SWIPE: Enhancing Robustness of ReRAM Crossbars for In-memory Computing

Sujan K. Gonugondla, Ameya D. Patil, Naresh R. Shanbhag
(gonugon2,adpatil2,shanbhag)@illinois.edu
Department of Electrical and Computer Engineering
University of Illinois at Urbana-Champaign

ABSTRACT

Crossbar-based in-memory architectures have emerged as an attractive platform for energy-efficient realization of deep neural networks (DNNs). A key challenge in such architectures is achieving accurate and efficient writes due to the presence of bitcell conductance variations. In this paper, we propose the Single-Write In-memory Program-vErify (SWIPE) method that achieves high accuracy writes for crossbar-based in-memory architectures at 5×-to-10× lower cost than standard program-verify methods. SWIPE leverages the bit-sliced attribute of crossbar-based in-memory architectures and the statistics of conductance variations to compensate for device non-idealities. Using SWIPE to write into ReRAM crossbar allows for a 2× (CIFAR-10) and 3× (MNIST) increase in storage density with $<$ 1% loss in DNN accuracy. In particular, SWIPE compensates for 4.8×-to-7.7× higher conductance variations. Furthermore, SWIPE can be augmented with injection-based training methods in order to achieve even greater enhancements in robustness.

KEYWORDS

resistive crossbars, in-memory, neural networks, ReRAM, crossbars, variations, error correction, compensation, neural networks

ACM Reference Format:
Sujan K. Gonugondla, Ameya D. Patil, Naresh R. Shanbhag. 2020. SWIPE: Enhancing Robustness of ReRAM Crossbars for In-memory Computing. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD '20), November 2–5, 2020, Virtual Event, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3400302.3415642

1 INTRODUCTION

A growing number of applications require efficient implementations of machine learning algorithms, especially on Edge devices. Deep neural networks (DNNs) have evolved into the state-of-the-art approach for machine learning tasks. However, realizing computationally intensive machine learning (ML) algorithms such as DNNs under stringent constraints on energy, latency, and form-factor is a formidable challenge.

In conventional von Neumann architectures, the energy and latency cost of realizing ML algorithms is dominated by memory accesses [9]. In-memory architectures have emerged as an attractive platform for such applications due to their energy-efficiency and high throughput. In particular, architectures that employ non-volatile resistive memory-based crossbars such as [3, 6, 14] have gained popularity due to the large storage density and realizing matrix-vector multiplications (MVMs) which are ubiquitous in machine learning applications.

Despite their benefits, crossbar architectures are highly susceptible to device and circuit non-idealities. For example, the achievable inference accuracy of crossbar architectures is limited by IR drop, device non-linearity, thermal noise, process variations, stuck-at-faults, write noise, and limited device endurance. A number of works have addressed these challenges such as [8] (IR drop), [19] (device variations), and [13] (conductance variations). Recently, on-chip training methods have been proposed [16] to minimize the impact of die-specific variations.

Device variability in crossbar memories stems primarily from spatial variations and cycle-to-cycle (C2C) variations (*write noise*). The impact of spatial variations is die-specific and can be compensated for post-fabrication via on-chip learning [16] methods. However, these methods are expensive and can only be employed if the memory writes are infrequent and they do not address C2C variations which occur in every write cycle. Alternatively, noise injection-based one-time offline training methods [8, 13, 19] determine an averaged set of network parameters for an ensemble of dies thereby avoiding the cost of on-chip learning. However, such methods incur a significant loss in inference accuracy as compared to on-chip training methods. Numerous methods have also focused on strategically restructuring the DNN weights into the array in order to mitigate effects of line resistance [1, 11], improve energy efficiency [4], and improve data reuse [20]. However, the methods are network-specific and difficult to implement on-chip.

Today *program-verify* methods [2, 5, 7, 10] are the predominant approach to address C2C variations due to their high write accuracy. However, these techniques require multiple read and program iterations each time the crossbar array is written into. Furthermore, such methods can write only a single bitcell of a crossbar at a time [7] thereby incurring a high energy and latency overhead. These overheads limit the energy efficiency and latency of in-memory DNN accelerators that employ crossbars [6, 14] especially when realizing complex networks. Thus, there is a compelling need for techniques that achieve the high write accuracy of program-verify methods while minimizing their energy and latency overheads.

In this paper, we propose the Single-Write In-memory Program-vErify (SWIPE) method for in-memory computing applications that

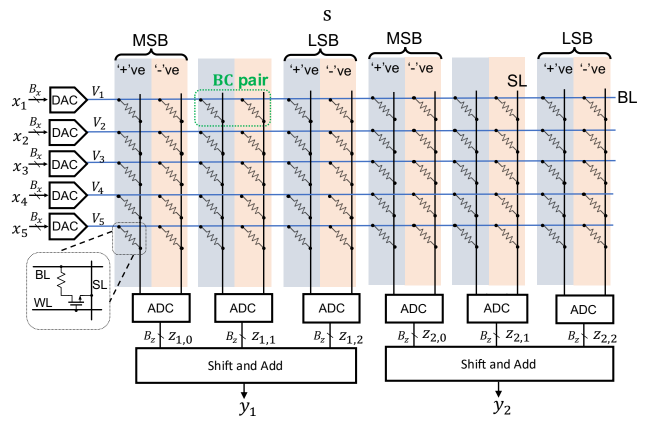

S
MSB
LSB
MSB
LSB
'+'ve
'-'ve
BC pair
SL
BL
DAC
x_1 x_2 x_3 x_4 x_5
B_x
V_1 V_2 V_3 V_4 V_5
WL
ADC
B_z
$z_{1,0}$ $z_{1,1}$ $z_{1,2}$ $z_{2,0}$ $z_{2,1}$ $z_{2,2}$
Shift and Add
y_1 y_2

Figure 1: A 1T-1R resistive crossbar-based in-memory architecture realizing a signed multi-bit MVM computation via differential representation and bit-slicing, with parameters $M = 2$, $N = 5$, and $N_c = 3$.

achieves program-verify [7] level write accuracy while being 5×-to-10× more energy and latency efficient. SWIPE realizes bitcell writes via a *single scan* of the array. It further leverages bit-slicing and the statistics of device variations to compensate for both device mismatch (spatial) and write noise (C2C) variations. Furthermore, SWIPE can be efficiently realized in crossbars by reusing peripheral circuits employed for MVM computations. In this manner, SWIPE *simultaneously* achieves both efficient and accurate programming of crossbar arrays. While SWIPE can be used in any resistive crossbar array, we focus on ReRAM-based crossbars in this paper. Our contributions are summarized below:

- we study the impact of conductance variations on the signal-to-noise ratio (SNR) of ReRAM crossbar MVM and find that it is dominated by the noise in the most significant bit cells.
- we propose SWIPE and justify it both theoretically and via simulations using the Stanford ReRAM Verilog-A model [12].
- we demonstrate the effectiveness of SWIPE in enhancing the accuracy of DNNs realized on ReRAM crossbars in the presence of conductance variations, both spatial and C2C.
- we show that augmenting SWIPE with noise injection methods further enhances the robustness of crossbar DNN implementations.

2 PRELIMINARIES

In this section, we provide the necessary background. We consider a crossbar implementation of the following MVM:

$$\mathbf{y} = \mathbf{W}^\mathsf{T}\mathbf{x} \tag{1}$$

where $\mathbf{x} = [x_1, \ldots, x_N]^\mathsf{T}$ is a $N\times1$ the input vector, $\mathbf{y} = [y_1, \ldots, y_M]^\mathsf{T}$ is the $M \times 1$ output vector, and $\mathbf{W}$ denotes a $M \times N$ weight matrix with weights w_{ij} at i-th row and j-th column. Without loss of generality, we assume $x_i \in [0, 1]$ and $w_{i,j} \in [-1, 1]$, with precisions B_x and B_w, respectively.

2.1 MVM via a Resistive Crossbar

The 1T-1R resistive crossbar-based in-memory architecture in Fig. 1 realizes a signed B_w-b×B_x-b MVM in (1). Each 1T-1R bitcell (BC) stores B_c bits and two adjacent BCs (BC pair) to realize a *signed* scalar using *differential representation* [6, 8]. Using a *bit-sliced* architecture [3, 14], a B_w-b signed weight $w_{i,j}$ is stored in N_c adjacent BC pairs with $B_w = N_c B_c + 1$. Thus, the $M \times N$ B_w-b weight matrix $\mathbf{W}$ requires a $2N \times N_c M$ crossbar. Each column pair in this architecture computes a partial dot product that is converted to digital domain via a differential analog to digital converter (ADC).

The j^{th} partial dot product $z_{j,k}$ is computed in a pair of columns as follows:

$$z_{j,k} = \frac{\alpha}{\Delta G_{\max} V_{\max}} \sum_{i=1}^{N} \Delta G_{i,j,k} V_i \tag{2}$$

where $V_i = x_i V_{\max}$ is the voltage on the i-th bit-line (BL) (see Fig. 1), $\Delta G_{i,j,k}$ is the difference between conductances of BCs in the k-th BC pair associated with $w_{i,j}$, α is a constant, $\Delta G_{\max}$ is the conductance range, and $V_{\max}$ is the voltage range for the DAC output V_i. Note that $\Delta G_{i,j,k} \in \mathcal{G} = \{g_{-L+1}, \ldots, g_0, \ldots, g_{L-1}\}$ where $|\mathcal{G}| = 2L - 1$, $L = 2^{B_c}$, $g_l = l\Delta g$ ($l = -L+1, \ldots, L-1$), and Δg is the differential conductance step (see Fig. 2).

The partial dot products $z_{j,k}$ are digitized and summed after binary weighing to realize the final dot product y_j (see (1)) as follows:

$$y_j = \sum_{k=0}^{N_c-1} 2^{-kB_c} z_{j,k} = \sum_{i=1}^{N} w_{i,j} x_i \tag{3}$$

Note that the $2N_c$ cells associated with each weight parameter $w_{i,j}$ can be stored across multiple banks [21].

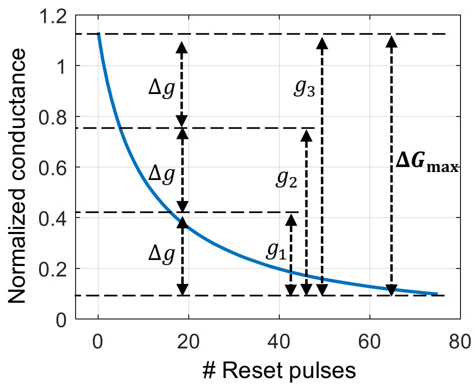
2.2 Non-idealities in Crossbar

ReRAM devices are programmed using two operations: (a) SET, and (b) RESET. The SET operation increases the conductance of the ReRAM device, while RESET reduces it. In most emerging non-volatile memory (eNVM) devices, the SET operation is abrupt, and only the RESET operation is used for multi-level conductance tuning [5]. To obtain the desired conductance change, RESET pulses are either modulated in time, amplitude, or the number of pulses [18]. ReRAM conductance as a function of the number of RESET pulses can be non-linear; appropriately choosing the number of pulses is required to achieve the desired conductance change (see Fig. 2(a)).

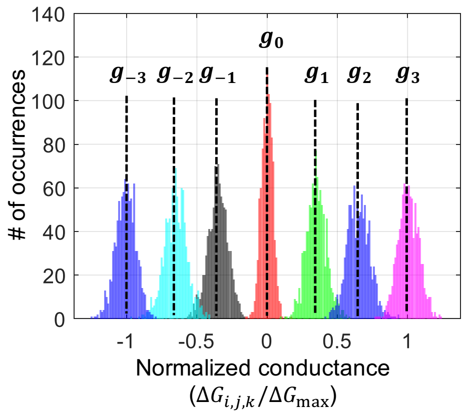
Stochastic non-idealities in a crossbar are due to: (a) *spatial variations* in the BCs, and (b) *write noise*, which includes C2C variations in the cell conductances. The C2C variations occur due to the unpredictability in the ReRAM conductance during the RESET or SET operation. For a BC pair, we model conductance variations due to C2C and the device mismatch as follows:

$$\Delta G_{i,j,k} = \Delta\tilde{G}_{i,j,k} + \eta_{i,j,k} \tag{4}$$

where $\Delta\tilde{G}_{i,j,k} \in \mathcal{G}$ the desired (ideal) cell conductance (*conductance state*), and $\eta_{i,j,k}$ is the variation in conductance caused by both spatial variations and write noise. The distribution of $\eta_{i,j,k}$ conditioned on the value of $\Delta\tilde{G}_{i,j,k}$ is denoted by $P_\eta(\eta_{i,j,k}|\Delta\tilde{G}_{i,j,k} =$



(a)



(b)

Figure 2: ReRAM write noise with $B_c = 2$ bits/cell using the Stanford ReRAM Verilog-A model [12]: (a) normalized conductance vs. number of RESET pulses under ideal (no conductance variation) conditions. The four conductance states are obtained by equipartitioning the conductance range $\Delta G_{\max}$ into three steps, and (b) Monte Carlo simulations showing conductance variations on the conductance difference of the BC pair due to C2C variation and device mismatch when an average number of RESET pulses from (a) is applied to each cell in the BC pair.

$g_l) = \mathcal{N}(0, \sigma^2_{g,l})$ (see Fig. 2(b)). Additionally, the *read noise* during MVM computation, that includes thermal noise and shot noise, can be modeled as an additive Gaussian random variable sampled in every read iteration as suggested in [8].

3 THE SINGLE-WRITE IN-MEMORY PROGRAM-VERIFY (SWIPE) METHOD

In this section, we present the proposed Single-Write In-memory Program-vErify (SWIPE) to minimize the impact of write noise and device mismatch on the SNR of MVM computations in crossbar arrays.

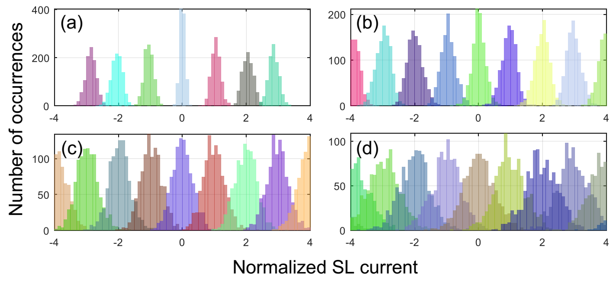


Figure 3: Histograms of the normalized SL current difference ($\propto z_{k,j}$) for different values of weights in the BCs where the average conductance standard deviation $\sigma_g/\Delta G_{\max} = 2.6\%$, $B_x = 1$, $B_c = 2$ with: (a) $N = 1$, (b) $N = 2$, (c) $N = 4$, and (d) $N = 8$. Stanford ReRAM Verilog-A model [12] was used for simulations.

3.1 Impact of Conductance Variations

During a crossbar-based in-memory MVM computation, individual BC currents are accumulated in the source lines (SLs) (see Fig. 1) to compute the dot product (2). This leads to an aggregation of noise due to device variations as shown below:

$$z_{j,k} + \gamma_{j,k} = \frac{\alpha}{\Delta G_{\max} V_{\max}} \sum_{i=1}^{N} \left(\Delta \tilde{G}_{i,j,k} + \eta_{i,j,k}\right) V_i \tag{5}$$

where $z_{j,k}$ the partial dot products from (2), and $\gamma_{j,k}$ is the total noise accumulated in the SL current. Monte Carlo simulations in Fig. 3 show that the outputs $z_{j,k}$ can be easily discriminated if $N = 1$ but becomes increasingly difficult as N increases due to the corresponding increase in the variance of $\gamma_{j,k}$s.

The SNR of the final dot product output y_j in (3) is dominated by variations in cells with higher significance ($k = 0$), i.e., SNR of $z_{j,0}$. In fact, even if high-precision ADCs were to be employed in the architecture in Fig. 1, the SNR of the output y_j is upper bounded as (see Appendix for derivation):

$$\text{SNR}_y \leq \left(\frac{\Delta G_{\max}}{\alpha}\right)^2 \frac{\mathbb{E}\left[y^2\right]\left(1 - 2^{-2B_c}\right)}{N\sigma^2_{g,\min}\mathbb{E}\left[x^2\right]\left(1 - 2^{-2N_c B_c}\right)} \tag{6}$$

where $\sigma^2_{g,\min} = \min_l \sigma^2_{g,l}$. Note that this upper bound flattens out as the weight precision B_c increases.

One way to overcome this SNR bound is to find a method to program $z_{j,k+1}$ such that it is inversely correlated with $\gamma_{j,0}, \ldots, \gamma_{j,k}$, i.e., the value stored in the less significant BC pair is chosen to compensate for the noise in BC pairs of higher significance. The proposed SWIPE method precisely does this by exploiting the bit-sliced nature of the crossbar-based in-memory architecture.

3.2 Proposed SWIPE Algorithm

The proposed SWIPE method writes the N_c BC pairs that store the word $w_{i,j}$ sequentially one BC pair at a time, in a specific order, i.e., from the most significant to the least significant BC pair. While

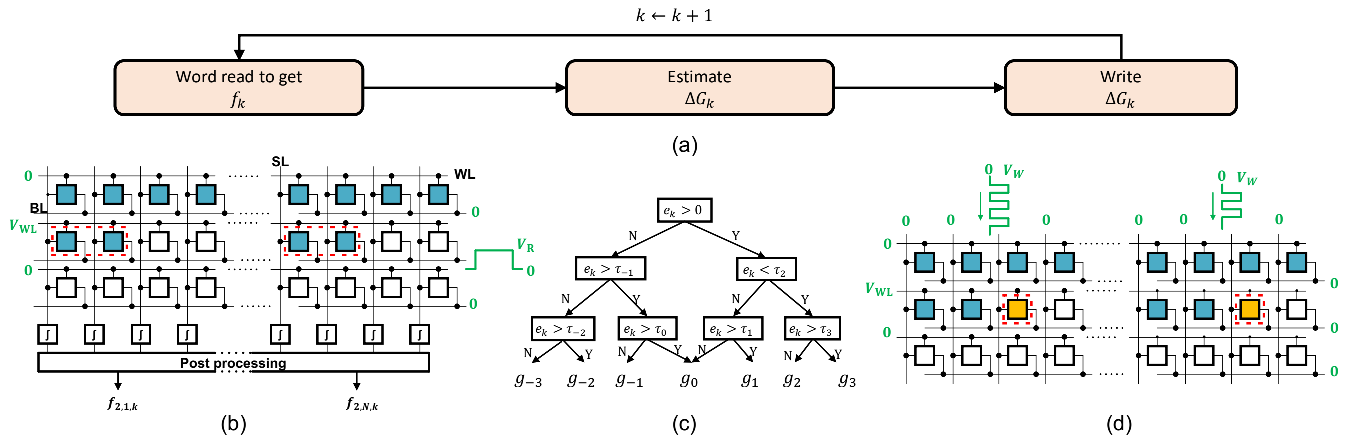


Figure 4: The proposed SWIPE method: (a) Flow diagram illustrating the three stages of SWIPE applied to the 2-nd row of a crossbar where $k = 2$, $N_c = 4$, and $B_c = 2$, (b) the word-read stage to obtain f_k, (c) decision tree to estimate ΔG, and (d) the write stage showing $1/N_c$-th row written.

writing the k-th BC pair of a given word $w_{i,j}$, SWIPE leverages the knowledge of the *composite* conductance value of the already written BC pairs 0 to $k-1$ (including their static variations) of that word. The knowledge and the parameters of distribution of device variations are leveraged to optimally choose the conductance value of the k-th BC pair to be written into. Note that BC pairs of all the words in a given row with the same significance are written in parallel, i.e., N_{col}/N_c BC pairs/row are programmed in every iteration. Furthermore, SWIPE programs each BC pair only once, making it significantly (5×-to-10×) more efficient compared to the standard program-verify techniques [2, 5, 7].

Formally described in Algorithm 1, each iteration k has following three steps (see Fig. 4(a)), where the cell indices i, j are dropped for simplicity:

Word-read: In this step, the composite value f_k of k already written BC pairs is read from the array (see Fig. 4(b)) as follows:

$$f_k = \frac{\alpha}{\Delta G_{\max}} \sum_{m=0}^{k-1} 2^{-mB_c} \Delta G_m \tag{7}$$

Note that, $f_k = 0$ for $k = 0$. Also, in absence of conductance variations, f_k is the $kB_c + 1$ bit quantized version of w, where w denotes the desired value of the word.

Estimate: In this step, f_k is used to determine the desired conductance state of the next BC pair $\Delta\tilde{G}_k$ (see Fig. 4(c)) as follows:

$$\Delta\tilde{G}_k = g_d \quad \text{if} \quad \tau_{d-1} \le e_k < \tau_d \tag{8}$$

where the error $e_k = \alpha^{-1}\Delta G_{\max} 2^{kB_c}(w - f_k)$ quantifies how far f_k is from the desired value w, $g_d \in \mathcal{G}$, and the thresholds $\mathcal{T} = \{\tau_d\}_{d=-L+1}^{d=L-1}$ are predetermined based on the variances of write noise per conductance state.

Write: The estimated conductance state $\Delta\tilde{G}_k$ is written to the k-th BC pair, as shown in Fig. 4(d).

Note that SWIPE can compensate accurately if the errors e_k in each iteration are maintained within the representable range of the unwritten LSB cells, i.e., if the $|e_k| \le 1$ for all k. Some devices in rare conditions observe large conductance changes even with small pulses. In these cases SWIPE may not be able to compensate completely due to a large changes in e_k. Such catastrophic effects are not acceptable when the array is used as a conventional memory as they lead to increased bit-errors. However, when the crossbar is used for implementation of MVM computations in DNNs, these rare effects minimally affect the classification accuracy due to the inherent error-tolerance of DNNs.

Since we assume an abrupt SET in the ReRAM device, SWIPE in Algorithm 1 assumes that the all the cells are set to high conductance state before the write process. Though, SWIPE reads only the cells that are previously written, it can be modified to read the the cells in the array. This modification will be useful if the cells are not set to high conductance state before the read process.

SWIPE programs one BC pair per word each iteration, thus requiring N_c iterations to write an entire row. To speedup the write process, SWIPE can be enhanced such that multiple BC pairs are programmed per iteration. However, doing so might result in lower accuracy as it increases the probability that the error e_k lies outside the compensation range.

3.3 Choosing Optimal Thresholds

The thresholds $\mathcal{T}\{\tau_d\}_{d=-L+1}^{d=L-1}$ need to be chosen to maximize SNR. At iteration k, given that f_k, the optimal value for $\Delta\tilde{G}_k$ is obtained by solving:

$$\begin{aligned}\Delta\tilde{G}_k &= \arg\min_{g_l \in \mathcal{G}} \mathbb{E}\left[(w - f_{k+1})^2 | f_k\right] \\ &= \arg\min_{g_l \in \mathcal{G}} \mathbb{E}\left[(e_k - g_l + \eta_{g,l})^2 | f_k\right] \\ &= \arg\min_{g_l \in \mathcal{G}} \left[(e_k - g_l)^2 + \sigma_{g,l}^2\right].\end{aligned} \tag{9}$$

where $\eta_{g,l}$ is a random variable capturing the conductance variations when the conductance state is g_l. Given the conductance variances $\{\sigma_{g,l}^2\}$ (4), the optimal thresholds $\mathcal{T}_{\text{opt}}$ are computed by solving (9) for every value of e_k. In practice, $\mathcal{T}_{\text{opt}}$ are pre-computed once and stored on-chip, therefore only (8) needs to be implemented on-chip to find $\Delta\tilde{G}_k$.

Algorithm 1 Single-Write In-memory Program-vErify (SWIPE).

```
Input: The target weight matrix $\mathbf{W}$, $\mathcal{G}$, $\{\sigma^2_{g,l}\}$, $B_c$, $N_c$.
Output: The crossbar array conductance $\Delta G_{i,j,k}$  $\forall$  $i, j, k$
1:  Estimate $\{\tau_i\}_{i=-L+1}^{L-1}$ by solving (9)
2:  for $i := 1$ to $N$ do
3:    parallel for $j := 1$ to $M$ do
4:      for $k := 0$ to $N_c - 1$ do
5:        Word-read row $i$ to obtain $f_{i,j,k}$
6:        $e_{i,j,k} \leftarrow \alpha^{-1}\Delta G_{\max}2^{kB_c}(w_{i,j} - f_{i,j,k})$
7:        Obtain $\Delta\tilde{G}_{i,j,k}$ as per (8)
8:        Write back $\Delta\tilde{G}_{i,j,k}$
9:      end for
10:   end parallel for
11: end for
```

3.4 Hardware Considerations

Implementing SWIPE in hardware requires minimal overhead since it can leverage the peripheral circuitry associated with existing crossbar based accelerators [3, 14].

Word-read: The *Word-read* operation to obtain f_k can be realized by employing one-hot encoded inputs during MVM (see Fig. 4(b)). However, the SL current needs to be amplified to match the ADC input range since one BC pair contributes to the SL current as compared to the standard MVM mode. Furthermore, the precision of the ADC should be large enough to make accurate comparisons required in (8).

Estimate: The threshold operation (8) requires one subtraction and B_c comparisons for each weight in a row (see Fig. 4(c)). This operation requires the storage of $2L - 1$ thresholds.

Write: The write operation in SWIPE requires N_c reads and write operations per row which makes it significantly efficient compared to conventional program-verify methods.

4 SIMULATION RESULTS

In this section, we discuss the effectiveness of SWIPE in enhancing robustness to write noise and device mismatches. First, we describe a custom simulation methodology used to reflect the circuit and device non-idealities in the system-level performance of the ReRAM crossbar. We study SWIPE in the context of DNN implementation on crossbars.

4.1 End-to-end Simulation Methodology

Figure 5 shows the evaluation methodology employed to quantify the system-level performance of DNNs on crossbars that integrate the circuit, architecture, and algorithmic parameters and design variables. We used the Stanford ReRAM Verilog-A model [12] for the ReRAM device characteristics, and commercial 22 nm FDSOI process to implement the access transistors. The design parameters used in the simulations are summarized in Table 1.

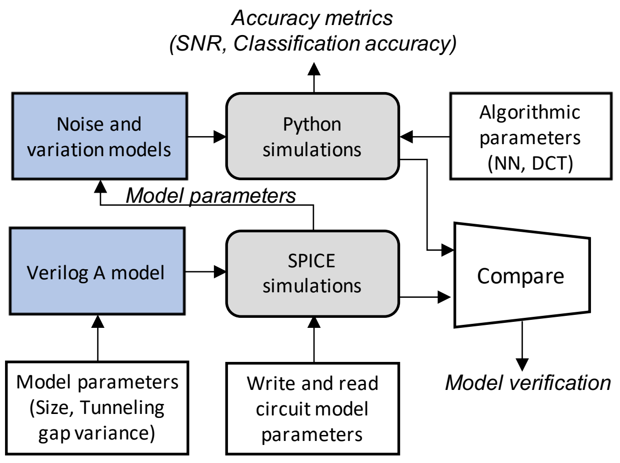

Accuracy metrics
(SNR, Classification accuracy)
Noise and variation models
Python simulations
Algorithmic parameters (NN, DCT)
Model parameters
Verilog A model
SPICE simulations
Compare
Model verification
Model parameters (Size, Tunneling gap variance)
Write and read circuit model parameters

Figure 5: Evaluation methodology.

Table 1: Device and circuit parameters used in simulations.

Parameter	Value	Parameter	Value
$G_{\max}$ (μS)	500	$G_{\min}$ (μS)	50
B_x (bits)	8	Crossbar Size	$64 \times 128N_c$
ADC prec. (bits)	7–to–10	B_w (bits)	2–to–9
SL Res. (Ω/Cell)	0.86	BL Res. (Ω/Cell)	0.47

4.1.1 ReRAM Variation Model. The cycle-to-cycle variations during the SET and RESET operations on the ReRAM device are modeled by introducing variation to the tunneling gap growth rate in the Verilog-A model as suggested in [12]. We modeled the device conductance distribution via circuit simulations of the ReRAM device, as shown in Fig. 2(b). The device-to-device variations can be modeled by introducing variation in the size of the ReRAM device.

4.1.2 Crossbar Array Model. We developed a Python model of the ReRAM crossbar that incorporates the effects of ReRAM device variations. We verified this model against circuit simulations of a 32×32 crossbar. For circuit simulations, we modeled ReRAM devices with equivalent resistors by appropriately choosing their conductance values. We used ideal voltage sources for inputs on BLs, ideal OpAmp-based current integrators on the SLs, and transistors based on 22 nm FDSOI technology.

We first compared the MVM operations on Python and circuit model by encoding conductance values based on 100 randomly chosen matrices and input vectors. We find that the Python model is within 0.2% and 1.1% of the circuit model with and without SL and BL parasitic resistances, respectively. The outputs of the Python and circuit models were compared to ideal expected outputs to verify the consistency of these models. The SL currents were quantized in post-processing assuming an ideal ADC. Next, we verified the data encoding procedures in SWIPE with circuit simulations. For each RESET operation, we sampled the BC conductance from ReRAM noise models obtained via Monte Carlo simulations of the Verilog A model.

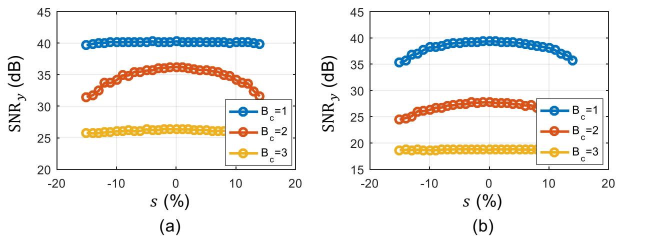


Figure 6: SNR$_y$ of 16-point DFT implementation on a $16 \times 32N_c$ crossbar with respect to the skew s in the thresholds with $B_w = 7$: (a) $\sigma_g/\Delta G_{\max} = 2.7\%$, and (b) $\sigma_g/\Delta G_{\max} = 10.8\%$. Input SNR was set at 41 dB.

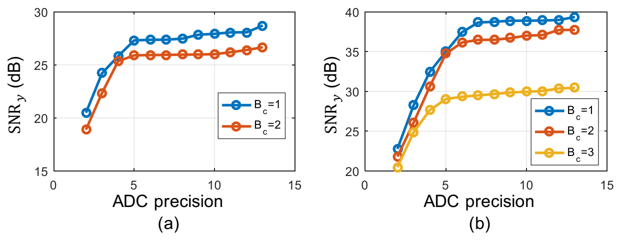


Figure 7: SNR$_y$ of 16-point DFT implementation on a $16 \times 32N_c$ crossbar with respect ADC precision during the word-read operation in SWIPE for: (a) $B_w = 5$, and (b) $B_w = 7$.

4.1.3 Application-level Simulations. The Python model was used for large scale statistical simulations of the application-level performance of the crossbar architecture. For the DNN simulations, we embed this Python model of the crossbar array into the PyTorch framework. The neural networks were mapped onto the crossbar using a naive network partitioning techniques, such as those presented in [15]. In order to ensure that the network can be easily quantized, we ensure the weights to lie between $[-1, +1]$ by clipping and scaling the weights during training. We study the impact of write noise on application-level accuracy metrics by varying $\sigma_g/\Delta G_{\max}$ where σ_g^2 is the conductance variance averaged over all the conductance states as follows:

$$\sigma_g^2 = \frac{1}{2L-1}\sum_l \sigma_{g,l}^2 \tag{10}$$

4.2 Optimality of SWIPE Thresholds

To demonstrate the optimality of the thresholds $\mathcal{T}$ obtained from (9), we observe the impact on the SNR of outputs y, SNR$_y$, under a skew s in each threshold $\tau \in \mathcal{T}$, as follows:

$$\tau_{\text{skew}} = \tau_{\text{opt}} + s\Delta g \tag{11}$$

Figure 6 shows that the the pre-ADC SNR of the output y_j (SNR$_y$) of 16-point discrete Fourier transforms (DFTs) is maximized when $s = 0$, $\tau_{\text{skew}} = \tau_{\text{opt}}$. For these simulations, the DFT weights and the inputs were encoded in fixed point and the DFT matrix parameters were written into the array using SWIPE. The input signal has 3 tones with randomly chosen frequencies, and the precision of the inputs was fixed at $B_x = 8$. The SNR is estimated by comparing the DFT outputs from the array with the outputs from an ideal floating point baseline.

In Fig. 6, we observe that SNR$_y$ is robust (flat) w.r.t. threshold variations for two scenarios: 1) quantization noise dominated scenario when σ_g and B_c are small as observed with $B_c = 1$ in Fig. 6(a); and 2) when conductance variations dominate, i.e., σ_g and B_c are high, as observed with $B_c = 3$ in Fig. 6(b).

Since, in SWIPE, the accuracy of the threshold operation (see Fig. 4(c)) is critical to maximize SNR$_y$, therefore, the ADC precision during the word-read operation needs to be high enough for accurate thresholding. Figure 7 shows that the ADC precision should be at least B_w for SNR$_y$ to be within 3 dB of the maximum achievable SNR$_y$. Note that the SNR improves for ADC precision higher than B_w as it enables a more accurate implementation of (8). Therefore, we choose ADC precision to be $B_w + 2$ in the rest of the chapter.

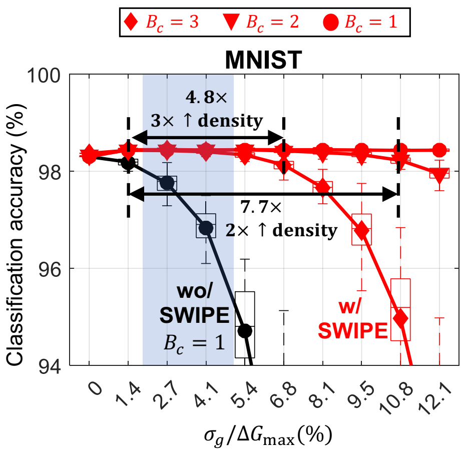
4.3 Robustness Improvements in DNN Implementations

We study two DNN architectures: (a) LeNet-300-10, and (b) an 8-layer CNN with 7 convolution layers[1] followed by an AveragePool and a fully connected layer for classification on the MNIST and the CIFAR-10 datasets, respectively. Both networks employ BatchNorm and a ClippedReLU nonlinearity at the output of each layer. The networks were trained with floating point weights and activations. The network weights and activations were quantized post training for inference using the Python model of the crossbar array. In order to reduce the impact of the outliers, the weights in each kernel were first clipped in the range $[-4\sigma_w, 4\sigma_w]$, where σ_w^2 is the varience of the weights in that kernel. This clipping is followed by uniform quantization with B_w bits. The scaling factors from the quantization scheme is absorbed into the batch-norm parameters. We choose weight precision $B_w = 7$, and the activation precision $B_x = 6$ to limit the accuracy loss with respect to a floating point design to $< 0.2\%$.

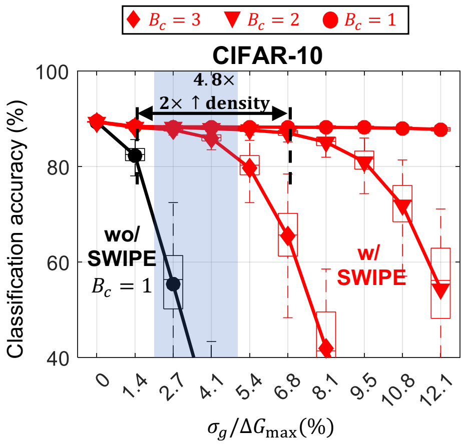
Figure 8 shows that SWIPE improves the robustness to conductance variations on both networks. The typical conductance variations in ReRAM due to write noise and device mismatch are about 2%-to-5% [17]. In this range, operating without SWIPE results in loss in accuracy with even with $B_c = 1$. In contrast, operating with SWIPE for $B_c = 1$ results *no loss in accuracy* in both networks for variations as high as $> 12\%$. These gains in robustness translates to gains in density, since SWIPE allows us to operate in the typical variation range with $< 1\%$ drop in accuracy for $B_c \leq 2$, and $B_c \leq 3$, on CIFAR-10 and MNIST, respectively. Thus, SWIPE allows us to simultaneously enhance robustness and density by 4.8×-to-7.7× and 2×-to-3×, respectively.

In NI-based training, Gaussian noise $\mathcal{N}(0, \sigma_{\text{NI}}^2)$ is added to network weights during the feed-forward pass of the back-propagation iterations during training. NI-based training has been observed to improve robustness to line resistance [8], and to device mismatch and conductance variation in [13, 19]. We studied the impact of noise injection, by training networks with different values of σ_{NI},

[1]The 7 convolutional layers are (3C64S1)×3–(3C128S2)–(3C128S1)–(3C256S2)–(3C512S1), where (aCbSc) indicates $a \times a$ kernel, b output channels, and stride c.

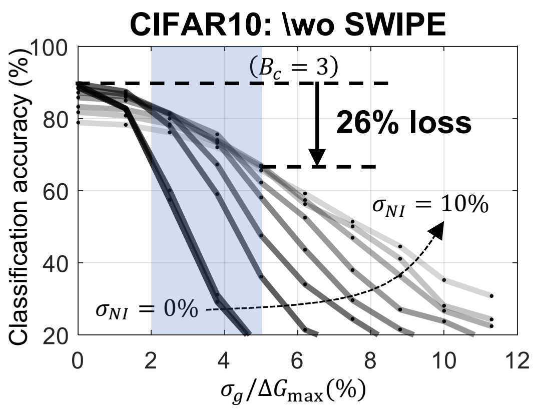


(a)

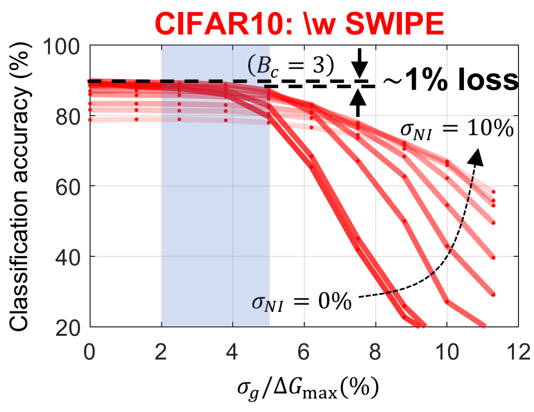


(b)

Figure 8: Accuracy in the presence of SWIPE with respect to average BL conductance variations for: (a) LeNet-300-100 on the MNIST dataset, and (b) the 8-layer CNN on the CIFAR-10 dataset. The box plots show the spread in network accuracy over 100 iterations. The shaded region marks the typical conductance variation range [17].



(a)



(b)

Figure 9: Accuracy of the 8-layer CNN for the CIFAR-10 dataset with NI-based trained where σ_{NI} ranges from 0% to 10% in steps of 1.25%, and with $B_c = 3$: (a) without SWIPE, and (b) with SWIPE. The shaded region marks the typical conductance variation range [17].

ranging from 0% to 10%. Though NI-based training improves robustness with increase in the training noise variance σ^2_{NI} (see Fig. 9(a)), we observe that it degrades the maximum achievable accuracy at $\sigma_g/\Delta G_{\max} = 0$. Furthermore, NI-based training still results in 26% accuracy loss in the typical conductance variation range (shaded) (2%-to-5%) on the CIFAR-10 dataset. In contrast, Fig. 9(b) shows that augmenting SWIPE applied on the weights obtained busing NI-based training results in $< 1\%$ loss in accuracy with $B_c = 3$ within the typical conductance variation range. Thus augmenting SWIPE with NI-based training will reduce the need to use σ_{NI}.

Augmenting SWIPE with NI-based training also enables the designs using crossbar-based in-memory architectures that are simultaneously robust to both read and write noise. We studied the impact of read noise by injecting additional random fluctuations in the device conductances for every MVM on the crossbars. Figure 10(a) shows that NI-based training improves robustness to read noise in the absence of write noise. However, in the presence of write noise, the classification accuracy decreases dramatically by 48% (see Fig. 10(b)) on the CIFAR-10 network. This loss in accuracy due to write noise is recovered by using SWIPE, as shown in Fig. 10(c).

Python simulations reveal that for $B_w = 7$, and for $\sigma_g/\Delta G_{\max} = 2.7\%$ to $\sigma_g/\Delta G_{\max} = 6.8\%$, conventional program-verify methods need up to 5-to-10 iterations implying that the use of SWIPE will result in 5×-to-10× lower write costs since it programs the array in a single pass.

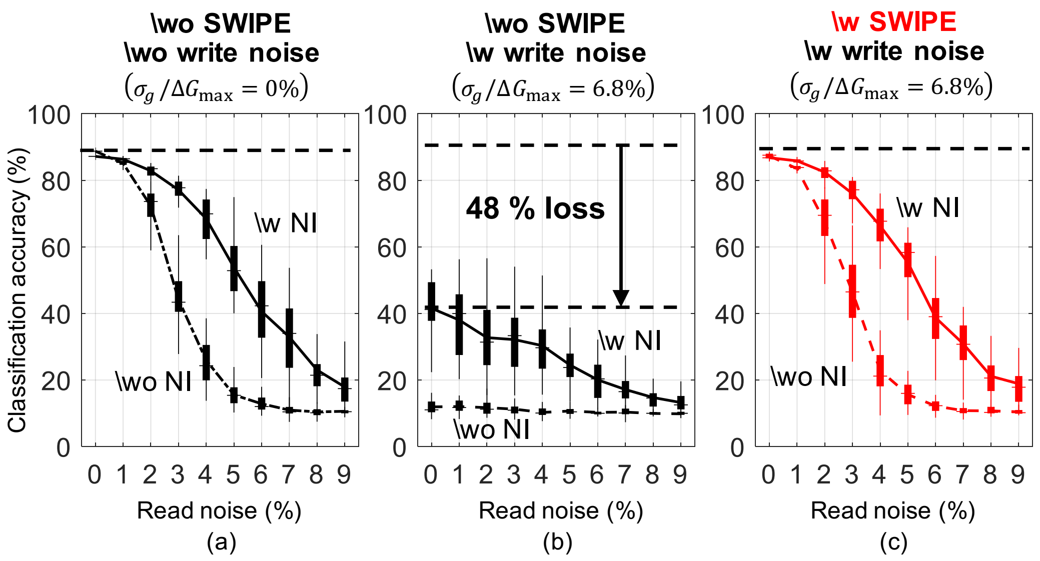


Figure 10: Accuracy of the 8-layer CNN on the CIFAR-10 dataset in the presence of read noise with $B_c = 2$: (a) without SWIPE and write noise ($\sigma_g/\Delta G_{\max} = 0\%$), (b) without SWIPE and with write noise ($\sigma_g/\Delta G_{\max} = 6.8\%$), and (c) with SWIPE and write noise ($\sigma_g/\Delta G_{\max} = 6.8\%$); $\sigma_{\text{NI}} = 5\%$ was used to train with NI. Box plots show the spread in accuracy over 100 iterations.

5 CONCLUSIONS

This paper presents SWIPE method to enable efficient and accurate writes in the presence of write noise and device mismatch for in-memory crossbars. We demonstrate that SWIPE enables DNN implementation on ReRAM crossbars with $< 1\%$ loss under typical values of write noise and device variations. Augmenting SWIPE with NI-based training enables DNN implementation on crossbars that are simultaneously robust to both write and read noise. SWIPE can also be used in other applications to enhance the SNR of MVMs implemented on resistive crossbars.

APPENDIX

In this appendix, we derive the SNR bound (6). Each dot product y_j is broken down into N_c dot-products $\{z_{j,k}\}_{k=0}^{N_c-1}$ across the columns based on (3). The conductance variations result in the accumulated noise in $\gamma_{j,k}$ added to $z_{i,k}$ as shown in (5), where $\gamma_{j,k}$ is as follows:

$$\gamma_{j,k} = \frac{\alpha}{\Delta G_{\max} V_{\max}} \sum_{i=1}^{N} \eta_{i,j,k} V_i \tag{12}$$

Assuming the precision of the ADC is sufficiently large, the total noise η_{y_j} in the final dot product y_j is given by:

$$\eta_{y_j} = \sum_{k=0}^{N_c-1} 2^{-kB_c} \gamma_{j,k} \tag{13}$$

Assuming conductance variations across the array zero mean independent identically distributed, and the weights are independent uniformly distributed, the final noise power can be estimated as:

$$\begin{aligned}\mathbb{E}\left[\eta_{y_j}^2\right] &= \sum_{k=0}^{N_c-1} 2^{-2kB_c} \mathbb{E}\left[\gamma^2\right] \\ &= \left(\frac{1-2^{-2N_cB_c}}{1-2^{-2B_c}}\right) \mathbb{E}\left[\gamma^2\right]\end{aligned} \tag{14}$$

Furthermore, from (12) we have the following:

$$\begin{aligned}\mathbb{E}\left[\gamma_{j,k}^2\right] &= \left(\frac{\alpha}{\Delta G_{\max} V_{\max}}\right)^2 \sum_{i=1}^{N} \mathbb{E}\left[\eta_{i,j,k}^2 V_i^2\right] \\ &= \left(\frac{\alpha}{\Delta G_{\max} V_{\max}}\right)^2 \sum_{i=1}^{N} \mathbb{E}\left[\eta_{i,j,k}^2\right] \mathbb{E}\left[V_i^2\right] \\ &\geq \left(\frac{\alpha}{\Delta G_{\max}}\right)^2 N \sigma_{g,\min}^2 \mathbb{E}\left[x^2\right]\end{aligned} \tag{15}$$

From (3), (14), and (15), the upper bound (5) on the output SNR is obtained:

$$\text{SNR}_y \leq \left(\frac{\Delta G_{\max}}{\alpha}\right)^2 \frac{\mathbb{E}\left[y^2\right](1-2^{-2B_c})}{N\sigma_{g,\min}^2 \mathbb{E}\left[x^2\right](1-2^{-2N_cB_c})} \tag{16}$$

ACKNOWLEDGMENTS

This work was supported by the Center for Brain-Inspired Computing (C-BRIC) funded by the Semiconductor Research Corporation (SRC) and the Defense Advanced Research Projects Agency (DARPA).

REFERENCES

[1] Amogh Agrawal, Chankyu Lee, and Kaushik Roy. 2019. X-CHANGR: Changing Memristive Crossbar Mapping for Mitigating Line-Resistance Induced Accuracy Degradation in Deep Neural Networks. *arXiv preprint arXiv:1907.00285* (2019).

[2] Fabien Alibart, Ligang Gao, Brian D Hoskins, and Dmitri B Strukov. 2012. High precision tuning of state for memristive devices by adaptable variation-tolerant algorithm. *Nanotechnology* 23, 7 (2012), 075201.

[3] Aayush Ankit, Izzat El Hajj, Sai Rahul Chalamalasetti, Geoffrey Ndu, Martin Foltin, R Stanley Williams, Paolo Faraboschi, Wen-mei W Hwu, John Paul Strachan, Kaushik Roy, and others. 2019. PUMA: A programmable ultra-efficient memristor-based accelerator for machine learning inference. In *International conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. ACM, 715–731.

[4] Aayush Ankit, Abhronil Sengupta, and Kaushik Roy. 2017. Trannsformer: Neural network transformation for memristive crossbar based neuromorphic system design. In *International Conference on Computer-Aided Design (ICCAD)*. IEEE, 533–540.

[5] Ming Cheng, Lixue Xia, Zhenhua Zhu, Yi Cai, Yuan Xie, Yu Wang, and Huazhong Yang. 2018. TIME: A training-in-memory architecture for RRAM-based deep neural networks. *IEEE Transactionss on Computer-Aided Design of Integrated Circuits and Systems* 38, 5 (2018), 834–847.

[6] P. Chi, S. Li, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, and Y. Xie. 2016. PRIME: A Novel Processing-in-Memory Architecture for Neural Network Computation in ReRAM-Based Main Memory. In *International Symposium on Computer Architecture (ISCA)*. IEEE/ACM, 27–39.

[7] Ligang Gao, Pai-Yu Chen, and Shimeng Yu. 2015. Programming protocol optimization for analog weight tuning in resistive memories. *IEEE Electron Device Letters* 36, 11 (2015), 1157–1159.

[8] Zhezhi He, Jie Lin, Rickard Ewetz, Jiann-Shiun Yuan, and Deliang Fan. 2019. Noise Injection Adaption: End-to-End ReRAM Crossbar Non-ideal Effect Adaption for Neural Network Mapping. In *Design and Automation Conference (DAC)*. IEEE/ACM, 57.

[9] Mark Horowitz. 2014. Computing's energy problem (and what we can do about it). In *IEEE International Solid-State Circuits Conference (ISSCC)*. IEEE, 10–14.

[10] Miao Hu, Catherine E Graves, Can Li, Yunning Li, Ning Ge, Eric Montgomery, Noraica Davila, Hao Jiang, R Stanley Williams, J Joshua Yang, and others. 2018. Memristor-based analog computation and neural network classification with a dot product engine. *Advanced Materials* 30, 9 (2018), 1705914.

[11] Miao Hu, John Paul Strachan, Zhiyong Li, Emmanuelle M Grafals, Noraica Davila, Catherine Graves, Sity Lam, Ning Ge, Jianhua Joshua Yang, and R Stanley Williams. 2016. Dot-product engine for neuromorphic computing: Programming 1T1M crossbar to accelerate matrix-vector multiplication. In *Design Automation Conference (DAC)*. IEEE, 1–6.

[12] Zizhen Jiang, Yi Wu, Shimeng Yu, Lin Yang, Kay Song, Zia Karim, and H-S Philip Wong. 2016. A compact model for metal–oxide resistive random access memory with experiment verification. *IEEE Transactions on Electron Devices* 63, 5 (2016), 1884–1892.

[13] Yun Long, Xueyuan She, and Saibal Mukhopadhyay. 2019. Design of Reliable DNN Accelerator with Un-reliable ReRAM. In *Design, Automation and Test in Europe Conference (DATE)*. IEEE/ACM, 1769–1774.

[14] Ali Shafiee, Anirban Nag, Naveen Muralimanohar, Rajeev Balasubramonian, John Paul Strachan, Miao Hu, R Stanley Williams, and Vivek Srikumar. 2016. ISAAC: A Convolutional Neural Network Accelerator with In-Situ Analog Arithmetic in Crossbars. In *International Symposium on Computer Architecture (ISCA)*. IEEE/ACM, 14–26.

[15] Linghao Song, Xuehai Qian, Hai Li, and Yiran Chen. 2017. Pipelayer: A pipelined ReRAM-based accelerator for deep learning. In *International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 541–552.

[16] Xiaoyu Sun and Shimeng Yu. 2019. Impact of Non-Ideal Characteristics of Resistive Synaptic Devices on Implementing Convolutional Neural Networks. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems (JETCAS)* 9, 3 (2019), 570–579.

[17] Wei Wu, Huaqiang Wu, Bin Gao, Peng Yao, Xiang Zhang, Xiaochen Peng, Shimeng Yu, and He Qian. 2018. A methodology to improve linearity of analog RRAM for neuromorphic computing. In *IEEE Symposium on VLSI Technology (VLSI-T)*. IEEE, T103–T104.

[18] Shimeng Yu, Ximeng Guan, and H-S Philip Wong. 2012. On the switching parameter variation of metal oxide RRAMâĂŤpart II: Model corroboration and device design strategy. *IEEE Transactions on Electron Devices* 59, 4 (2012), 1183–1188.

[19] Bonan Zhang, Lung-Yen Chen, and Naveen Verma. 2019. Stochastic data-driven hardware resilience to efficiently train inference models for stochastic hardware implementations. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 1388–1392.

[20] Zhenhua Zhu, Jilan Lin, Ming Cheng, Lixue Xia, Hanbo Sun, Xiaoming Chen, Yu Wang, and Huazhong Yang. 2018. Mixed size crossbar based RRAM CNN accelerator with overlapped mapping method. In *International Conference on Computer-Aided Design (ICCAD)*. IEEE, 1–8.

[21] Zhenhua Zhu, Hanbo Sun, Yujun Lin, Guohao Dai, Lixue Xia, Song Han, Yu Wang, and Huazhong Yang. 2019. A Configurable Multi-Precision CNN Computing Framework Based on Single Bit RRAM. In *Design and Automation Conference (DAC)*. IEEE/ACM, 56.