally, we calculate $\Psi = L\Phi$.

IV. Timing Encoded Signature Test (TEST)

Timing encoded signature test (TEST) aims at predicting whether the accuracy of a DUT is above or below a predefined cutoff A_{cutoff}. While the proposed framework uses the regressor-based performance prediction approach of [13], we develop a novel *timing encoded signature* customized for TTFS SNN chips. The proposed TEST framework has two steps. ① Each image im_i of a compact test stimulus consisting of N_t images ($\frac{N_t}{C}$ random images from each of the C classes of test dataset) is applied sequentially to a DUT. We define DUT response $DR \in \mathbb{R}^{N_t\times C}$ such that DR_{ij} refers to the spiking time of the j-th output neuron corresponding to the i-th image of the compact test stimulus. In the absence of a spike, we set $DR_{ij} = t_{max}$. We flatten DR to obtain the DUT signature $sig \in \mathbb{R}^{N_t \cdot C}$ ②This signature is passed to a trained regressor (called the testing regressor), which predicts the accuracy of the DUT $\hat{A} = f_\theta(sig)$. The function $f_\theta(\cdot)$ maps the regressor input to its output. To avoid DUT mislabeling caused by regression error, if $|\hat{A} - A_{cutoff}| < \delta_{max}$ (δ_{max} is the maximum probable regression error), the DUT accuracy is estimated using exhaustive test. During exhaustive test, all images in the entire test dataset are applied to the DUT and the accuracy of the DUT is the percentage of correctly classified images. Finally, if $\hat{A} < A_{cutoff}$, the DUT undergoes post-manufacture tuning.

The testing regressor is trained offline using the timing-encoded signatures and accuracies of D training DUTs. The signatures of these D DUTs are obtained using the steps above. The accuracy of each DUT is estimated using an exhaustive test.

V. Post Manufacture Tuning

A. Overall Framework

At timestep t, the membrane potential of the j-th neuron in the l-th layer can be written as $v_j^{(l)}(t) = v_j^{(l)}(t-1) + i_j^{(l)}(t)$, where $i_j^{(l)}(t)$ is the input current to the j-th neuron. We

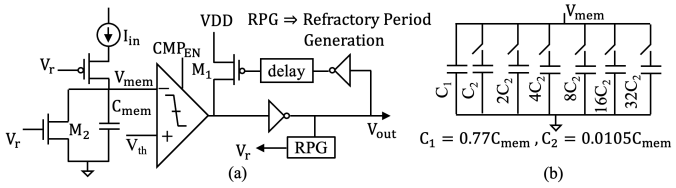

Figure 2: (a) Neuron circuit (b) Programmable capacitor to implement gain tuning

Algorithm 1 Simulated Annealing

1: Initialize $g^{(l)} = 1$ for $1 \leq l \leq L, T = T_0$, $\sigma = 0.1$
   $A = A_{best} =$ accuracy of the DUT before tuning
2: **while** Stopping criteria is not reached **do**
3: $T = r_1 T$ // Set temperature scheduler
4: Set $\tilde{g}^{(l)} \sim \mathcal{N}(g^{(l)}, \sigma^2)\ \forall\ 1 \leq l \leq L$ // choose neighbor
5: clamp $\tilde{g}^{(l)}$ to $[g_{min}, g_{max}]$
6: Evaluate the DUT accuracy $\tilde{A}$ using forward propagation
7: **if** $\tilde{A} > A$ or $uniform(0,1) < \exp(\frac{\tilde{A}-A}{bT})$ **then**
8: Set $A = \tilde{A}$, $g^{(l)} = \tilde{g}^{(l)}$
9: **end if**
10: **if** $\tilde{A} > A_{best}$ **then**
11: Set $A_{best} = \tilde{A}$ and $g^{(l)*} = \tilde{g^{(l)}}$
12: **end if**
13: Set $\sigma = r_2 \sigma$
14: **end while**
15: **return** $g^{(l)*}$

add a *programmable gain* parameter $g^{(l)}$ to the membrane potential update equation to compensate for the effects of process variation:

$$v_j^{(l)}(t) = v_j^{(l)}(t-1) + g^{(l)} i_j^{(l)}(t)$$

The goal of the tuning framework is to find the optimal values of the tuning knobs $g^{(l)*}$ ($1 \leq l \leq L$, where L is the number of SNN layers) for the hidden and output layers of the network. For all neurons in a layer, we use same value of gain. The tuning framework has two steps: ① Before deploying the tuning framework for real time tuning, for a set of D DUTs, the signature of the d-th DUT is calculated as sig_d and optimal tuning knobs $g_d^{(l)*}$ are calculated using simulated annealing (SA) as explained in Algorithm 1. ② During tuning, we use the timing encoded signature sig generated during testing. We predict the optimal tuning knobs using nearest neighbor regression. We first calculate $d^* = \arg\min_{1 \leq d \leq D} ||sig - sig_d||$. Next we set the tuning knobs of the current DUT equal to the optimal tuning knobs of the d^*-th DUT: $g^{(l)*} = g_{d^*}^{(l)*}$

Algorithm 1 explains SA, which starts with initial gain values and an initial temperature T_0. A_{best} is set to the accuracy of the DUT before tuning. In every iteration of SA, the temperature is reduced by a constant cooling rate r_1. New values of the tuning knobs are calculated in line 4, and they are clamped within the interval $[g_{min}, g_{max}]$ to ensure that hardware implementation of gains have low-overhead. The accuracy of the DUT, $\tilde{A}$ is re-evaluated with new tuning knobs $\tilde{g}^{(l)}$. If the accuracy of the current iteration $\tilde{A}$ is better than the accuracy A corresponding to the previously accepted tuning knobs $g^{(l)}$, then the new tuning knobs $\tilde{g}^{(l)}$ are always accepted. Otherwise, the new tuning knob values are accepted with a probability $\exp(\frac{\tilde{A}-A}{bT})$. At every iteration, the value of σ is reduced by a constant rate r_2 to shrink the search space when temperature is low. The algorithm stops when the temperature T reaches below a pre-defined temperature T_c and returns the tuning knob values that achieve the best accuracy.

Next, we explain how programmable gain can be implemented in hardware. In an analog spiking neuron, as shown in Fig. 2a, the input current (I_{in}) is integrated to membrane potential (V_{mem}) across the capacitor C_{mem}. The comparator compares V_{mem} with spiking threshold V_{th}. When V_{mem} exceeds V_{th}, V_{out} becomes '1'. We refer the reader to [8] for detailed functionality of the neuron circuit. To implement gain tuning, we replace the *fixed capacitor* (C_{mem}) with a *programmable capacitor bank* as shown in Fig. 2b. To implement a range of gains from 0.7 to 1.3 ($g_{min} = 0.7, g_{max} = 1.3$ in Algorithm 1), the variable capacitor should be programmable from $0.77C_{mem}$ to $1.43C_{mem}$, as gain is inversely proportional to capacitance. We set $C_1 = 0.77C_{mem}$ and $C_2 = 0.0105C_{mem}$ as shown in Fig. 2b.

B. Criticality Aware Gain Tuning

To reduce the overhead of the tuning circuitry, we propose a lightweight criticality aware gain tuning framework, which adds gain programmability only to a subset of hidden layer neurons. The critical neurons are identified using a *sensitivity score* metric. The spiking time of neuron j is calculated for every image in the training dataset across a set of D DUTs in simulation. If the spiking time of the j-th neuron corresponding to the i-th image and d-th DUT is $t_{j,i}^d$, we first calculate the variance of the firing times across D DUTs, i.e., $v_{j,i} = \frac{1}{D}\sum_{d=1}^{D}(t_{j,i}^d - \frac{1}{D}\sum_{d'=1}^{D}(t_{j,i}^{d'}))^2$. Next we define the sensitivity score as $score_j = \frac{1}{N}\sum_{i=1}^{N} v_{j,i}$. Finally, we sort the neurons according to their sensitivity scores and add gain programmability only to neurons with top p-percerntile scores.

VI. Results

A. Simulation Setup

We evaluate our framework on an SNN trained for the MNIST dataset using temporal backpropagation [8]. We use the SNN architecture implemented in [5] , which has one hidden layer with 400 neurons and an output layer with 10 neurons. The network achieves an accuracy of 94.95%. Following [10], we fix random variation to be 50% of the total variation, i.e., $\sigma_{rand}^2 = \frac{\sigma_{tot}^2}{2}$ (where $\sigma_{tot}^2 = \sigma_{sys}^2 + \sigma_{cor}^2 + \sigma_{rand}^2$). We simulate for different proportions of systematic and spatially correlated variations. We use $\sigma_{tot} = 0.3$, $\lambda = 10^{-3}$, $r_1 = 0.90$, $r_2 = 0.95$. We use 1000 DUTs to train the two regressors used for testing and tuning. The testing regressor is implemented using the gradient boosting regressor from the Scikit Learn library [14]. The performances of the timing encoded signature test and the gain tuning framework are evaluated on another 500 DUTs. For tuning, we use yield improvement and mean accuracy improvement as performance metrics. If the accuracies of 500 DUTs used for performance evaluation are $\{A_i\}_{i=1}^{500}$, then for an accuracy cutoff A_{cutoff}, yield is defined as $Y = \frac{1}{500}\sum_{i=1}^{500}\mathbb{I}[A_i > A_{cutoff}] \times 100\%$. We also calculate the mean (μ_a) of $\{A_i\}_{i=1}^{500}$. Assuming, yield and mean accuracy before (after) tuning are Y^{pre} (Y^{post}) and μ_a^{pre} (μ_a^{post}), we define yield improvement

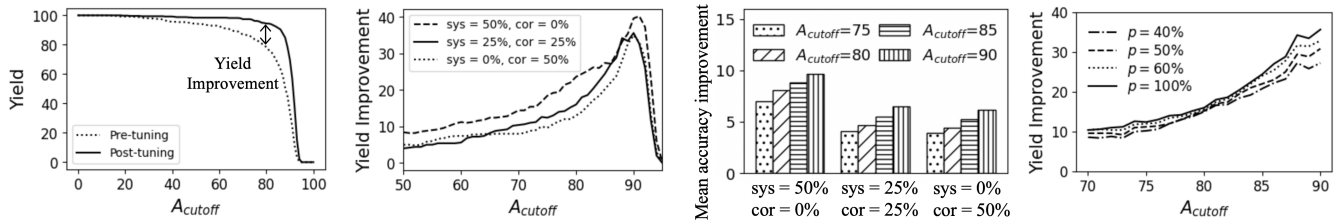

Figure 3: (a) Yield before and after tuning (b) Yield Improvement from gain tuning (c) Mean accuracy improvement from gain tuning (d) Yield improvement from criticality aware gain tuning

Table I: Performance of TEST and CAGT

		TEST	CAGT	
sys(%)	cor (%)	MAE (%)	ΔY	$\Delta\mu_a$
50	0	1.41	32.6	8.54
25	25	1.50	30.8	5.80
0	50	1.57	27.2	5.45

$\Delta Y = Y^{post} - Y^{pre}$ and mean accuracy improvement as $\Delta\mu_a = \mu_a^{post} - \mu_a^{pre}$.

B. Test Performance

Ideally the testing regressor should have low mean absolute error (MAE) and the compact test stimulus should have fewer number of images (N_t). To determine N_t, we set systematic and correlated variations to 25% each. For $N_t = 10, 20, 50$ and 100, the testing regressor achieves a MAE of $2.21\%, 1.77\%, 1.51\%$ and 1.48% respectively. While $N_t = 100$, achieves marginal improvements in MAE compared to $N_t = 50$, it requires applying 2x images to each DUT. Hence, for all subsequent experiments, we choose $N_t = 50$. As shown in Table I, the testing regressor achieves MAE of less than 1.6% for all simulation conditions.

C. Gain Tuning and Criticality Aware Gain Tuning

In Fig. 3a, we show SNN yields before and after gain tuning assuming 25% contributions each from systematic and correlated variability. Since the accuracy of pretrained SNN is 94.95%, beyond that both pre and post tuning yields are zero. For wide performance range of interest ($70-90\%$), post-tuning yield is higher than pre-tuning yield. Fig. 3b and Fig. 3c respectively show the impact of gain tuning on yield and mean accuracy improvement. It is seen that yield improves by as much as 35% and mean accuracy improves by 3-10%.

To evaluate criticality aware gain tuning, we first assume 25% variability from systematic and spatially correlated components and evaluate yield improvements for different values of p, where gain tuning is applied to $p\%$ hidden layer neurons. For $p = 50\%$, yield improvement is comparable to the case of $p = 100\%$ (Fig. 3d), using only half the neurons. As a result, for subsequent experiments we choose $p = 50\%$. Finally we evaluate the impact of CAGT assuming $A_{cutoff} = 90\%$ and $p = 50\%$. Table I shows that CAGT can improve yield by up to 32.60%, mean accuracy by up to 8.54%. Experiments for $A_{cutoff} = 75\%, 80\%$ and 85% also show similar trends.

D. Tuning Time and Overhead

To calculate tuning time, we run the entire framework (timing encoded signature test and gain tuning) on a CPU. To tune one DUT, it requires 152ms. Assuming, $C_{mem} = 199$fF [5], the proposed gain tuning requires adding 86fF capacitor to each neuron circuit.

VII. Conclusion

To address process variation induced manufacturing yield loss in memristive TTFS SNNs, a post-manufacture testing and gain tuning framework is proposed. Further, a neuron criticality aware gain tuning framework is developed to reduce tuning overhead. Experiments show that the proposed framework can recover manufacturing yield by up to 34% in presence of realistic process variation patterns.

Acknowledgment

This research was supported by the U.S. National Science Foundation under Grant: 2128149.

References

[1] K. Roy *et al.*, "Towards spike-based machine intelligence with neuromorphic computing," *Nature 575*, pp. 607–617, 2019.

[2] L. Bonilla *et al.*, "Analyzing time-to-first-spike coding schemes: A theoretical approach," *Frontiers in Neuroscience*, vol. 16, 2022. [Online]. Available: https://www.frontiersin.org/articles/10.3389/fnins.2022.971937

[3] J. Göltz *et al.*, "Fast and energy-efficient neuromorphic deep learning with first-spike times," *Nature Machine Intelligence*, vol. 3, p. 823–835, 2021.

[4] A. E. Arrassi *et al.*, "Energy-efficient snn implementation using rram-based computation in-memory (cim)," in *2022 IFIP/IEEE 30th International Conference on Very Large Scale Integration (VLSI-SoC)*, 2022, pp. 1–6.

[5] S. Oh *et al.*, "Neuron circuits for low-power spiking neural networks using time-to-first-spike encoding," *IEEE Access*, vol. 10, pp. 24 444–24 455, 2022.

[6] S. A. El-Sayed *et al.*, "Compact functional testing for neuromorphic computing circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, no. 7, pp. 2391–2403, 2023.

[7] A. Saha *et al.*, "A resilience framework for synapse weight errors and firing threshold perturbations in rram spiking neural networks," in *2023 IEEE European Test Symposium (ETS)*, 2023, pp. 1–4.

[8] S. R. Kheradpisheh *et al.*, "Temporal backpropagation for spiking neural networks with one spike per neuron," *International Journal of Neural Systems*, vol. 30, no. 06, p. 2050027, May 2020.

[9] p. pouyan *et al.*, "Memristive crossbar memory lifetime evaluation and reconfiguration strategies," *IEEE Transactions on Emerging Topics in Computing*, vol. 6, no. 2, pp. 207–218, 2018.

[10] Z. Deng *et al.*, "Variability-aware training and self-tuning of highly quantized dnns for analog pim," in *2022 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2022, pp. 712–717.

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

[12] J. Xiong *et al.*, "Robust extraction of spatial correlation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 4, pp. 619–631, 2007.

[13] K. Ma *et al.*, "Efficient low cost alternative testing of analog crossbar arrays for deep neural networks," in *2022 IEEE International Test Conference (ITC)*, 2022, pp. 499–503.

[14] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in python," *Journal of machine learning research*, vol. 12, no. Oct, pp. 2825–2830, 2011.