Signature Driven Post-Manufacture Testing and Tuning of RRAM Spiking Neural Networks for Yield Recovery

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

***Abstract*—Resistive random access Memory (RRAM) based spiking neural networks (SNN) are becoming increasingly attractive for pervasive energy-efficient classification tasks. However, such networks suffer from degradation of performance (as determined by classification accuracy) due to the effects of process variations on fabricated RRAM devices resulting in loss of manufacturing yield. To address such yield loss, a two-step approach is developed. First, an alternative test framework is used to predict the performance of fabricated RRAM based SNNs using the SNN response to a small subset of images from the test image dataset, called the SNN *response signature* (to minimize test cost). This diagnoses those SNNs that need to be performance-tuned for yield recovery. Next, SNN tuning is performed by modulating the spiking thresholds of the SNN neurons on a layer-by-layer basis using a trained regressor that *maps the SNN response signature to the optimal spiking threshold values* during tuning. The optimal spiking threshold values are determined by an off-line optimization algorithm. Experiments show that the proposed framework can reduce the number of out-of-spec SNN devices by up to 54% and improve yield by as much as 8.6%.**

***Index Terms*—Spiking Neural Network, Yield Recovery, Alternative Test, Post-manufacture Tuning**

I. Introduction

Compute-in-memory (CIM) leveraging Memristive Crossbar Arrays (MCAs) is attractive for implementing spiking neural networks (SNNs) due to low energy consumption [1]. MCAs can be implemented using emerging non-volatile memories such as Resistive Random Access Memory (RRAM) [2]. However, during manufacturing RRAM devices in a crossbar suffer from device-to-device (random) and chip-to-chip (systematic) variations, which alter the weights of an SNN implemented with such devices from their ideal values, degrading SNN performance (classification accuracy).

Prior research has addressed testability and error resilience of SNNs under hard failures and transient errors. The work of [3] proposes fault models for analog neuron circuits. In [4], training with dropout and fault hopping are investigated for failure recovery. A bound-and-protect technique is presented in [5] to address soft errors in SNNs. In [6], a fault-aware retraining of neuron firing threshold voltages is proposed to mitigate accuracy degradation in the presence of hard faults in systolic arrays and in [7], the impact of non-idealities in memristive crossbars on the accuracy of SNNs is evaluated. The work of [8] investigates the impact of neuron threshold variations on the accuracy of SNNs. In [9], an online test is developed to detect anomalous operation caused by hardware level faults and in [10], a novel test methodology is presented for detecting open and short defects as well as slow and fast integration faults. To address Stuck-at-faults in RRAM based neuromorphic chips, fault aware weight mapping is proposed in [11].

However, *the problem of mitigating the impact of random and systematic process variations on RRAM crossbar based SNN performance* (as determined by SNN classification accuracy) has remained largely unexplored. The work of [12] shows that accelerators based on compute-in-memory can suffer significantly from process variability effects resulting in loss of performance. Due to such effects, many chips manufactured with RRAM crossbar technology are unusable, resulting in loss of manufacturing yield. To resolve this, solutions to two relevant problems are developed in this research. First, a methodology for testing manufactured RRAM crossbar based SNNs is developed that allows *prediction of SNN performance* from its response to a compact image dataset (as opposed to the complete test image dataset of 10,000 images for CIFAR-10, for example). The combined response of the SNN to the applied compact image dataset is called its *response signature*. This minimizes SNN test costs and efficiently isolates SNN chips that are out-of-spec and need tuning. Second, a fast *post-manufacture tuning* technique is developed that *uses the response signature to modulate a (small) number of tuning knobs in the SNN hardware to restore SNN performance to the maximum extent possible* for yield recovery.

While prior work has developed compact functional tests for SNNs using image ranking [13] and Fast Gradient Sign Method [14], these tests do not address the problem of predicting SNN performance (accuracy) from its response to the applied tests. To address systematic process variations in RRAM based DNNs, online digital calibration has been proposed in [15]. However, to the best of our knowledge, the problem of tuning RRAM based SNNs under the impact of systematic and random process variations has not been addressed in prior research. In summary, the key contributions of our work are:

(a) Performance-predictive alternative test methodology for RRAM based SNNs: A compact image dataset is designed in such a way that the classification accuracy of an SNN can be *directly predicted from its response signature using a trained regressor*. This is used for efficiently isolating SNNs that need to be tuned for yield recovery.

(b) Fast SNN Tuning Methodology: The same response signature used for testing is used to directly predict the optimal tuning knob parameters of the RRAM based SNN for restoration of SNN performance (called *signature driven tuning*). The tuning parameters consist of the neuron spiking threshold values of individual layers of the network (one per layer).

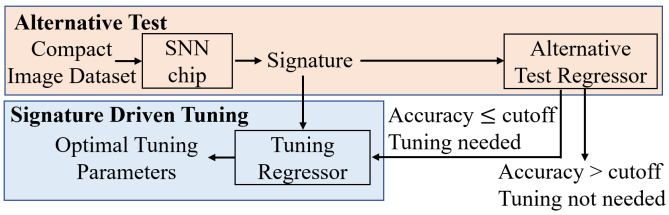


Figure 1: Testing and Tuning Framework

II. Approach Overview

Fig. 1 describes the proposed testing and tuning framework. Post-manufacture, SNN chips are subjected to the proposed alternative test framework. This allows prediction of the classification accuracy of the SNN from its response to a compact image dataset (called its *response signature*) using a trained alternative test regressor (ATR). If the predicted accuracy is below a predefined cutoff, the chip undergoes post-manufacture tuning. During tuning, the spiking threshold voltages of the SNN neurons are used as tuning parameters on a layer-by-layer basis (one per layer) and these are re-calibrated to recover SNN accuracy. Note that the spiking threshold voltage of a neuron is defined as the *value of the membrane potential of a neuron above which it produces a spike at its output*. In analog implementations of spiking neurons, this is implemented as an external input to the neuron from a reference voltage source [8] (which can be modulated for tuning purposes). During signature driven tuning, a *Tuning Regressor* is trained to predict the optimal spiking threshold value for the neurons in each layer of the SNN. These optimal spiking threshold values are determined by an off-line optimization algorithm using calibrated simulation models of the SNN (as described in Section III-C). Consequently these spiking threshold values are re-programmed into the SNN hardware.

III. Preliminaries and Variability Modeling

A. Preliminaries of SNN

Analog implementations of spiking neurons typically use the Integrate and Fire (IF) model due to its simplicity [16]. Here, the input spikes to an IF neuron are integrated in its membrane potential and when the membrane potential exceeds the spiking threshold Th, the neuron emits a spike and its membrane potential is reset. At any timestep t, the membrane potential of a spiking neuron is $m(t) = m(t-1)+i(t)$, where $i(t)$ is the input to the neuron. If $m(t)$ exceeds Th, then its output is high i.e., its output $y(t) = 1$ and $m(t)$ is reset as $m(t) = m(t) - u_{rst}$, where u_{rst} is the reset potential.

For image classification tasks, an SNN has one neuron per class in its output layer. When an image is applied to an SNN, it predicts the class corresponding to the output neuron which emits the most number of spikes.

B. Hardware Overview of SNN

In RRAM based memristive SNNs, weights are programmed as conductances in memristive crossbar arrays (MCA). Fig. 2(a) shows how a weight matrix is mapped to an equivalent conductance matrix. Each weight w_{ij} is mapped to a positive weight w_{ij}^+ and negative weight w_{ij}^-. If w_{ij} is positive, we set $w_{ij}^+ = |w_{ij}|$ and $w_{ij}^- = 0$. Otherwise, we set

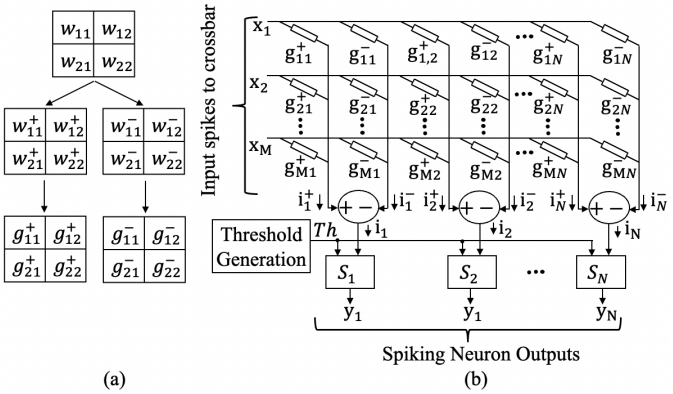


Figure 2: (a) Mapping between weight and conductance (b) Architecture of Memristive SNN

$w_{ij}^+ = 0$ and $w_{ij}^- = |w_{ij}|$. The positive weight w_{ij}^+ is mapped to an equivalent conductance g_{ij}^+ and the negative weight w_{ij}^- is mapped to g_{ij}^-.

Fig 2(b) shows how input currents to IF neurons are computed using a RRAM crossbar. Input spikes $\{x_1, x_2, \cdots x_M\}$ are applied along the rows of the crossbar. If the input spike along m-th row is x_m and the conductance of $(2n-1)$-th column in m-th row is g_{mn}^+, then the output current from the $(2n-1)$-th column of m-th row is $(x_m \times g_{mn}^+)$ and the output current from $(2n-1)$-th column is, $i_n^+ = \sum_{m=1}^{M}(x_m \times g_{mn}^+)$. Similarly, the output current from $2n$-th column is $i_n^- = \sum_{m=1}^{M}(x_m \times g_{mn}^-)$. The input current to the n-th spiking neuron S_n is, $i_n = i_n^+ - i_n^-$. Inside the neuron, the current is integrated and the integrator output (membrane potential) is compared with the spiking threshold. All neurons in a layer of SNN share the same spiking threshold Th, which is re-calibrated during post-manufacture tuning (explained in Section V-C).

C. Variability Modeling

We use RRAM devices with two distinct states: a Low Resistance State (LRS) representing logic '1' and a High Resistance State (HRS) representing logic '0'. The LRS is programmed using the set operation and the HRS is programmed using the reset operation. We explain our proposed variability model assuming a 6-bit fixed point representation for weights. Each 6-bit weight is mapped to a positive and negative weight (explained in Section III-B) which are represented using 6 RRAM devices each.

Process variation is modeled by varying the gap dynamics fitting parameter (γ) of each RRAM device [17] in a crossbar as $\gamma = \gamma_0 + \gamma_{sys} + \gamma_{rand}$, where γ_0 is the value of the gap dynamics fitting parameter in the absence of any variation and $\gamma_{sys}, \gamma_{rand}$ capture the amounts of systematic and random variations respectively. Following the approach of [15], we assume that all RRAM devices in a crossbar have the same systematic variation (γ_{sys}). Random variation of each RRAM device is sampled independently from a zero mean normal distribution with variance σ_{rand}^2.

Algorithm 1 explains the proposed variability injection framework. An SNN is initialized with trained weights in Line 1. The systematic component of variability γ_{sys} is sampled from a zero mean normal distribution with σ_{sys}^2 variance (line

Algorithm 1 DUT Generation using Variability Modeling

1: Initialize DUT with pretrained weights
2: Sample $\gamma_{sys} \sim \mathcal{N}(0, \sigma^2_{sys})$
3: **for** each individual weight w in DUT **do**
4: **if** $w > 0$ **then**
5: $w^+ = |w|$, $w^- = 0$
6: **else**
7: $w^+ = 0$, $w^- = |w|$
8: **end if**
9: Convert w^+ to binary representation $\{b_6^+, \cdots, b_1^+\}$
10: Convert w^- to binary representation $\{b_6^-, \cdots, b_1^-\}$
11: **for** $j = 1, 2, ..., 6$ **do**
12: Convert b_j^+, b_j^- to ideal conductances G_j^+ and G_j^-
13: Sample $\gamma^+_{j,rand}$ and $\gamma^-_{j,rand} \sim \mathcal{N}(0, \sigma^2_{rand})$
14: Set $\gamma_j^+ = \gamma_0 + \gamma_{sys} + \gamma^+_{j,rand}$ and $\gamma_j^- = \gamma_0 + \gamma_{sys} + \gamma^-_{j,rand}$
15: Calculate actual RRAM conductance corresponding to b_j^+, $Gni_j^+ = f(G_j^+, \gamma_j^+)$ and conductance corresponding to b_j^-, $Gni_j^- = f(G_j^-, \gamma_j^-)$
16: **end for**
17: Calculate effective conductance $G_{ni}^{eff} = \sum_{j=1}^{6}(Gni_j^+ - Gni_j^-) \times 2^{j-1}$
18: Convert G_{ni}^{eff} to non-ideal weight w_{ni}
19: Replace w with w_{ni}
20: **end for**

2). For each weight w in the SNN, we calculate w^+ and w^-, which are stored using RRAM devices and find the equivalent binary representation of w^+ and w^- (line 4-10). Next, each bit in the binary representation of w^+ and w^- is converted to the corresponding ideal conductance of the RRAM device storing that bit (line 12). For example, if the j-th bit of w_j^+ is $b_j^+ = 1$, then the equivalent conductance is $G_j^+ = G_{LRS}$. Similarly, if the j-th bit of w_j^+ is $b_j^+ = 0$, then the equivalent conductance is $G_j^+ = G_{HRS}$. We then sample the random component of the variability and calculate the gap dynamics fitting parameter as a sum of its nominal value, systematic component of variability and random component of variability (line 13-14). The ideal conductances are then converted to non-ideal conductances as a function $f(.,.)$ of the ideal conductance and gap dynamics fitting parameter of the device (line 15). The function $f(G_j^+, \gamma_j^+)$ is a polynomial function of G_j^+ and γ_j^+ and the polynomial is estimated using Hspice simulations of RRAM devices. To quantify the impact of non-ideal conductances, we introduce the effective conductance as a weighted sum of the conductances corresponding to each bit of the weight (line 17). Finally, the effective conductance is mapped back to non-ideal weight w_{ni} and the ideal weight w in the SNN is replaced with w_{ni}.

IV. Testing Framework

A. Objectives of the Alternative Test Framework

The accuracy of an SNN trained using an image classification dataset with N_1 training images and N_2 test images is $A = \frac{1}{N_2}\sum_{j=1}^{N_2} \mathbb{I}[ypred(x_j) = y_j]$, where x_j is an image in the test dataset, y_j is the actual label of the image x_j and $ypred(x_j)$ is the label predicted by the SNN corresponding to the image x_j. Here $\mathbb{I}[.]$ refers to the Indicator Function. The goal of the tuning framework is to tune an SNN chip if its accuracy A is below a pre-defined cutoff A_{cutoff}, i.e., $A \leq A_{cutoff}$. It is possible to measure the accuracy A of an SNN Device Under Test (DUT) using an *Exhaustive Test* by applying N_2 images from the test dataset to the DUT and counting the number of correctly classified images. Such an exhaustive test requires inference on a prohibitively large number of images. To overcome the higher test complexity of an exhaustive test, we propose an Alternative Test that characterizes the performance of a DUT using a compact subset of the test dataset. The alternative test has three steps: (1) *Signature Generation* from a DUT, (2) *Training Alternative Test Regressor (ATR)* to predict accuracy of a DUT from its signature and (3) *Performance prediction* using the ATR.

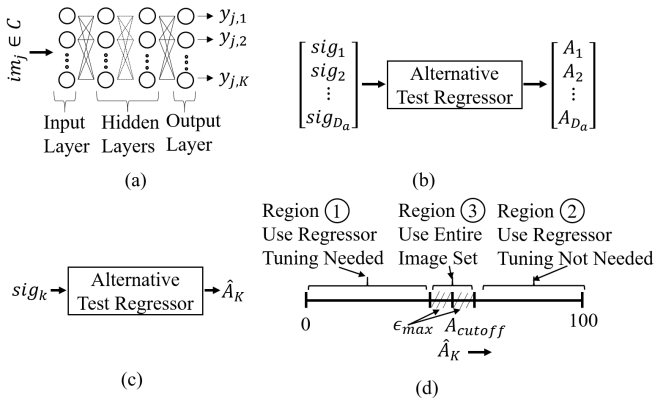


Figure 3: Alternative Test Framework (a) Generation of Signature (b) Training of Alternative Test Regressor (ATR) (c) Accuracy prediction using ATR (d) Mitigating regression error

B. Steps of the Alternative Test

Fig. 3(a) shows how the signature is generated corresponding to a DUT. We first construct a Compact Image Dataset (CID) $\mathcal{C} = \{im_1, im_2, \cdots, im_N\}$ consisting of N images. The images are randomly chosen from the test dataset such that the CID includes images from all classes. During the alternative test, we apply each $im_j \in \mathcal{C}$ and observe the number of spikes emitted by each neuron in the output layer of the DUT. If the k-th output neuron emits $y_{j,k}$ spikes when the j-th image of the CID, im_j is applied, and the output layer has total K neurons then we define the signature of the DUT as a vector of integers, $sig = [y_{1,1}\ y_{1,2} \cdots y_{1,K}\ y_{2,1}\ y_{2,2} \cdots y_{2,K} \cdots y_{N,1}\ y_{N,2} \cdots y_{N,K}]$, $sig \in \mathbb{Z}^{N.K}$.

As shown in Fig. 3(b), the ATR's training data consists of the signatures of first D_a DUTs and the corresponding accuracy of each DUT (measured using an exhaustive test). The ATR is then trained to map the DUT signature to its accuracy. We provide implementation details of the ATR in Section VI-A. As shown in Fig. 3(c), once the ATR is trained, the accuracy of a newly manufactured DUT ($\hat{A}_k$) is predicted from its signature (sig_k) by applying only N images in the CID to the DUT. For example, if a dataset has 10,000 test images and we choose $N = 32$, a chip can be tested using only 32 images resulting in a 10000/32 = 312.5x reduction in test complexity compared to the exhaustive test.

If the accuracy of a DUT is A_k and the accuracy predicted by the ATR is $\hat{A}_k$, the regression error $A_k^{err} = |\hat{A}_k - A_k|$ can affect the efficacy of this alternative test, necessitating mitigation of its impact. For example, if $\hat{A}_k < A_{cutoff} < A_k$,

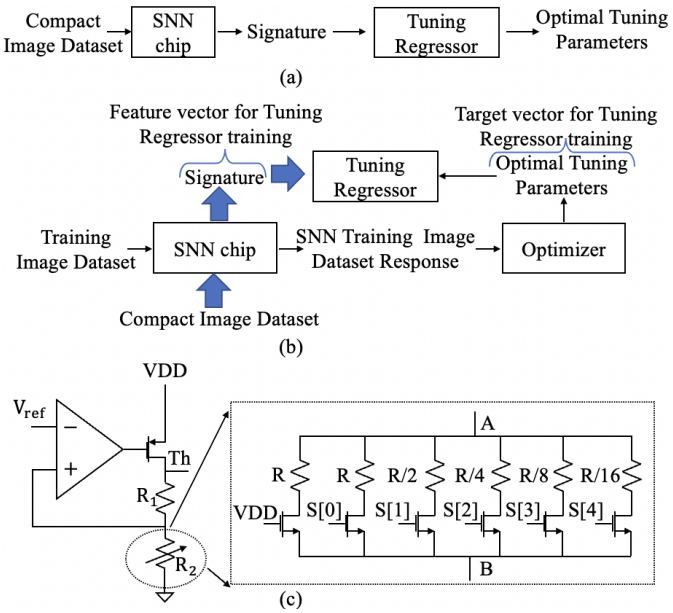


Figure 4: (a) Signature Driven Tuning Framework (b) Offline Training of the Tuning Regressor (c) Hardware Implementation of Spiking Threshold Programmability

then the DUT does not require tuning, whereas the regression output predicts that the DUT requires tuning. On the other hand, if $A_k < A_{cutoff} < \hat{A}_k$ then the DUT needs to undergo tuning whereas the test predicts the opposite. To avoid such incorrect labeling, we introduce a confidence interval ϵ_{max} such that if the predicted accuracy is sufficiently close to the cutoff accuracy, i.e., $|\hat{A}_k - A_{cutoff}| \leq \epsilon_{max}$, we discard the prediction of the ATR and estimate the accuracy A_k of the DUT using all N_2 images in the test dataset. To estimate ϵ_{max}, we compute A_k^{err} $(1 \leq k \leq D_a)$ for the D_a DUTs used for training the ATR. If the mean and standard deviation of $\{A_1^{err}, A_2^{err} \cdots A_{D_a}^{err}\}$ are μ^{err} and σ^{err} respectively, we set $\epsilon_{max} = \mu^{err} + 2 \times \sigma^{err}$.

In summary, as shown in Fig. 3(d), if $|\hat{A}_k - A_{cutoff}| > \epsilon_{max}$ (region 1 and 2), the ATR result determines whether the DUT requires tuning. Otherwise (region 3 in the figure), the actual accuracy A_k is estimated using the entire Test Image Set and the DUT undergoes post-manufacture tuning if $A_k \leq A_{cutoff}$

V. Post-Manufacture Tuning

A. Steps of Signature Driven Tuning

If an SNN has total L layers and the spiking threshold of the l-th layer is $Th^{(l)}$, then the goal of the tuning process is to find the optimal spiking threshold of the l-th layer $Th^{(l)^*}$, which maximizes the manufactured chip's accuracy for $l \in \{1, 2, \cdots, L\}$. Fig. 4(a) shows how an SNN chip is tuned using Signature Driven Tuning. The signature generated during the alternative test is passed to the Tuning Regressor which predicts the optimal tuning parameters from the signature.

Fig. 4(b) shows how training data for the tuning regressor is obtained from D_b DUTs. For each of these D_b chips, we generate its signature sig_k and obtain the optimized tuning parameters of the chip $Th_k^{(l)^*}$ $(1 \leq l \leq L)$ by applying images from the training dataset to the SNN. An optimizer uses the output response of the SNN to the training images to compute $Th_k^{(l)^*}$. The set of signatures and optimized spiking thresholds obtained from D_b number of chips are used as the training data to map the signature of a DUT to optimal tuning parameters. We implement the tuning regressor using Nearest Neighbor regression and we use L_1 norm as a measure of distance. To tune a DUT which has a signature sig_j, we find the SNN chip k^* out of the first D_b chips which has the closest signature to sig_j. Mathematically, we find:

$$k^* = argmin_{k \in \{1,2,\cdots,D_b\}} ||sig_j - sig_k||_1 \quad (1)$$

Here $||.||_1$ denotes L_1 norm of a vector. The tuning regressor predicts that the spiking thresholds of the DUT which is being tuned ($Th_j^{(l)^*}$) are equal to the spiking thresholds of the k^*-th DUT ($Th_{k^*}^{(l)^*}$), i.e., $Th_j^{(l)^*} = Th_{k^*}^{(l)^*} \quad \forall l \in \{1, 2, \cdots, L\}$.

B. Details of Optimization

Algorithm 2 Optimization of Tuning Parameters

1: fix batch size = S, number of epochs = E, number of images in training dataset = N_1, number of batches $B = \frac{N_1}{S}$
2: **for** epoch = 1 upto E: **do**
3: **for** batch id = 1 upto B: **do**
4: Get S images from the training dataset and their corresponding labels
5: Perform inference with the DUT
6: Calculate Loss $\mathcal{L}$ from output response of the DUT and the actual label of images
7: **for** l = 1 upto L: **do**
8: Calculate $\frac{\partial \mathcal{L}}{\partial Th^{(l)}}$ using backpropagation
9: **end for**
10: **for** l = 1 upto L: **do**
11: update $Th^{(l)} = f(Th^{(l)}, \frac{\partial \mathcal{L}}{\partial Th^{(l)}})$ using Adam Optimizer
12: **end for**
13: **end for**
14: **end for**

Algorithm 2 explains how the optimizer in Fig. 4(b) optimizes the spiking thresholds of an SNN. In every iteration of the optimization, we choose S images from the training dataset and the SNN performs inference on those images (line 4-5). Cross-entropy loss is calculated from the output of the DUT and the actual label of the images. The gradient of the loss with respect to each $Th^{(l)}$ is calculated using backpropagation (line 7-9). Once the required gradients are calculated, all spiking threshold values are updated using the Adam Optimizer [18] (line 10 - 12).

C. Hardware Implementation

In Fig. 2(b), we showed that all neurons in a layer share a common spiking threshold. Fig. 4(c) shows how a programmable spiking threshold Th can be generated from a reference voltage V_{ref}, a fixed resistor R_1 and another variable resistor R_2. Using virtual ground principle, it can be shown that $Th = V_{ref} \times (1 + \frac{R_1}{R_2})$. As shown in Fig. 4(c), R_2 is realized as a parallel combination of six resistors and switches $S[0:4]$, where R_2 is the effective resistance between node A and B. For $S[0:4] = 00000$, the effective resistance between A and B is $R_2 = R$. For $S[0:4] = 11111$, we obtain $R_2 = \frac{R}{32}$. In general, for different values of $S[0:4]$, we can obtain $R_2 = \frac{R}{\alpha}$, where $1 \leq \alpha \leq 32$. So,

$V_{th} = V_{ref} \times (1 + \frac{\alpha R_1}{R})$, where the value of α depends on $S[0:4]$. As a result, once the regressor predicts the value of the spiking threshold of a layer, it can be set just by changing 5 digital bits $S[0:4]$.

VI. Results

A. Experimental Methodology

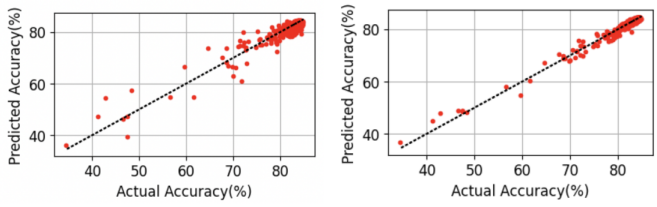
We evaluate the proposed testing and tuning framework on a VGG-9 Convolutional Spiking Neural Network (CSNN), trained on the CIFAR-10 [19] dataset using the SNNTorch framework [20]. We choose VGG-9 architecture to ensure that the proposed framework is scalable to Deep Spiking Neural Networks. We use $T = 25$ timesteps and an initial learning rate of 2×10^{-4}. The network is trained for 50 epochs using the Adam optimizer [18]. Each weight of the CSNN is quantized to a 6-bit fixed point representation. The baseline accuracy of the network is $A_{baseline} = 85.36\%$. Variability injection is performed using PyTorch. If standard deviations of systematic, random and total variability are σ_{sys}, σ_{rand} and σ_{tot} respectively, then $\sigma_{tot}^2 = \sigma_{sys}^2 + \sigma_{rand}^2$. Following the work of [15], we assume 50% contributions from systematic variability and 50% contribution from random variability, i.e., $\sigma_{sys} = \sigma_{rand} = \frac{\sigma_{tot}}{\sqrt{2}}$. We choose σ_{tot} to be 1.6% of the nominal value of gap dynamics fitting parameter $\gamma_0 = 16.5$ [21] [17]. All Hspice simulations are performed using [22].

We implement the Alternative Test Regressor (ATR) using the Gradient Boosting Regressor from Scikit-Learn library [23]. We gather training data for the ATR and the Tuning Regressor from $D_a = 1000$ and $D_b = 400$ DUTs respectively. We evaluate the alternative test and the signature driven tuning framework on another 500 DUTs generated using the variability injection framework.

B. Evaluation Metrics

The classification accuracy of the 500 DUTs used for evaluation are predicted using the ATR. The performance metric of the ATR is the Mean Absolute Error (MAE) computed between the actual accuracy (A) and predicted accuracy ($\hat{A}$) of the DUTs. We use Yield as the metric to evaluate the tuning framework. For a fixed accuracy drop δ, if a DUT has accuracy $A > A_{baseline} - \delta$, it is deemed a good device. If $A \leq A_{baseline} - \delta$, it is deemed a bad device. Out of the 500 DUTs used for evaluation, if there are G_1 good devices and B_1 bad devices, then we define pre-tuning yield as $Y_1 = \frac{G_1}{G_1+B_1} \times 100\%$. After applying post-manufacture tuning, if the number of "good" and "bad" devices are G_2 and B_2 respectively, then the post-manufacture yield is $Y_2 = \frac{G_2}{G_2+B_2} \times 100\%$. We define yield improvement as $\Delta Y = Y_2 - Y_1$.

We compare the results of the signature driven tuning with a baseline method in which the spiking thresholds of each bad device are optimized using Algorithm 2. If post-manufacture tuning was not constrained by computational resources and tuning time, the baseline method would provide the best possible yield achievable using spiking threshold calibration. However the baseline method is impractical for real time tuning due to very high tuning time. The purpose of this comparison is to evaluate the effectiveness of signature driven tuning compared to the baseline method.



(a) (b)

Figure 5: Actual Accuracy of DUTs (A) vs Predicted Accuracy by the ATR ($\hat{A}$) for (a) N = 4 (b) N = 32

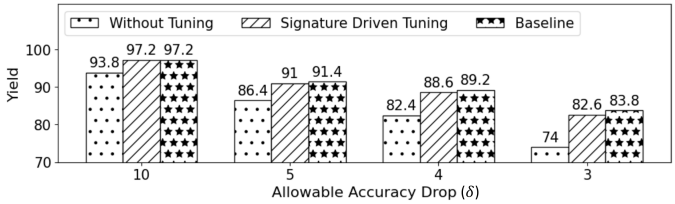


Figure 6: Impact of Tuning on Yield

C. Alternative Test

To choose the size of the Compact Image Dataset N, we sweep N from 4 to 64. Fig. 5(a) and Fig. 5(b) show how the alternative test fares for $N = 4$ and 32 respectively. As we increase N from 4 to 32, the performance of the regressor improves and $\hat{A}$ approaches A. Table I shows the MAE of the regressor as a function of N. Initially, for very small N, the regressor has a large MAE of 1.03%. As N increases (increasing test complexity), the MAE reduces. Doubling N from 32 to 64 reduces the MAE by only 0.02%, indicating marginal gains from the additional test complexity. As a result, for our testing and tuning framework, we choose $N = 32$.

Table I: Number of Images in Compact Image Dataset vs Mean Absolute Error of the ATR

N	4	8	16	32	64
MAE	1.03	0.77	0.70	0.54	0.52

D. Impact of Tuning on Yield

Fig. 6 shows the impact of post manufacture tuning. Reducing the allowable accuracy drop δ causes more DUTs to have accuracy in the range $[0, A_{baselineline} - \delta]$, reducing yield. The trend is similar in presence of tuning as well. For $\delta = 3\%$, signature driven tuning improves yield from 74% to 82.6% resulting in a yield improvement of 8.6%. Yield improvement due to tuning increases as δ is reduced. For large values of δ (10%), signature driven tuning matches the baseline, with both achieving 97.2% yield. For small values of δ, the yield of signature driven tuning is within 1.2% of the baseline.

Table II shows how many bad devices the tuning framework is able to recover for a range of δ values. For a fixed δ if the number of bad devices before tuning is B_1 and after signature driven tuning is B_2, then the percentage recovery is defined as $\frac{(B_1-B_2)}{B_1} \times 100\%$. As shown in Table II, signature driven tuning can recover up to 54.83% of bad devices. The percentage recovery is within 4.62% of the baseline (for $\delta = 3$).

E. Overhead

For the VGG-9 network, the proposed tuning scheme requires reprogramming total 9 5-bit registers (one register per

Table II: Recovery of Bad Devices using Post Manufacture Tuning

	Before Tuning	Signature Driven Tuning		Baseline Method	
Accuracy Drop (δ)	Bad Devices (B_1)	Bad Devices (B_2)	Percentage Recovery ($\frac{B_1-B_2}{B_1} \times 100\%$)	Bad Devices (B_3)	Percentage Recovery ($\frac{B_1-B_3}{B_1} \times 100\%$)
10	31	14	54.83	14	54.83
5	68	45	33.82	43	36.76
4	88	57	35.22	54	38.63
3	130	87	33.07	81	37.69

layer). Since the ATR and the Tuning Regressor are both implemented off-chip, they do not add any area overhead. Tuning time of signature driven tuning is measured as the time required by the Tuning Regressor to predict the spiking thresholds. Tuning time of the baseline method is measured as the time required by the optimization algorithm running on a NVIDIA V100 GPU. Tuning times of signature driven tuning and the baseline method are 0.1 seconds 1070 seconds respectively. In summary, the proposed signature driven tuning achieves *10,700x speedup* compared to the baseline method while compromising only 1.2% of the yield.

VII. Conclusion and Future Work

In this research, we address the challenge of manufacturing yield loss stemming from process variations in RRAM based SNNs. A learning assisted post-manufacture tuning framework predicts the optimal values of a set of tuning knobs (spiking threshold voltage of each layer in the SNN). Tuning effort is reduced by performing post-manufacture tuning only on SNN chips which suffer accuracy degradation beyond a tolerable limit. These out-of-spec SNN chips are isolated by an alternative testing framework. The proposed testing framework, enabled by a trained regressor, achieves orders of reduction in test complexity compared to exhaustive test.

In future, we plan to extend the tuning framework to more complicated spiking neuron models as well as time to first spike (TTFS) encoded SNNs.

Acknowledgement

This research was supported in part by School of Electrical and Computer Engineering, Georgia Institute of Technology and by the U.S. National Science Foundation under Grant: 2414361.

References

[1] A. Ankit, A. Sengupta, P. Panda, and K. Roy, "Resparc: A reconfigurable and energy-efficient architecture with memristive crossbars for deep spiking neural networks," in *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2017, pp. 1–6.

[2] A. E. Arrassi, A. Gebregiorgis, A. E. Haddadi, and S. Hamdioui, "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.

[3] S. A. El-Sayed, T. Spyrou, A. Pavlidis, E. Afacan, L. A. Camuñas-Mesa, B. Linares-Barranco, and H.-G. Stratigopoulos, "Spiking neuron hardware-level fault modeling," in *2020 IEEE 26th International Symposium on On-Line Testing and Robust System Design (IOLTS)*, 2020, pp. 1–4.

[4] 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 *2021 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2021, pp. 743–748.

[5] R. V. W. Putra, M. A. Hanif, and M. Shafique, "Softsnn: Low-cost fault tolerance for spiking neural network accelerators under soft errors," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*. Association for Computing Machinery, 2022, p. 151–156.

[6] A. Siddique and K. A. Hoque, "Improving reliability of spiking neural networks through fault aware threshold voltage optimization," in *2023 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2023, pp. 1–6.

[7] A. Bhattacharjee, Y. Kim, A. Moitra, and P. Panda, "Examining the robustness of spiking neural networks on non-ideal memristive crossbars," in *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design*, ser. ISLPED '22. Association for Computing Machinery, 2022.

[8] A. Saha, C. Amarnath, and A. Chatterjee, "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.

[9] T. Spyrou and H.-G. Stratigopoulos, "On-line testing of neuromorphic hardware," in *2023 IEEE European Test Symposium (ETS)*, 2023, pp. 1–6.

[10] 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 *2022 IEEE International Test Conference (ITC)*, 2022, pp. 92–99.

[11] H. Shin, M. Kang, and L.-S. Kim, "Fault-free: A fault-resilient deep neural network accelerator based on realistic reram devices," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*, 2021, pp. 1039–1044.

[12] Z. Yan, X. S. Hu, and Y. Shi, "Computing-in-memory neural network accelerators for safety-critical systems: Can small device variations be disastrous?" in *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, ser. ICCAD '22. Association for Computing Machinery, 2022.

[13] S. A. El-Sayed, T. Spyrou, L. A. Camuñas-Mesa, and H.-G. Stratigopoulos, "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.

[14] H.-Y. Tseng, I.-W. Chiu, M.-T. Wu, and J. C.-M. Li, "Machine learning-based test pattern generation for neuromorphic chips," in *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, 2021, pp. 1–7.

[15] Z. Deng and M. Orshansky, "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.

[16] A. N. Burkitt, "A review of the integrate-and-fire neuron model: I. homogeneous synaptic input," *Biological cybernetics*, vol. 95, no. 1, pp. 1–19, 2006.

[17] K. Ma, A. Saha, C. Amarnath, and A. Chatterjee, "Efficient low cost alternative testing of analog crossbar arrays for deep neural networks," in *2022 IEEE International Test Conference (ITC)*. IEEE, 2022, pp. 499–503.

[18] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[19] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.

[20] J. K. Eshraghian, M. Ward, E. Neftci, X. Wang, G. Lenz, G. Dwivedi, M. Bennamoun, D. S. Jeong, and W. D. Lu, "Training spiking neural networks using lessons from deep learning," *arXiv preprint arXiv:2109.12894*, 2021.

[21] P.-Y. Chen and S. Yu, "Compact modeling of rram devices and its applications in 1t1r and 1s1r array design," *IEEE Transactions on Electron Devices*, vol. 62, no. 12, pp. 4022–4028, 2015.

[22] Nanoscale Integration and Modeling (NIMO) Group, ASU, "Predictive Technology Model," http://ptm.asu.edu, 2011, online; accessed 8 April 2022.

[23] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.