RWriC: A Dynamic Writing Scheme for Variation Compensation for RRAM-based In-Memory Computing

Yucong Huang[1,2], Jingyu He[1], Kwang-Ting Cheng[1], Chi-Ying Tsui[1], Terry Tao Ye[2*]
[1]Hong Kong University of Science and Technology (HKUST), Hong Kong SAR
[2]Southern University of Science and Technology (SUSTech), Shenzhen, China

ABSTRACT

RRAM-based compute-in-memory (CIM) suffers from programming variation issues, specifically device-to-device variation (DDV) and cycle-to-cycle variation (CCV), which can have a detrimental impact on inference accuracy. To address these variation issues, we propose RWriC, a dynamic **Wri**ting scheme for variation **C**ompensation for **R**RAM-based CIM. RWriC sequentially programs the weights, implemented by multiple RRAM cells, starting from the high significance cell (HSC) and moving towards the low significance cell (LSC). This approach leverages the knowledge of current cumulative errors and the programming targets (PTs) of other RRAM cells to dynamically adjust the PT of the RRAM currently under programming. By shifting the PT of HSC, RWriC enables the LSC to compensate for the programming errors of the HSC. Moreover, when the variation is substantial, RWriC allows the magnitude of LSC to be scaled up, providing an even wider compensation range. Through the combined application of the shifting and scaling techniques, experimental results show that the inference accuracy for ResNet50 on the CIFAR-10 dataset only drops by 0.9% under 18% device variation. In comparison to the conventional writing scheme, our RWriC approach achieves a 5-11x improvement in variation robustness for ResNet50 and Yolov8 across different tasks.

1 INTRODUCTION

Resistive Random Access Memory (RRAM) has attracted significant research attention owing to its remarkable features, including non-volatility, multiple-level cell (MLC) support, high speed, and compatibility with complementary metal-oxide-semiconductor (CMOS) technology [1, 2]. However, RRAM devices encounter a substantial challenge in the form of conductance variation, leading to deviations between the target conductance and the actual values. These variations adversely affect neural network (NN) accelerator inference accuracy, making it a major hurdle for the widespread adoption of RRAM in CIM applications. The variations in RRAM can be broadly classified into device-to-device variation (DDV) and cycle-to-cycle variation (CCV) [1]. DDV arises from differences in conductance between two RRAM devices located in different positions but subjected to the same programming pulses. CCV refers to conductance differences observed within a single RRAM device when the same programming pulses are applied at different times.

Previous studies have been conducted to mitigate the impact of variations of RRAM on NN inference accuracy. For software-based methods, [3, 4, 5] introduce random variations during training, and [6] retrains layers when transferring weights to RRAM. However, injecting noise during training may fail to converge to high accuracy, especially when dealing with simple NN architectures or significant variations. The retraining approach entails additional training epochs, which are time-consuming. For hardware-based methods, these approaches involve introducing additional hardware components to compensate for variations in RRAM either through compensating the inputs [7] or weights [8, 9]. However, this leads to increased area and power overhead for the RRAM macro. Recently, a sequential writing scheme, SWIPE [10], aims to address variations by leveraging the bit-slicing principle in the RRAM crossbar. This approach involves encoding cumulative errors from already-programmed cells into subsequent unprogrammed cells for compensation. However, this method has limitations in offering comprehensive compensation performance, primarily due to the difficulty of encoding large errors into unprogrammed cells, which are less significant in the weight value magnitudes. Unprogrammed cells can easily have their programming targets pushed out of range, leading to truncation at the minimum or maximum boundary. Preventing cell truncation in this sequential writing scheme is crucial for enhancing compensation performance.

To mitigate accuracy loss in in-memory computing caused by variations in RRAM devices, in this work, we introduce RWriC, a dynamic writing scheme for variation compensation. RWriC is implemented with minimal additional overhead in RRAM CIM macros, enabling the dynamic adjustment of cell values to prevent cell truncation during programming. By shifting the programming target (PT) of high significance cell (HSC) and scaling the magnitude of low significance cell (LSC), errors detected in HSC can be effectively compensated by re-targetting the conductance of LSC. This approach significantly reduces multiply-accumulate (MAC) calculation errors and greatly enhances NN inference accuracy on RRAM CIM macros. Our RWriC scheme offers several key advantages. Firstly, it is implemented during the writing process, incurring only a one-time non-recurring cost. Secondly, RWriC is a fine-grained compensation technique that addresses errors in individual cells, enhancing the robustness of the NN against substantial variations. Thirdly, RWriC necessitates minimal modifications and introduces negligible hardware overhead to RRAM macros. Lastly, RWriC does not require any training modifications or retraining overhead. The primary contributions of our work are as follows:

1) We introduce a novel shifting PT writing scheme to prevent RRAM cell truncation during the subsequent sequential writing,

*Corresponding author: yet@sustech.edu.cn

DAC '24, June 23–27, 2024, San Francisco, CA, USA
© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0601-1/24/06…$15.00
https://doi.org/10.1145/3649329.3656227

thereby improving compensation effectiveness compared to existing methods.

2) We further propose a novel scaling magnitude scheme for scenarios with even larger device variations. The column magnitude is determined based on the errors from programmed cells, ensuring that the programming target remains within the available device range. We also present an optimization approach to identify the optimal scaling factor for each column.

3) We present a comprehensive scheme that integrates the above two techniques. Experimental results demonstrate that our approach offers superior robustness under substantial variations.

2 PRELIMINARIES

RRAM is an emerging non-volatile memory technology that utilizes the alterations in the resistance of a conductive filament to store information. This is achieved by applying voltage in different directions across two terminals, resulting in different resistance values. The process of changing the conductance is called the write operation. To retrieve data from an RRAM device, a small, non-destructive voltage is applied across its terminals, and the corresponding current is measured.

When RRAM is configured in a crossbar architecture, it can efficiently perform MAC operations, which are fundamental in neural networks, with a high degree of parallelism. Input data is encoded as read voltages on the bit line. According to Ohm's law and Kirchhoff's law, the current passing through each device is collected at the source line, and the sum of these currents represents the dot product of the input vector and the weight vector stored in an RRAM column. However, due to deviations present in RRAM devices, errors are introduced at the output. These deviations are represented by the error term $e_{i,j}$ in Eq. 1, where m is the total number of rows, y_j is the output of j^{th} column, x_i is the input of i^{th} row, and $w_{i,j}$ is the corresponding weight. To capture the output result, the current is converted into voltages, which are then sampled by a transimpedance amplifier (TIA) and a sample-and-hold (S&H) circuit. An analog-to-digital converter (ADC) is then used to convert the output data into a digital form. To reduce the digital-to-analog converter (DAC) overhead, the input n-bit vector is broken down into n 1-bit segments, and the bits are applied serially to the crossbar in multiple cycles.

$$y_j = \sum_{i=0}^{m-1} x_i (w_{i,j} + e_{i,j}) \tag{1}$$

In scenarios where multi-bit weights are used and the resolution of a single RRAM device is insufficient to map the weight, the weights are divided into multiple segments, with each segment stored in a single cell within the same row. The partial sums obtained from the corresponding columns are then accumulated in a shift-and-add (S+A) block.

3 VARIATION-AWARE WRITING SCHEME

3.1 Overview

To represent a synaptic weight quantized with high precision, e.g. higher than 8-bit, multiple RRAM cells are needed to encode the weight due to the limited resolution of a single RRAM cell. For instance, a B_w-bit synaptic weight is sliced and mapped to N_c adjacent cells in the same row, with each cell representing B_c bits. The value of each sliced weight is encoded into the conductance of the corresponding RRAM cell, as depicted in Fig. 1. The scaling magnitude (M_k) of the k^{th} column is given by the following equation:

$$M_k = 2^{(N_c-1-k)B_c} \tag{2}$$

Thus, a quantized synaptic weight (W) is given by:

$$W = \sum_{k=0}^{N_c-1} M_k w_k, \tag{3}$$

where w_k is the quantized sliced weight of the k^{th} column. Let G_{max} and G_{min} be the maximum and minimum conductance of an RRAM cell, respectively, the target conductance g_k representing a quantized sliced weight w_k is given by:

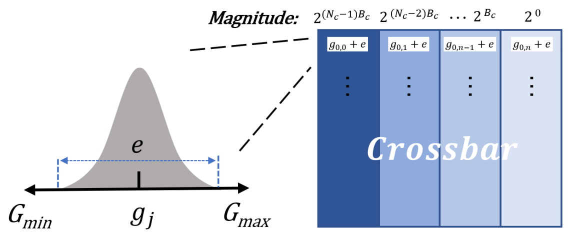
$$g_k = \frac{G_{max} - G_{min}}{2^{B_c} - 1} w_k + G_{min} \tag{4}$$

and w_k can be expressed as a simple linear function of g_k with a slope α and a bias β:

$$w_k = \alpha \cdot g_k + \beta \tag{5}$$

Due to DDV and CCV, the conductance of RRAM cells cannot be accurately programmed to the target value and exhibits a Gaussian-like variation across its conductance range as presented in [11, 12, 13]. The error in conductance, denoted as e, follows a normal distribution as shown Fig. 1(a). Given a target conductance g_j, the actual conductance $\tilde{g}_j$ after programming is then equal to:

$$\tilde{g}_j = g_j + e_j \quad e_j \sim N\left(0, \sigma^2\right). \tag{6}$$



(a) (b)

Figure 1: (a) Gaussian-like error distribution centered on target conductance. (b) Synaptic weights are stored in the RRAM crossbar in the form of conductance.

To tackle this variation, a variation-aware writing scheme was proposed in [10], where RRAM cells are sequentially programmed from the most significant cell to the least significant cell. After programming the k^{th} cell, the actual accumulated value ($\widetilde{W}_k$) from all the programmed cells can be expressed as:

$$\widetilde{W}_k = \sum_{j=0}^{N_c-1-k} M_j \left(\alpha \cdot \tilde{g}_j + \beta\right) \tag{7}$$

where $\tilde{g}$ is the actual conductance after programming. To offset the errors arising from the programming variations, the PT of the next cell, the $(k+1)^{th}$ cell, is adjusted and set to be the difference between the target accumulated value (i.e. W_{k+1}) after the $(k+1)^{th}$ cell is programmed and the actual accumulated programmed value (i.e. $\tilde{W_k}$) after the k^{th} cell is programmed, as formulated in the following:

$$g_{k+1} = \alpha^{-1}\left(W_{k+1} - \widetilde{W_k} - \beta\right) \quad (8)$$

Under ideal conditions, where each cell can perfectly compensate for the errors generated from the programming of the preceding cell, error accumulation does not occur and the total error of a synaptic weight is solely attributed to the least significant cell. Given that the magnitude of the least significant cell is the smallest, the overall error in representing the weight is significantly reduced.

However, in reality, adjusting the PT of RRAM cells is not able to compensate for errors perfectly due to the limited range of the conductance values that can be used to represent a weight. If the PT attempts to exceed the boundary of the device conductance range, it will be truncated to the minimum or maximum value. For instance, as illustrated in Fig. 2(a), if an RRAM cell is used to represent a 2-bit value with an available range of [0, 3], the original PT for the HSC and LSC are the conductance values corresponding to 2.0 and 0.0, respectively. After programming the HSC and if there is a variation in the programmed conductance that corresponds to a +0.2 weight error, which requires compensation based on the writing scheme proposed by SWIPE [10]. The subsequent LSC theoretically necessitates a PT shift from 0.0 to -0.8 in order to compensate for the error to preserve the current accumulated weight value. However, this PT shift surpasses the available range. Consequently, the PT of LSC can only be truncated to 0.0, leading to a significant deviation from the original target value in the programmed result. To address these challenges, we propose methodologies for shifting and scaling PT to prevent PT truncation and enhance the effectiveness of writing compensation.

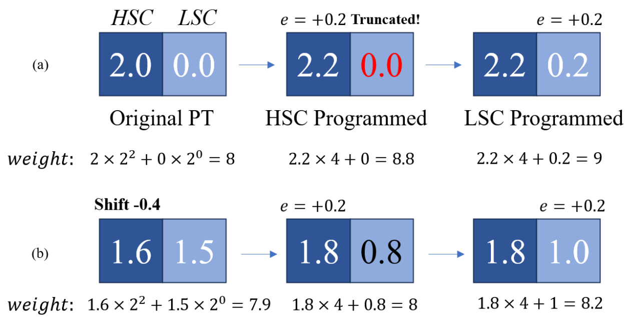


Figure 2: (a) Example of cell truncation in sequential writing scheme. (b) Example of *PT_Shift*.

3.2 Shifting Programming Target

To prevent cell truncation during sequential writing, particularly when the cell's original target is at either end of the maximum and minimum range, we propose the Shifting Programming Target (*PT_Shift*) scheme. Before programming the HSC, its PT is dynamically shifted upwards or downwards before writing based on the original PT of LSC. This shifting of HSC will cause the PT of LSC to shift in the opposite direction, thereby expanding the available compensation range to minimize the chance for PT truncation during the actual programming for compensation. Fig. 2(b) illustrates an example. When the PT of HSC shifts from 2.0 to 1.6, the original target PT of the LSC must shift from 0.0 to 1.5 to maintain consistent weight values. With this shifting, under the situation where there is a +0.2 error when programming the HSC, the LSC still has room to adjust its PT to compensate for the error by shifting the PT 1.5 to 0.8. By providing this additional room for compensation, the chance of PT truncation is minimized, resulting in a significant reduction in the final programmed error. Similarly, if the original PT of LSC is already at the maximum value, the PT of HSC can be increased to move the target PT of the LSC towards the center of the conductance range to provide more room for compensation.

3.3 Scaling Programming Target

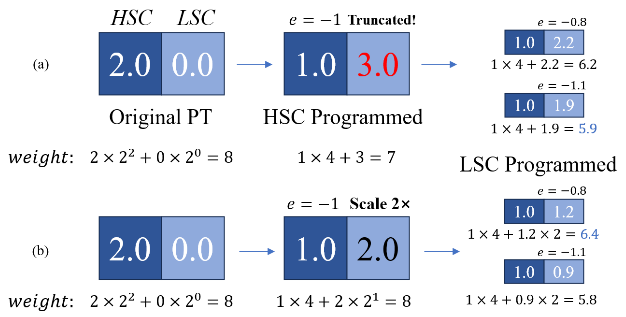


Figure 3: (a) Example of cell truncation under large variation. (b) Example of *PT_Scale*.

To further enhance compensation effectiveness in the presence of significant device variations, we introduce the Scaling Programming Target (*PT_Scale*) scheme. In situations where the errors from the HSC exceed the entire representation range of the LSC, as given by $(G_{max} - G_{min})/2^{Bc}$, PT truncation occurs at the LSC, as illustrated in Fig. 3(a). For example, with an error of -1.0 from the HSC, the PT of LSC theoretically needs to shift from 0.0 to 4.0, but due to truncation, it is limited to 3.0. In this case, the weight value can only target 7.0 instead of 8.0. To address this issue, we propose dynamically scaling up the magnitude of LSC to increase the compensation range, ensuring that it is sufficient to accommodate the shifted programming target. As shown in Fig. 3(b), scaling up the LSC's magnitude from 2^0 to 2^1, reducing the programming target from 4.0 to 2.0, allowing it to remain within the available range and avoiding truncation. Note that the scaling factor can be larger than 2 if the PT of LSC still gets truncated even after scaling by 2. In the example shown in Fig. 3, if the programming error of LSC is -0.8, applying *PT_Scale* reduces the overall error in the weight compared to the case where the PT of LSC is truncated. However, scaling up the LSC's magnitude also magnifies the programming error at the LSC, which could potentially result in a worse outcome. For instance, as depicted in Fig. 3, if the programming error of the LSC is -1.1, applying *PT_Scale* with a scaling factor of 2 results in an even larger error in the overall weight compared to the case not using *PT_Scale*. The decision of whether to employ *PT_Scale* and the choice of an appropriate scaling factor become critical during the writing process. The method of making this decision will be discussed in the next subsection. It is also worth noting that scaling is performed in multiples of 2, and the same scaling factor is applied for the whole column. The scaling factor for each column is stored in registers. This enables the re-scaling and restoration of the weights to their correct values after the ADC process by simply controlling the shifting bits for addition.

3.4 Parameters Optimization

This section will discuss how to set the parameters for *PT_Shift* and *PT_Scale* to maximize accuracy restoration in the presence of variations. Specifically, we will discuss how to set the shift distance (sd) of the *PT_Shift* and the scaling factor (sc) of the *PT_Scale*.

For *PT_Shift*, determining the appropriate sd for HSC is challenging because the exact programming error is not yet known. If sd is too small, the LSC would still be truncated after incorporating the error obtained from the HSC in setting the PT. Conversely, if it is too large, it may lead to truncation in the opposite direction. From the perspective of the LSC, the best strategy is to maintain the original PT at the midpoint of the conductance range to provide room for compensating either positive or negative errors from the HSC. To cater for this, the sd of HSC (k^{th} cell) should be set based on the original target PT of LSC (g_{k+1}) and the mid-point conductance value (g_{mid}) and is given by:

$$sd_k = \frac{g_{k+1} - g_{mid}}{2^{B_c}}. \tag{9}$$

The adjusted PT g'_k after conducting *PT_Shift* and incorporating error from programmed cells is represented by:

$$g'_k = g_k + sd_k - 2^{B_c} e_{k-1} \quad g'_k \in [G_{\min}, G_{\max}] \tag{10}$$

For *PT_Scale*, simply scaling the magnitude of the column is not sufficient to alleviate the situation when the PT truncation occurs at the negative end, where the PT is smaller than G_{min}. To tackle this issue, we introduce an additional step before scaling by adding a midpoint conductance value to the PT, as depicted in Eq. 11, where g''_k represents the adjusted PT after scaling with a scaling factor, sc, of 2.

$$g''_k = \frac{g_k + g_{mid}}{2} \quad g''_k \in [G_{\min}, G_{\max}] \tag{11}$$

If g''_k still exceeds G_{min} or G_{max} after adjustment, we can perform another scaling operation by a factor of 2 to bring it back within the permissible range.

The actual composite value ($\widetilde{w}''_k$) of the k^{th} weight after shifting g_{mid} and scaling with 2 is represented by:

$$\widetilde{w}''_k = 2M_k\left(\alpha\left(g''_k + e_k\right) + \beta\right) = \widetilde{w}_k + M_k\left(\alpha\left(g_{mid} + e_k\right) + \beta\right) \tag{12}$$

The *PT_Scale* scheme introduces an additional error and a constant bias to the weight. The constant weight bias can be subtracted from the final accumulated results in the digital domain. However, the additional error can be significantly amplified if a scaling factor sc is applied to avoid cell truncations. Hence, applying scaling has a different effect on the weight representation accuracy, and determining an optimal value for the scaling factor sc in an RRAM column is crucial. To determine the optimal scaling factor, we consider a target value of weight (g_k) to be programmed and the variation distribution of the programmed weight. By computing the expected errors under different scaling factors, we can identify the scaling factor that yields the minimum expected errors. This selected scaling factor is then chosen as the optimal value. Next, we will discuss how to obtain the expected error.

When the PT is smaller than G_{min} or larger G_{max}, it will be truncated to G_{min} or G_{max}, respectively. Thus, the probability distribution of the programmed conductance does not actually follow a Gaussian distribution centered on the target PT, especially when the device variation is significant or the PT is near the boundary, as shown in Fig. 4(a). Therefore, the expected error for a given target PT cannot be accurately calculated using the conventional method based on normal distribution. In theory, for a single cell, the probability of the actual conductance is assumed to follow Gaussian distribution and is given by Eq. 13. With truncation, the actual error distribution can be divided into three intervals: 1) When the PT value is smaller than G_{min} and is truncated to G_{min}. 2) When the PT value is within the permissible range. 3) When the PT value is larger than G_{max} and is truncated to G_{max}. To obtain the expected error for each target PT value g_k, we integrate Eq. 14. The expected errors corresponding to different target g_k values are illustrated in Fig. 4(b), indicating that the minimum expected error occurs when the PT is at the edges of the permissible range. Since each cell in a column has different target PTs, and the whole column needs to share the same scaling factor sc, we select the cell with the median expected error to represent the entire column. For each sc value, including $sc = 1$, we compute the median expected error of the column. Then, the scaling factor value that yields the smallest median expected error is chosen as the actual scaling factor.

$$f_k(g) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(g-g_k)^2/2\sigma^2} \tag{13}$$

$$E_{rr}(g_k) = \begin{cases} \int_{-\infty}^{G_{\min}} f_k(g)\cdot(G_{\min} - g_k)\,dg & \text{if } g_k \in (-\infty, G_{\min}) \\ \int_{G_{\min}}^{G_{\max}} f_k(g)\cdot|g - g_k|\,dg & \text{if } g_k \in [G_{\min}, G_{\max}] \\ \int_{G_{\max}}^{\infty} f_k(g)\cdot(g_k - G_{\max})\,dg & \text{if } g_k \in (G_{\min}, +\infty) \end{cases} \tag{14}$$

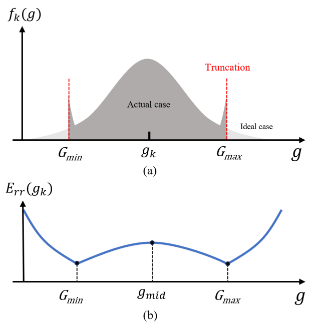


Figure 4: (a) Error probability distribution under truncation. (b) Error expectation across conductance range.

3.5 Joint Dynamic Writing Scheme

To obtain optimal compensation effectiveness, we introduce a joint dynamic writing scheme, which combines both *PT_Shift* and *PT_Scale* techniques together. The scheme is depicted in Fig. 5, which outlines the overall design flow and includes an illustrative example showing the writing process for a 3×3 RRAM array. In this example, we assume each cell has 2 bits with a weight range of [0, 3], and the total bit width of the weight is 6 bits. Therefore, 3 cells are required to represent each weight.

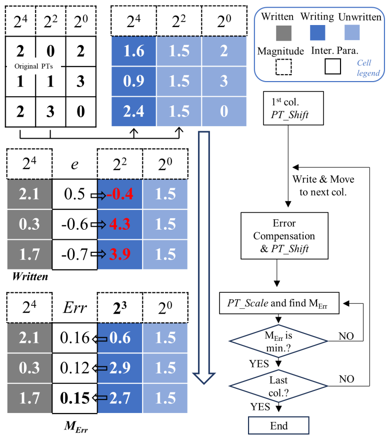


Figure 5: The example and flow diagram of joint dynamic writing scheme.

The writing process begins with the programming of the 1st column. To realize *PT_Shift*, the original PT values of the 1st and 2nd columns need to be shifted, as demonstrated by the changes in PT values in the matrix of the example shown in Fig. 5. As this is the first column and there are no previous column errors to compensate for, the 1st column is programmed according to the shifted PT values. Subsequently, the programmed errors (e) of the 1st column are obtained after the write process and stored in a column of registers as intermediate parameters. The writing process then proceeds to the 2nd column. In this case, the 2nd column needs to compensate for the errors generated from the previous column and accommodate the required *PT_Shift* for the next column at the same time. As a result, the adjusted PT values exceed the weight range representation of the cell, as highlighted in red in the figure.

To address the out-of-range PT issue, the 2nd column employs the *PT_Scale* technique. Scaling the PT values facilitates the keeping of the PT values within the weight range, thus minimizing the chance of truncation. The expected errors (Err) of the cells for a given scaling factor are computed and stored in the error column, replacing the programmed errors from the 1st column. At the same time, the median of the expected errors in the column (M_{Err}) is computed. This computation is repeated for different scaling factors, typically including 1, 2, 4, 8 and 16, which cover the variation effectively. The scaling factor that results in the minimum median expected error is selected for the current column allowing the writing process to proceed. The entire process repeats for the subsequent columns, with each column compensating for the errors from the previous column while accommodating the necessary *PT_Shift* for the next column.

The proposed dynamic writing scheme for compensation forms a one-time sequential writing process, progressing from the most significant column to the least significant column. The calculation of error expectations and finding the median can be carried out on an offline platform, similar to a conventional writing scheme [6, 11, 12].

4 EVALUATION

4.1 Experimental Setup

In this section, we conduct simulations to evaluate five techniques for RRAM writing, including the baseline (BL), SWIPE [10], *PT_Shift* only, *PT_Scale* only and the proposed joint dynamic writing scheme (*RWriC*). The evaluation is based on two metrics: inference accuracy and mean square error (MSE) in intermediate layers. The baseline (BL) represents the conventional writing scheme without any compensation techniques. During the simulations, we introduce device programming variation with a standard deviation (σ) expressed as a percentage of G_{max}, into the RRAM cells. Four benchmarks, including ResNet50 with CIFAR-10, ResNet50 with ImageNet, Yolov8x with detection, and segmentation on COCO, are used in the simulations. The simulations are implemented using PyTorch, allowing us to configure various parameters like quantization bit, DAC/ADC resolution, and the number of bits allocated per RRAM cell. In all cases, the input activations and weights of the benchmark networks are all quantized using 8 bits. Each weight is sliced and mapped onto multiple MLC RRAM cells.

4.2 Experimental Results

We first conducted experiments on Resnet50 with the CIFAR-10 dataset, and the corresponding results for different B_c values are shown in Fig. 6(a)-(c). When compared to BL and SWIPE, our proposed methods demonstrate significant improvements at different B_c values. When $B_c = 4$, the accuracy drop of RWriC is less than 5% in the presence of 12% variation. This variation tolerance increases to 18% when $B_c = 3$. For $B_c = 2$, when the variation is 18%, RWriC only incurs a 0.9% decrease in accuracy. For comparison, before the accuracy drops below 90%, BL and SWIPE can only tolerate variation up to 4% and 10%, respectively. In contrast, RWriC can tolerate a 24% variation in this scenario. This indicates that RWriC exhibits 5 times and 1.4 times improvement in robustness compared to BL and SWIPE, respectively. As the accuracy drops below 80%, continuous increases in variation will have a more pronounced impact on accuracy. SWIPE can maintain accuracy above 80% when the variation exceeds 12%, while *PT_Shift* can handle variation up to 16%. Both techniques exhibit limited tolerance to increasing variation due to their limited compensation ranges. As the variation increases, more cells are truncated, leading to a notable decrease in accuracy. On the other hand, the *PT_Scale* scheme demonstrates the ability to maintain above 80% accuracy even under 26% variation. By combining the *PT_Shift* and *PT_Scale* techniques, RWriC further achieves 28% in the same scenario, surpassing the performance of using *PT_Scale* or *PT_Shift* alone.

We can observe similar trends and conclusions across different models and datasets, as shown in Fig. 6(d)-(f). When dealing with large datasets such as ImageNet and COCO, the robustness of models like ResNet50 and Yolov8x against variations diminishes. Even slight variations can have a significant impact on accuracy. As a result, the effect of applying SWIPE on accuracy is not significant compared to BL. On the other hand, RWriC consistently demonstrates minimal accuracy degradation in the presence of variation and achieves a robustness improvement of 9-11 times compared to the baseline, as long as the accuracy is maintained above 70%.

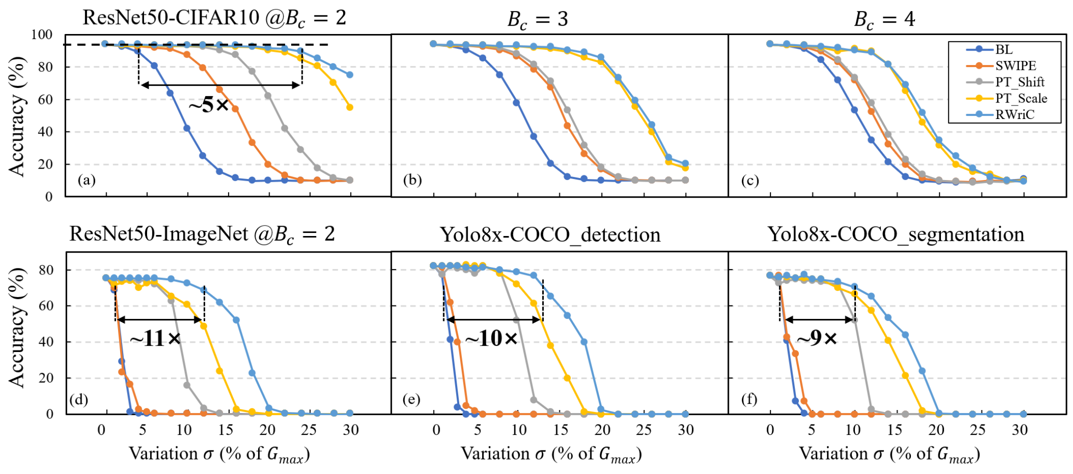


Figure 6: Inference accuracy comparison by deploying different techniques on different NNs: ResNet50 on CIFAR-10 with (a) $B_c = 2$, (b) $B_c = 3$, (c) $B_c = 4$. (d) ResNet50 on ImageNet, (e) Yolov8x on COCO-detection, and (f) segmentation when $B_c = 2$.

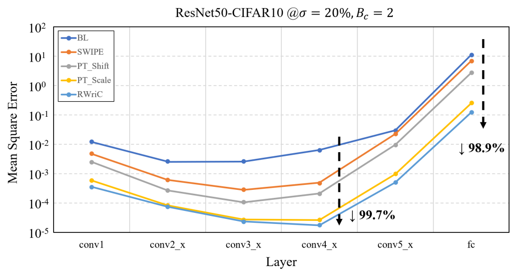


Figure 7: The mean square error across layers of ResNet-50 on CIFAR-10 when deploying different techniques.

The mean square errors (MSE) of the outputs of the intermediate layers, after variations are introduced, are good indicators of the impact on accuracy. We conducted simulations to collect the MSE data at these intermediate layers during the inference of the benchmark models. Fig. 7 presents the data for ResNet50 running on CIFAR-10 with a 20% variation in the weight conductance. It can be seen that in the *conv4_x* layer, SWIPE achieves a significant reduction in MSE, amounting to only 7.6% of the MSE obtained with the baseline. At the same time, RWriC further reduces the MSE to a mere 0.3%. In the *fc* layer, SWIPE only achieves a 37% reduction in MSE compared to BL, indicating a substantial accuracy drop. In contrast, RWriC reduces the MSE to 1.1%, resulting in only a 2.2% drop in the overall inference accuracy.

5 CONCLUSION

This paper introduces RWriC, a dynamic writing scheme specifically designed for fine-grained compensation, aimed at minimizing accuracy loss due to device variation in RRAM-based in-memory computing. Experimental results show that RWriC achieves remarkable performance, with a mere 0.9% inference accuracy loss for ResNet50 on the CIFAR-10, even under 18% device variation. RWriC significantly improves variation robustness and outperforms conventional writing schemes by 5-11x. Additionally, RWriC incurs minimal overhead on RRAM macro, and no offline retraining is needed.

6 ACKNOWLEDGEMENT

This research was supported by ACCESS – AI Chip Center for Emerging Smart Systems, sponsored by InnoHK funding, Hong Kong SAR, and INA – Institute of Nanoscience and Applications, SUSTech, Shenzhen, China.

REFERENCES

[1] H-S Philip Wong, Heng-Yuan Lee, Shimeng Yu, Yu-Sheng Chen, Yi Wu, Pang-Shiu Chen, Byoungil Lee, Frederick T Chen, and Ming-Jinn Tsai. 2012. Metal–oxide rram. *Proceedings of the IEEE*, 100, 6, 1951–1970.
[2] Yangyin Chen. 2020. Reram: history, status, and future. *IEEE Transactions on Electron Devices*, 67, 4, 1420–1433.
[3] Yun Long, Xueyuan She, and Saibal Mukhopadhyay. 2019. Design of reliable dnn accelerator with un-reliable reram. In *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1769–1774.
[4] Gouranga Charan, Jubin Hazra, Karsten Beckmann, Xiaocong Du, Gokul Krishnan, Rajiv V Joshi, Nathaniel C Cady, and Yu Cao. 2020. Accurate inference with inaccurate rram devices: statistical data, model transfer, and on-line adaptation. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.
[5] Beiye Liu, Hai Li, Yiran Chen, Xin Li, Qing Wu, and Tingwen Huang. 2015. Vortex: variation-aware training for memristor x-bar. In *Proceedings of the 52nd Annual Design Automation Conference*, 1–6.
[6] Peng Yao, Huaqiang Wu, Bin Gao, Jianshi Tang, Qingtian Zhang, Wenqiang Zhang, J Joshua Yang, and He Qian. 2020. Fully hardware-implemented memristor convolutional neural network. *Nature*, 577, 7792, 641–646.
[7] Jilan Lin, Cheng-Da Wen, Xing Hu, Tianqi Tang, Chao Lin, Yu Wang, and Yuan Xie. 2020. Rescuing rram-based computing from static and dynamic faults. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 40, 10, 2049–2062.
[8] Jingyu He, Yucong Huang, Miguel Lastras, Terry Tao Ye, Chi-Ying Tsui, and Kwang-Ting Cheng. 2023. Rvcomp: analog variation compensation for rram-based in-memory computing. In *Proceedings of the 28th Asia and South Pacific Design Automation Conference*, 246–251.
[9] Ziqi Meng, Weikanu Oian, Yilonz Zhao, Yanan Sun, Rui Yang, and Li Jiang. 2021. Digital offset for rram-based neuromorphic computing: a novel solution to conquer cycle-to-cycle variation. In *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1078–1083.
[10] Sujan K Gonugondla, Ameya D Patil, and Naresh R Shanbhag. 2020. Swipe: enhancing robustness of reram crossbars for in-memory computing. In *Proceedings of the 39th International Conference on Computer-Aided Design*, 1–9.
[11] Miao Hu et al. 2018. Memristor-based analog computation and neural network classification with a dot product engine. *Advanced Materials*, 30, 9, 1705914.
[12] Can Li et al. 2018. Analogue signal and image processing with large memristor crossbars. *Nature electronics*, 1, 1, 52–59.
[13] Yan Liao, Bin Gao, Feng Xu, Peng Yao, Junren Chen, Wenqiang Zhang, Jianshi Tang, Huaqiang Wu, and He Qian. 2020. A compact model of analog rram with device and array nonideal effects for neuromorphic systems. *IEEE Transactions on Electron Devices*, 67, 4, 1593–1599.