Rescuing RRAM-based Computing from Static and Dynamic Faults

Jilan Lin, Cheng-Da Wen, Xing Hu, Tianqi Tang, Ing-Chao Lin, *Senior Member, IEEE*, Yu Wang, *Senior Member, IEEE*, and Yuan Xie, *Fellow, IEEE*

***Abstract*—Emerging Resistive Random Access Memory (RRAM) has shown the great potential of in-memory processing capability, and thus attracts considerable research interests in accelerating memory-intensive applications, such as neural networks. However, the accuracy of RRAM-based NN computing can degrade significantly, due to the intrinsic statistical variations of the resistance of RRAM cells. In this paper, we propose SIGHT, a SynergIstic alGorithm-arcHitecture fault-Tolerant framework, to holistically address this issue. Specifically, we consider three major types of faults for RRAM computing: non-linear resistance distribution, static variation, and dynamic variation. From the algorithm level, we propose a resistance-aware quantization to compel the neural network parameters to follow the exact non-linear resistance distribution as RRAM, and introduce an input regulation technique to compensate for RRAM variations. We also propose a selective weight refreshing scheme to address the dynamic variation issue that occurs at run-time. From the architecture level, we propose a *general* and *low-cost* architecture accordingly for supporting our fault-tolerant scheme. Our evaluation demonstrates almost no accuracy loss for our three fault-tolerant algorithms, and the proposed SIGHT architecture incurs performance overhead as little as 7.14%.**

***Index Terms*—RRAM, Reliability, Neural Network**

I. Introduction

NEURAL network (NN) is now at the core of the artificial intelligence community, given its excellent performance on various machine learning topics, including image recognition [1], object detection [2], natural language processing [3] and so on. However, the inference of neural networks is typically considered memory-intensive, as a high volume of memory bandwidth is usually required. This large amount of data movement would lead to numerous energy consumption, which appears to be unacceptable in edge devices. As such "memory wall" becomes the chock-point and degrades the performance in traditional von-Neumann architectures, RRAM-based accelerators attract considerable research interests in accelerating NN workloads [4], [5]. With the unique resistance-switching character and crossbar structure, RRAM is able to perform matrix multiplications, which is the key operation in NN models, inside the memory array and reduces half of the data fetching [6], [7]. Beyond the in-memory processing capability, RRAM also provides O(1) computation complexity and ultra-low power consumption [8], [9], making itself a competitive candidate for the next-generation computing platform.

Manuscript received XX XX, XXXX; revised XX XX, XXXX; accepted XX, XX, XXXX. Date of publication XX, XX, XXXX. This work is supported in part by the NSF 1816833, 1725447, and 1719160, and in part by the Ministry of Science and Technology under grant MOST 106-222-E-006-027-MY3 and 109-2628-E-006-012-MY3 and in part supported by the Intelligent Manufacturing Research Center (iMRC) from the Featured Areas Research Center Program by the Ministry of Education, Taiwan (ROC).

J. Lin, X. Hu, T. Tang, and Y. Xie are with the Department of Electrical and Computer Engineering, the University of California at Santa Barbara, Santa Barbara (e-mail: jilan@ucsb.edu; huxing@ece.ucsb.edu; tianqi_tang@ucsb.edu; yuanxie@ucsb.edu).

C. Wen and I. Lin are with the Department of Computer Science and Information Engineering, National Cheng Kung University, Tainan (e-mail: p76071145@gs.ncku.edu.tw, iclin@mail.ncku.edu.tw).

Y. Wang is with the Department of Electronic Engineering, Tsinghua University, Beijing (e-mail: yu-wang@tsinghua.edu.cn).

Jilan Lin and Cheng-Da Wen contributed equally to this work.

Unfortunately, previous studies on RRAM characterizations have revealed that current RRAM devices exhibit several reliability issues and non-ideal faults, and we hardly have the RRAM resistance being the exact value expected [10]–[13]. Different from using RRAM as a memory device, such faults can lead to severe accuracy loss when using RRAM for computation [14], [15]. In this paper, we pay special attention to three types of faults that are commonly seen in RRAM devices. ***(1) Non-linear Resistance Distribution***: Multi-level cell (MLC) has been broadly leveraged in RRAM-based accelerators as it significantly increases the data density and saves the design budget. It has been indicated that the resistance in MLC is not continuously tunable and there may be resistance gaps between different resistance levels [16], [17]. However, prior work simply applied linear quantization to the NN models assuming that an n-bit fix-pointing value can be mapped to an n-bit cell, which actually does not seem to hold for all the RRAM cells. For example, we found that different resistance states in some RRAM models are exponentially increased [13]. The non-linear resistance distribution exposes a challenge that mapping traditional NN to such RRAM may not work and novel quantization algorithms specialized for RRAM-based computing are demanded. ***(2) Static Variation***: It has been widely known that RRAM exhibits serious variations, meaning that the actual value we write into one cell can deviate a lot from the expected one [18]–[20]. There are mainly two types of static variations in the current RRAM technology: device-to-device variation and write-to-write variation. The device-to-device variation makes it difficult to generate a unified solution for a single RRAM crossbar array since different cells deviate differently from each other. The write-to-write variation results in the huge overhead in the conventional re-write scheme to address the variation issue, as we may need to re-write multiple times until getting the correct value. ***(3) Dynamic Variation***: As the static variation refers to the variation caused by programming the RRAM cell, the dynamic variation means that the cell resistance keeps changing over time [21]–[23]. This issue is particularly concerning because it could get worse in the NN acceleration scenario where those read operations

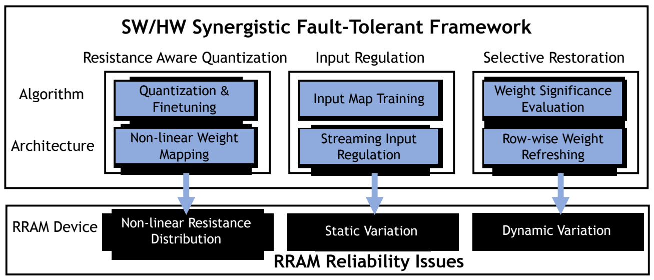

SW/HW Synergistic Fault-Tolerant Framework
Resistance Aware Quantization
Input Regulation
Selective Restoration
Algorithm
Architecture
Quantization & Finetuning
Non-linear Weight Mapping
Input Map Training
Streaming Input Regulation
Weight Significance Evaluation
Row-wise Weight Refreshing
RRAM Device
Non-linear Resistance Distribution
Static Variation
Dynamic Variation
RRAM Reliability Issues

Fig. 1. An overview of SIGHT, a SW/HW synergistic fault-tolerant framework. We leverage algorithm-architecture co-design to address three reliability issues in RRAM-based computing.

need to charge the RRAM crossbar much more frequently and thus degrade the performance in NN workloads.

Prior work has made efforts on addressing such reliability issues on both hardware and software sides. From the hardware side, Li proposed a verify-after-write approach to tune the RRAM cell more accurately against the variation [24] while Cheng further improved the energy efficiency of this solution with a RESET-free approach [25]. From the software side, Chen proposed a mapping scheme for NN parameters to avoid important weights being in the variation-affected RRAM cells [14]. NN training techniques have also been explored to enhance the model robustness against variations and non-linear resistance distribution [15], [26]. Lammie1 proposed a variation-aware CNN architecture that is specific for RRAM to solve the variation problem [27]. However, as these studies only touched only one or two reliability issues at a time, there lacks a systematic and unified solution to tackle all three faults. Therefore, we present SIGHT, a SynergIstic alGorithm-arcHitecture fault-Tolerant framework, to holistically address those problems. As shown in Fig. 1, SIGHT leverages algorithm-architecture co-optimizing, which trains and maps NN models with the awareness of RRAM faults and facilitate the processing with dedicated architectural support. We summarize our key contributions as follows:

- We propose a resistance-aware NN quantization algorithm, which forces the weights in the NN model to follow the resistance distribution as RRAM and tackle the non-linearity issue. Compared with prior work [26], we extend and evaluate our technique to various resistance distributions and demonstrate no accuracy loss.
- We introduce an input regulation method to avoid the accuracy degradation incurred by the static variation. We compensate for the error caused by variation through an integrated input map to regulate input vectors. Compared with prior work [15], the input regulation could resume the accuracy under much larger static variation.
- We propose an RRAM refreshing scheme to resolve the dynamic variation issue. We define the significance of each cell by its weight, and selectively refresh the important RRAM cells at run-time.
- We architect *general* and *low-cost* hardware based on existing RRAM-based accelerator [5] for supporting our fault-tolerant techniques above. The hardware simulation reveals that SIGHT introduces 7.14% performance overhead on average for various NN workloads.

Our paper is then organized as follows: We introduce preliminaries of NN and RRAM in Section II and present the detailed models of the three RRAM fault types in Section III. We introduce our fault-tolerant scheme in Section IV and the architectural support in Section V. The evaluation is in Section VI and we summarize the related work in Section VIII.

II. Preliminaries

In this section, we introduce the background on neural network, RRAM device and RRAM-based computing for further discussion.

A. Neural Network Basis

Neural networks (NNs) origins from mimicking the neuron system in humans [28]. The architecture of an NN often consists of artificial neurons and synapses between them. The neurons are organized layer by layer. The neurons in one layer receive the outputs from the previous layer and then propagate their outputs to the next layer.

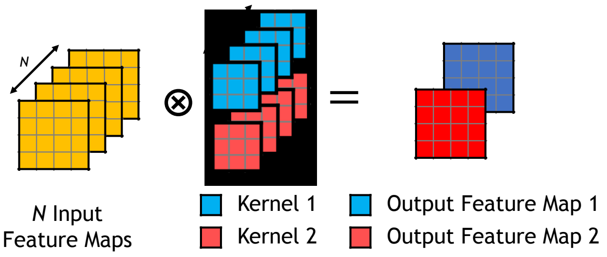


Fig. 2. An illustration of a CNN layer, which is composed of N (N = 4 in the figure) input feature maps convoluted by 2 kernels.

Convolutional Neural Network (CNN) is an important branch of the NN family that mainly targets computer vision tasks. As shown in Fig. 2, the inputs for a convolutional layer are a bunch of 2-D images, which are called feature maps. A group of 3-D convolutional kernels (shown as blue and red) then filter the feature maps with a fixed-size sliding window. Since each kernel generates one output feature map, we finally get 3-D output for the next layer. The computation in the convolutional layer can be expressed in Eq. 1:

$$\boldsymbol{d}^{l+1}_{x,y,n^{l+1}} = f(\sum_{n^l=0}^{N^l-1}\sum_{r=0}^{R-1}\sum_{s=0}^{S-1} \boldsymbol{d}^l_{x+r,y+s,n^l} \times \boldsymbol{w}^l_{r,s,n^l,n^{l+1}}) \quad (1)$$

where $\boldsymbol{d}$ is a 3-D tensor representing the feature map and $\boldsymbol{w}$ is a 4-D tensor representing the convolutional kernels. The superscript l denotes the l-th layer. Therefore, $\boldsymbol{d}^l$ has the shape of $X \times Y \times N^l$ ($4 \times 4 \times 4$ as shown in Fig. 2) and represents the input feature map, while $\boldsymbol{d}^{l+1}$ represents the out feature map. $\boldsymbol{w}^l$ has the shape of $R \times S \times N^l \times N^{l+1}$ ($3 \times 3 \times 4 \times 2$ as shown in Fig. 2) and represents the convolutional kernels. f is an activation function that aims to add non-linearity into the neural network.

B. RRAM Basis

Resistive Random Access Memory (RRAM) is one of the emerging non-volatile memories, which is also known as the memristor [20], [29]. As shown in Fig. 3(a), the RRAM cell is a passive bipolar device and usually applies the metal-insulator-metal structure. The middle insulator showing resistive switching characteristics can be made of various materials such as HfO_2 [17], TiO_x [30], NiO [31] and so on.

The attractive resistive switching property usually comes from the conductive filament between two electrodes [31], [32]. When applying a certain programming voltage to the cell, the filament grows up and connect two electrodes together, which then reduces the cell resistance and make it conductive. This process is called a SET operation where it tunes the RRAM cell from the high resistance state (HRS, representing ”0”) to the low resistance state (LRS, representing ”1”). The reversed operation which destroys the filament between two electrodes is thus called a RESET operation. Both SET and RESET are considered as write operations to the RRAM.

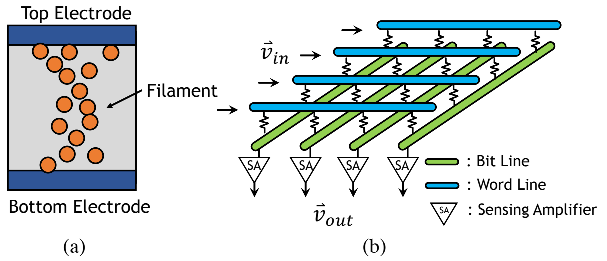


Fig. 3. (a) An RRAM cell with the metal-insulator-metal structure. The top and bottom electrodes are made of metal and the middle insulator shows resistance switching character. (b) The structure of an RRAM crossbar. The input voltage opens one word line and data are read from bit lines.

RRAM cells can be organized in a crossbar structure for higher area-efficiency, as shown in Fig. 3(b). When using as a memory device, the row decoder opens one word line according to the address and sends a read voltage, and data are read from selected bit lines through sensing amplifiers. To further improve the density and save the hardware cost, multi-level cell (MLC) is broadly considered, where one RRAM cell can store several bits of information by tuning it to multiple resistance states [16], [17]. In that case, higher resolution for analog-to-digital interfaces is then required to decode the data.

C. RRAM-based Computing System

With the crossbar structure and resistive switching character, researchers have explored RRAM's potential of in-memory computing [4], [5]. First, we can store an $n \times n$ matrix in an $n \times n$ crossbar. Then, all the word lines are opened simultaneously and we set input voltages with respect to a vector. Therefore, we have the output currents from bit lines being the result of matrix-vector multiplication. The scalar multiplication is done by voltage-conductance multiplication, and the result is accumulated by all the currents within a bit line. Assume that the input voltage vector is (V^i) and the output is (V^o), the computation can then be expressed as in Eq. 2:

$$\begin{bmatrix} V_1^o \\ \vdots \\ V_M^o \end{bmatrix} = \begin{bmatrix} c_{1,1} & \cdots & c_{1,N} \\ \vdots & \ddots & \vdots \\ c_{M,1} & \cdots & c_{M,N} \end{bmatrix} \begin{bmatrix} V_1^i \\ \vdots \\ V_N^i \end{bmatrix} \tag{2}$$

$$c_{i,j} = -g_{i,j}/g_s \tag{3}$$

where $c_{i,j}$ is the matrix parameter, which depends on the RRAM conductance in the corresponding position (i, j) and the reference conductance g_s in sensing amplifiers as shown in Eq. 3. Since RRAM conductance can only be positive, two crossbars are needed to represent an application matrix with both positive and negative parameters [33].

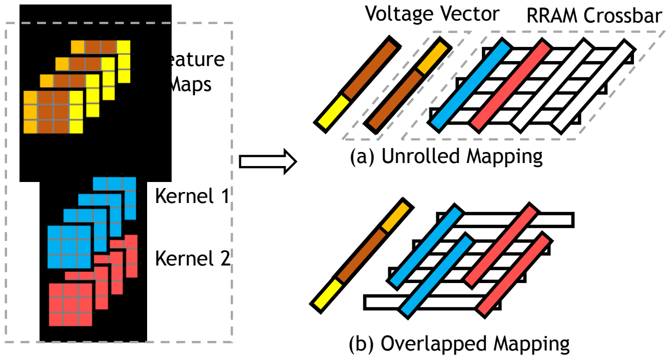


Fig. 4. An illustration of mapping a CNN layer to RRAM crossbars. (a) A CNN layer with two convolutional kernels. Brown pixels are reused by two adjacent sliding windows. (b) Direct mapping: Unfold inputs and kernels into vectors and conduct vector multiplication, where kernel vectors are mapped to RRAM directly. (c) Overlapped mapping: Duplicate kernel vectors in RRAM crossbars and concatenate feature map vectors together to reduce the latency.

As this computing mechanism significantly reduces the complexity of matrix-vector multiplication from $O(n^2)$ to $O(1)$, previous work has intensively studied how to leverage it for NN acceleration [4], [34].

One challenge here is to map the convolution onto RRAM crossbar. Fig. 4 presents two methods to map a convolutional layer [35]. The first way is to unroll both weight kernels and feature maps into matrices and complete the convolution in a General Matrix Multiplication (GEMM) manner. As shown in Fig. 4(a), we use an RRAM column to store an unrolled convolution kernel, and the feature maps are tiled into vectors and sent to the crossbar as input voltages. Then, we receive each output pixel cycle by cycle. This method is quite straightforward and leveraged by many accelerator designs [5], [27]. Fig. 4(b) demonstrates another way to improve the computation throughput by reusing input data in feature maps [36]. As shown, we duplicate weight kernels twice and put them into two RRAM columns in an overlapped manner. Then, we concatenate the two input vectors in Fig. 4(a) into one vector, and the overlapped RRAM rows can share and reuse the same input voltages. Thus, we could get 2 output pixels in a single cycle. This reduces the inference latency and increases the computation throughput by trading-off RRAM resources.

III. RRAM FAULTS MODELING

In this section, we introduce the fault models we wish to address in RRAM-based computing. We analyze the faults under three categories: non-linear resistance distribution, static variation, and dynamic variation.

A. Non-linear Resistance Distribution

As mentioned, MLC helps us gain more data density and provides considerable savings under the limited hardware budget. Since the parameter $c_{i,j}$ in a matrix should be linearly mapped into the corresponding RRAM's conductance $g_{i,j}$ according to Eq. (3), the conductance/resistance distribution must be exactly the same as the distribution of matrix parameters. However, we find that as the RRAM processing technology has not converged and various types of RRAM based on different materials exist, the distributions of multiple resistance states in MLC are quite diverse.

Here we show in Fig. 5 two cases of resistance distribution. Both of them are 2-bit cells from existing work, with WO_x [12] and HfO_2 [16] based RRAM respectively. To distinguish the non-linear distribution (among all resistance states) and the static variation (for one particular resistance state), we here only consider the mean value for each state. As seen, we hardly expect the resistance distribution to be perfectly linear. For the distribution illustrated in Fig. 5(a) (where it shows the number of cells measured in different resistances), four different resistance levels appear to be linear. However, some deviations are making the resistance gaps between them not strictly the same. For the distribution illustrated in Fig. 5 (b), we find four resistance levels exponentially increased under the logarithmic axis, where the highest resistance state is about $1000\times$ larger than the lowest one.

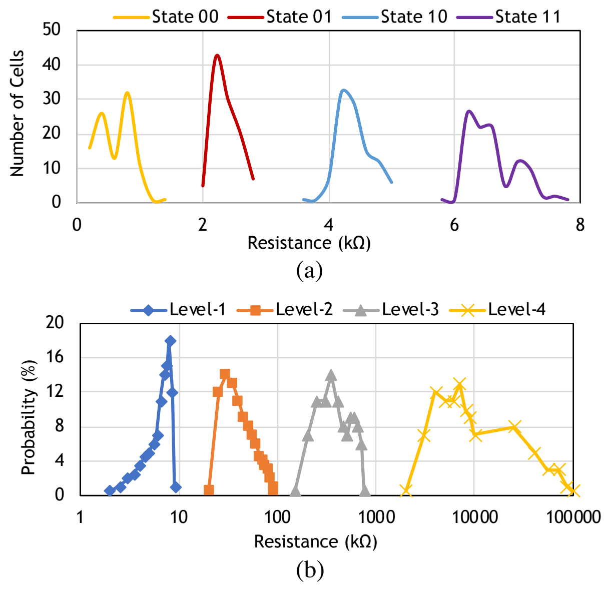


Fig. 5. The multi-state resistance distribution in MLC for (a) WO_x-based RRAM [12] and (b) HfO_2-based RRAM [16].

To mathematically analyze the non-linear resistance distribution for further discussion, we propose three fitting functions to model the various resistance distribution: ***(a) Deviated Linear Model:*** For the case shown in Fig. 5(a), we apply a deviated linear model where the conductance states are approximately linear and we add random noise δ_k to them, as expressed in Eq. 4(a) where k means the k-th conductance state. ***(b) Exponential Model:*** For the case shown in Fig. 5(b), we fit the conductance states with a exponentially increased function. As expressed in Eq. 4(b), the base β in the exponential function will reflect how the conductance grows. ***(c) Power Model:*** For other cases that can be seen in other work [16], [17], [37] where either the linear or exponential function may not be proper to represent them, we propose a power model that has a moderate growth speed between the linear and exponential model, as shown in Eq. 4(c) where the α is the exponent/index.

$$\begin{cases} g_k = Ck + \delta_k & (a) \\ g_k = C\beta^k & (b) \\ g_k = Ck^\alpha & (c) \end{cases} \tag{4}$$

To decide the parameters δ, β and α, one can simply apply a minimal square root error (MSE) based fitting algorithm.

B. Static Variation

Unlike the binary cell that only stores 0 or 1, RRAM as an analog device exhibits serious uncertainty regarding the resistance value, as we observed in Fig. 5. Here we call such uncertainty static variation since the variation is fixed after setting or unsetting it. This variation is usually caused by the non-uniformity when forming the filament between two electrodes as it is very difficult to control two filaments to be exactly the same, either for two RRAM cells or for two SET operations. Therefore, there are conventionally two types of static variations: device-to-device variation and write-to-write variation, referring to the resistance difference between two RRAM cells and two SET operations respectively.

When used as a memory device, the static variation seems not troubling because we do not necessarily require the RRAM cell to be precise, as long as two different resistance states are distinguishable. However, it surely becomes a huge threat to the performance when using RRAM for computing, since any change to the operators may lead to incorrect results. Therefore, this RRAM characteristic must be taken into consideration. Here we assume that the higher resistance level suffers from more serious variation, which is indicated in Fig. 5 and other previous work [16], [20], [38]. As expressed in Eq. 5, we add a stochastic noise to the ideal resistance to model the static variation, which obeys the standard normal distribution. The variance of the distribution is linearly related to the resistance itself with a coefficient λ.

$$r_{actual} = r_{ideal} + \Delta, \Delta \sim \mathcal{N}(0, \lambda r_{ideal}) \tag{5}$$

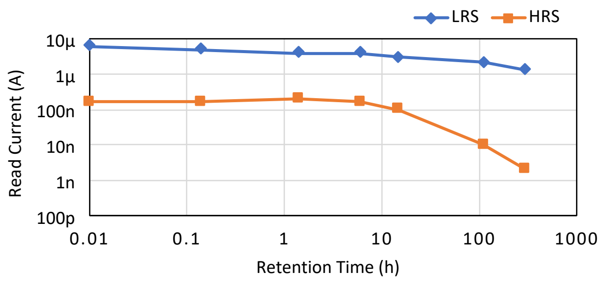


Fig. 6. The resistance drifting for both HRS and LRS in an HfO_2-based RRAM [23] baking at 200°C. The y-axis represents the read current, and the median value is shown. The x-axis represents the time in logarithmic unit.

C. Dynamic Variation

Both the non-linear resistance distribution and static variation are statically fixed after mapping a NN model. On the other hand, researchers also observed the RRAM resistance may drift over time, as the formation of filament is not stable and can be easily affected by the temperature, read current or other environmental factors [39]–[42]. This issue is also known as the retention problem describing how long an RRAM device would keep its data. When using RRAM for accelerators, the resistance may drift even worse as the number of reads increases and multiple rows are opened simultaneously.

Fig. 6 presents how two resistance states drift over time for an HfO_2-based RRAM [23]. We find that different from the static variation, the dynamic variation tends to increase the cell resistance as the formed filament tends to be narrowed instead of keeping growing. Therefore, here we use a simplified model and make the assumption that the dynamic variation is single-directional, meaning the resistance keeps increasing over time. Since the dynamic variation also shows stochasticity and Fig. 6 presents only median values, we still use noises with standard normal distribution to model the dynamic variation. As expressed in Eq. 6, we apply a similar model as static variations but take the absolute value of the noise instead.

$$r_{drifted} = r_{original} + |\Delta|, \Delta \sim \mathcal{N}(0, \lambda r_{ideal}) \quad (6)$$

IV. Fault-Tolerant Scheme

In this section, we discuss how to address the three types of RRAM faults mentioned above in the algorithm level.

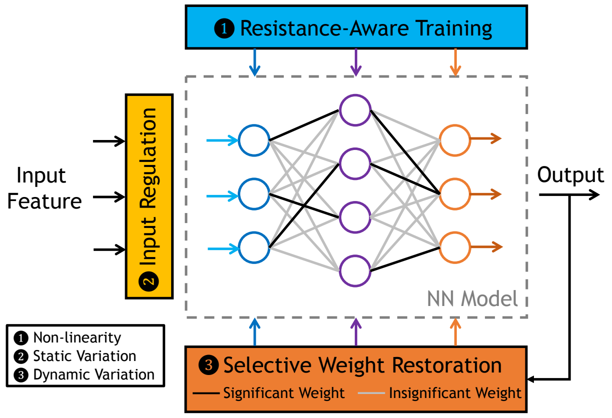


Fig. 7. The workflow for the proposed fault-tolerant scheme, where three techniques are proposed to address three types of faults respectively.

A. Overview

Fig. 7 shows an overview of our scheme workflow, where we propose three dedicated techniques to tackle the three types of RRAM faults respectively. First, given an RRAM-based NN accelerator, we derive the resistance/conductance distribution according to the specific RRAM technology, which indicates the values we can map to the RRAM crossbar. With this distribution information, we quantize the NN models with our proposed ❶ resistance-aware quantization algorithm. The technique is performed offline and forces the NN parameters to follow the exact distribution as RRAM. Second, after mapping the NN model to RRAM successfully, we read the static variation of each cell from RRAM crossbars and train an input map offline with our ❷ input regulation technique. The input map regulates the input during inference to compensate for the known variations in RRAM crossbars. Finally, before online execution, we extract significant weights in the NN model under an RRAM-friendly pattern. We determine the significance of weight with respect to a whole RRAM row. After the accelerator starts to execute the inference, we ❸ periodically refresh the resistance of such significant cells in selected rows to resist against dynamic variations.

B. Resistance-Aware Quantization

The non-linear distribution in RRAM devices makes it difficult to map the NN models, because existing quantization algorithms are mostly linear quantization [43]–[45]. Some research proposed non-linear quantizing the NN model using $\{1, 2, 4, 8, ...\}$, facilitating the NN inference in CMOS-based platforms by switching multiplication to bit operation, since multiplying the input by 4 means shifting left by 2 bits [46], [47]. However, there is very little work offering an RRAM-aware quantization and tackling the non-linear problem in RRAM-based NN accelerators [26].

Algorithm 1 Resistance-Aware Quantization

```
Require: Positive weight tensor $\mathcal{W}$; Quantization width $n$; Conductance list $G = [g_1, g_2, ..., g_{2^n}]$; Base $\beta$ (for exponential) or Index $\alpha$ (for power)
1:  Initialize:
2:      Quantization level $L = 2^n$;
3:      Decision boundaries $B = [b_0, b_1, b_2, ..., b_L]$, $b_0 = 0$, $b_L = \infty$;
4:      Quantized weight $\mathcal{W}_{quan}$ = zero_tensor ($\mathcal{W}$.shape);
5:      Weight scale $\gamma = \mathcal{W}$.max / $g_L$
6:
7:  if Deviated Linear then $B$ = boundary_decision_linear($G$)
8:  else if Exponential then $B$ = boundary_decision_exp($\beta$)
9:  else if Power then $B$ = boundary_decision_power($\alpha$)
10: end if
11:
12: for $k = 1; k \leq L;$ ++$k$ do
13:     $\mathcal{W}_k = ((\mathcal{W}_k > b_{k-1})$ & $(\mathcal{W}_k \leq b_k)) \times g_k \times \gamma$
14:     $\mathcal{W}_{quan}$ += $\mathcal{W}_k$
15: end for
16: Return $\mathcal{W}_{quan}$
17:
18: *****************Decision Functions *****************
19: function BOUNDARY_DECISION_LINEAR($G$)
20:     for $k = 1; k < L;$++$k$ do
21:         $b_k = \gamma \times (g_k + g_{k+1})/2$
22:     end for
23: end function
24:
25: function BOUNDARY_DECISION_EXP($\beta$)
26:     for $k = 1; k < L;$ ++$k$ do
27:         $b_k = \gamma \times \beta^{k+0.5} \times (g_k/\beta^k)$
28:     end for
29: end function
30:
31: function BOUNDARY_DECISION_POWER($\alpha$)
32:     for $k = 1; k < L;$++$k$ do
33:         $b_k = \gamma \times (k + 0.5)^\alpha \times (g_k/k^\alpha)$
34:     end for
35: end function
```

Therefore, we propose a resistance-aware quantization algorithm to make NN parameters aligned with the RRAM resistance/conductance distribution, which is presented in Algorithm 1. First, the algorithm receives a positive weight tensor as input since an RRAM crossbar can only represent positive values. So, we need to extract the positive and negative parts of a weight tensor and quantize them separately. Other inputs include the quantization width, and conductance distribution information as discussed in Section III. Second, we initialize a quantized weight tensor and derive a scale factor γ between the max value of $\mathcal{W}$ and the highest conductance state. Then, we calculate the decision boundaries that decide what a particular weight value should be quantized to. For the deviated linear model, we follow the traditional quantization approach and take the midpoint (mean value) of two quantization intervals as the decision boundary, as shown in Lines 19-23. But for the exponential model and power model, such a choice does not make too much sense because the distribution of quantization intervals is not uniform. Therefore, we here apply the midpoint of exponents (for the exponential model) and bases (for the power model) as the decision boundary. That is, $\beta^{k+0.5}$ and $(k+0.5)^{\alpha}$ respectively. Finally, with these decision boundaries, we can decide which quantization interval a weight locates at and combine them together to form the quantized weight tensor according to Lines 12-15, which can be mapped to the RRAM crossbar array favorably.

Note that the quantization algorithm does not work alone, as the model accuracy may degrade after pure quantization. Therefore, we take a finetune process for the quantized NN model as described in [44]. Therefore, we keep quantizing and re-training the model iteratively until the accuracy converges. The quantization algorithm is applied before both the inference and the forward stage in the training.

C. Input Regulation

After mapping the quantized model to RRAM crossbar arrays, we may still encounter the static variation in RRAM that severely hurts the performance. Previous work on tackling the RRAM variation mostly relies on RRAM cell re-writing [24], [25] or NN parameter re-mapping [14]. However, these methods may cause large hardware overheads due to the write-to-write variation. On the other hand, some work leverages NN training to make the model more robustness against variation [15], [48] by injecting stochastic variation into the training process. We will show in the experiment regarding comparison to such technique.

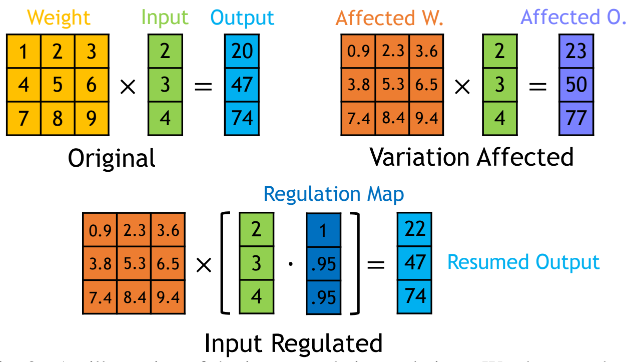


Fig. 8. An illustration of the input regulation technique. We show on the top that variation-affected weights lead to incorrect output, where all the outputs are rounded to integers. The bottom figure shows how the input regulation map works to resume the output.

To tackle the static variation, we leverage the opportunity that such variations are fixed and can be known immediately after mapping. Therefore, we propose to regulate the input and compensate the known variations instead of pursuing a perfectly correct RRAM cell. As the output relies on both weight and input, we slightly adjust the input voltage to keep the final results correct under the variation-affected weight. As shown in Fig. 8, we take a matrix-vector multiplication as an example. The original computation is $\mathcal{W}x = y$. After we map the $\mathcal{W}$ onto the RRAM crossbar, we get a variation-affected weight matrix $\mathcal{W}_{noised}$, which would lead to an incorrect output shown as purple. Then, we add an input map $\mathcal{M}$ to the input x accordingly, where element-wise multiplication will be performed over $\mathcal{M}$ and x. This process will scale the input a little to compensate for the certain variation pattern. Finally, we resume the output more accurately to the original one.

So, our goal is to search for an optimal input map $\mathcal{M}$ that best resists against the static variation. This problem can be formulated mathematically to an optimization problem, where we wish to minimize the computation error under the matrix-vector multiplication constraint, as expressed in Eq. 7:

$$\begin{aligned} \min_{\mathcal{M}} \quad & Error(y, y_{noised}) = \sqrt{||y - y_{noised}||_2} \\ \text{s.t.} \quad & y = \mathcal{W}x \\ & y_{noised} = \mathcal{W}_{noised} \times (\mathcal{M} \cdot x) \end{aligned} \tag{7}$$

However, the input x in the equation above remains unknown. Our opportunity is that since the neural network itself is trained over a specific dataset, for a particular piece of weight, the input may thus follow some specific patterns. For example, the input in the corner of an image may have more white pixels. Therefore, we propose to obtain the $\mathcal{M}$ through the same training procedure as to train the neural network. First, we initialize the $\mathcal{M}$ to an all-1 tensor, meaning that the x remains unchanged after getting multiplied by 1. Second, we can approximate the optimal $\mathcal{M}$ using common training techniques such as stochastic gradient decent. In this step, we fix the weight matrix and update the input map only. Through going over all training images, we finetune the input map iteratively until convergence.

D. Selective Weight Restoration

The resistance-aware quantization and input regulation help tolerate the static faults which can be detected at the mapping stage. However, the reliability issue also occurs at the inference state when we keep running the RRAM-based accelerator. As we discussed in Section III-C, the resistance of RRAM may drift over time. Previous work mainly focused on the endurance problem [49] to protect RRAM cells from frequent writing, but the retention problem in the NN acceleration scenario has been rarely touched.

To address this problem, we propose a selective weight restoration method which determines the significant weights in an NN model and restores them from dynamic variation. The idea origins from the observation that only a small fraction of weights show the importance to NN models and slight changes to the insignificant weight will not hurt the accuracy due to the intrinsic robustness of NN [44]. Therefore, we can back up these significant weights and restore them when they suffer from the resistance drifting/dynamic variation.

However, the challenge is that the significant weights are in fact randomly distributed among RRAM crossbars, and thus it would take a long time to re-write all these weights. To overcome the writing overhead, we then introduce an RRAM-friendly weight restoration method, which leverages the RRAM crossbar's nature that it is possible to write one row in the RRAM crossbar simultaneously to reduce the latency [49]. So, after we map the NN model to RRAM, we detect the weight significance in a row-wise manner for each crossbar. First, we sum up the weight within a whole row as expressed in Eq. 8, which indicates the significance of this row. Since we already separate the positive and negative weights, we do not need to take the absolute value. Second, we select a number of rows with the largest significance and back up them in the main memory. Every crossbar selects the same number of rows to avoid the restoration imbalance. Finally, we periodically

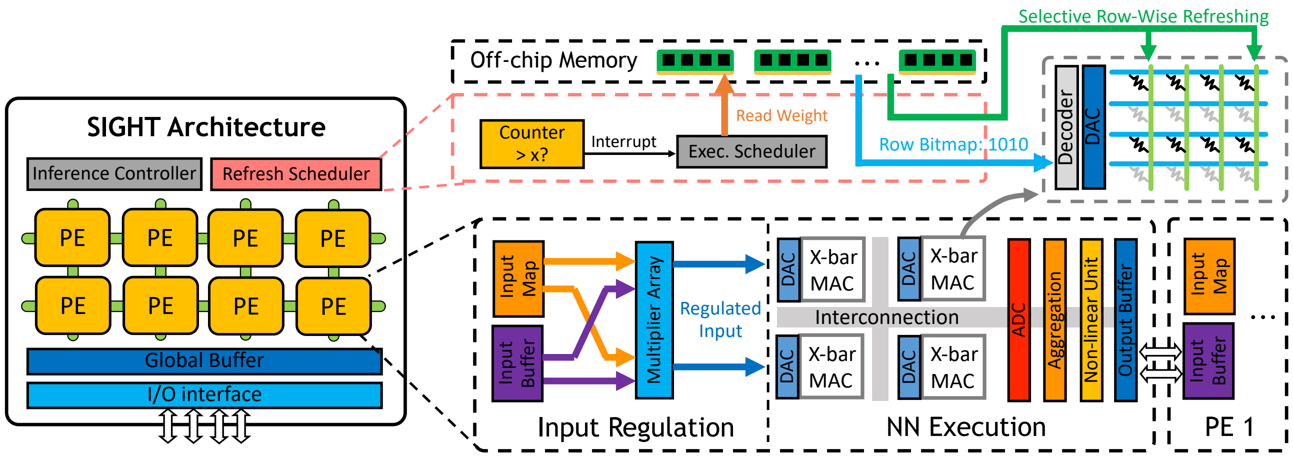


Fig. 9. The SIGHT architecture overview. We mainly design (a) the input regulation unit for each PE to execute the NN inference and (b) the weight refreshing scheduler to restore the significant weights in RRAM crossbars.

refresh these rows in RRAM crossbars to ensure the accuracy performance.

$$\mathcal{S}_j = \sum_{i=0}^{H} w_{i,j} \tag{8}$$

Note that to avoid the contradiction to input regulation that has fixed input maps for the static variation, we need to write the cell precisely during refreshing RRAM crossbars. We consider this overhead very small because we only re-write a small portion in an RRAM crossbar and the refreshing is conducted after a period of time. We will quantitively evaluate the overhead in Section VI.

V. Architecture Design

In this section, we discuss how we architect the RRAM-based accelerator and equip it with the fault-tolerant capability according to the techniques proposed in Section IV.

A. Overview

As RRAM reliability issues broadly exist in various accelerators, our design principle is to make our design (1) *unified*, where we address all the reliability issues in one framework; (2) *general*, meaning it applies to various RRAM-based accelerators; (3) *low-cost*, as negligible performance and hardware overhead would be introduced. For these purposes, we architect add-on hardware in the existing RRAM-based accelerator to support our fault-tolerant scheme.

Fig. 9 gives an overview of the SIGHT, which is based on the ISAAC-like [5] processing flow. The SIGHT consists of two parts, the NN inference accelerator and fault-tolerant units. To process the NN inference, SIGHT distributes the workload to a number of RRAM-based processing elements (PEs). Each PE tile is mainly composed of the input/output buffer, RRAM crossbar arrays for matrix multiplication, registers for result aggregating and non-linear units for activation functions. The PEs are organized in a mesh manner, for better reconfiguring with various NN workloads. Beyond the functional modules for NN executions, we design fault-tolerant units to resist the RRAM faults. First, we add a multiplier array beside the RRAM crossbar, which regulates the input voltage to resist the static variation. Second, we design a weight refreshing scheduler. The scheduler issues interrupt signals periodically or on-demand to NN executions, and then refreshes the significant weights to recover the accuracy loss caused by the dynamic variation.

B. Hardware Design

RRAM-based PE is similar to the tile design in ISAAC. Multiple RRAM crossbar arrays are used to compute the matrix multiplication. The crossbars are interfaced by the digital-to-analog converter (DAC) and analog-to-digital converter (ADC). As the area overhead of ADC is usually considerably larger than DAC [5], we make each crossbar have its own DAC arrays but share the ADC arrays with other crossbars within the PE. We use SRAM to buffer the input and output activations. A PE's input/output buffers are interconnected with adjacent PEs for the convenience of layer propagation. We also use aggregation registers to aggregate the partial sums of (1) split matrix and (2) split precision from different RRAM crossbars. The latter one is for the reconfigurable precision purpose. Since the RRAM, DAC and ADC are fixed-precision and some NN workloads demand higher precision, splitting most significant bits (MSBs) and least significant bits (LSBs) thus becomes a common technique used in RRAM-based computing [4], [5]. Finally, we put a non-linear unit to support activation functions and pooling operations in an NN model.

Input Regulation Unit is to adjust the input voltage according to the fault-tolerant scheme presented in Section IV-C. This unit is composed of a multiplier array, which locates right beside RRAM crossbar arrays. We also put an extra buffer within the PE to store the input map $\mathcal{M}$. Then, the multiplier array receives operators from the input buffer and map buffer and sends the regulated data to DACs. This only incurs very little performance overhead of one multiplier cycle, which we will discuss in detail in Section VI.

Refreshing Scheduler is a separate piece of control logic that aims to refresh the RRAM crossbars affected by the dynamic variation. The scheduler decides to refresh according to an internal counter, which records the time interval from the last refreshing. Whenever the counter exceeds the pre-set threshold, the scheduler issues an interrupt signal and starts to refresh the RRAM crossbars. As all the backup weights are stored offline, the significant weights will be read from off-chip memory and written to crossbars row-wisely. For each crossbar, we use a bitmap to decide which rows are going to refresh and put a dedicated decoder to decode the bitmap (searching for non-zero positions) and process the rows sequentially.

Interconnection & Controller: The PEs are organized as a 2-D mesh, where one PE can receive the input activations from others or route its output to adjacent PEs. The RRAM crossbars within a PE are connected with a shared bus. Once an NN model is mapped, all the data paths for execution are fixed offline. The execution controller then runs a finite state machine and takes charge of all the pipelines according to the control registers, which determines how the results are routed.

C. Execution Flow

Mapping: Similar to the ISAAC and other RRAM-based accelerators, SIGHT puts all NN parameters on-chip, which are distributed over all the RC tiles. This assumption holds because the RRAM itself is a memory device and writing RRAM repeatedly will significantly reduce its endurance [49]. Therefore, we first map the model to the accelerator offline, and one layer could be mapped to one or several PEs. After mapping, we read actual weight values from RRAM crossbars that are affected by static variations. Then, we train the input map $\mathcal{M}$ offline and write the map to each regulation buffer. Finally, we program the control registers in the execution controller with respect to the data flow fixed by the mapping.

Execution: After we start the execution controller, the global buffer loads the image from the off-chip memory and sends it to PEs that process layer 0. The input will first be regulated by multiplying with the input map and then sent to the corresponding RRAM crossbars. The computation results are temporarily stored at the aggregation registers, where the partial sums are accumulated. Finally, after going through the non-linear function unit, the output of this layer is then stored in the output buffer and waiting for routing to the next PE.

Refreshing: The RRAM crossbars refresh themselves with a pre-set time interval. When it is time for refreshing, the refreshing scheduler first sends an interrupt to the execution controller and suspends the inference processing. As backup weights are stored offline, the scheduler then issues memory requests to the I/O interface and reads the row-wise significant weight and the bit map. To reduce the overhead, we wish all crossbars to refresh simultaneously. Therefore, we make the weight reading and crossbar refreshing in pipeline. First, one significant row is read for each crossbar at a time. Second, the crossbar refreshes this row with the address decoded from the bit map. Meanwhile, the scheduler starts to read the next row to hide the I/O latency. Since we restore a certain percentage of rows for every crossbar, no refreshing imbalance would be introduced consequently.

VI. Evaluation

In this section, we provide the experiment results of our proposals and analyze the insight from them.

A. Methodology

We evaluate our proposal from both software and hardware aspects. For the software aspect, we demonstrate the efficacy of the fault-tolerant scheme by the accuracy results from different neural network benchmarks. For the hardware aspect, we evaluate our architecture design by simulations, showing the performance, energy breakdown, and area overhead.

TABLE I
The SIGHT configuration

SIGHT Configuration @ 1.2 GHz, 32 nm Technology		
Component	**Parameters**	**Spec**
RRAM Crossbar	Precision	2-bit
	Size	128*128
	Number/PE	32
Input/Output Buffer	Size	8KB
	# Banks	4
	Data Rate	64B/cycle
Input Map Buffer	Size	4KB
	# Banks	4
	Data Rate	32B/cycle
ADC	Precision	8
	Number/PE	32
DAC	Precision	1
	Number/PE	32*128
Regulation Array	# Multipliers	32
PE	Number	256
Global Buffer	Size	32KB
	# Banks	8
	Data Rate	128B/cycle
I/O	Config.	PCIe 4.0*4
	Bandwidth	16GB/s

Evaluation Tools: We implement our fault-tolerant schemes with PyTorch, a commonly-used python-based NN framework. We also build up an in-house simulator to evaluate the hardware performance. The RRAM parameters including area, energy, and latency, are derived from NVSim [50], while the SRAM are simulated by CACTI [51]. We also implement other digital components, such as activation units and multipliers, in Verilog and synthesize them with Synopsys Design Compiler to estimate their performance.

Accelerator Configuration: The system is configured to 1.2GHz and simulated under 32nm technology. For each PE, we use 32 RRAM crossbar arrays with a size of 128$\times$128. We assume 2-bit precision for each RRAM cell. We use 1-bit DAC and 8-bit ADC as described in [5], where each RRAM crossbar has 128 DACs and shares one ADC. The regulation array is set to 8-bit precision where it has 32 multipliers. We apply 8KB SRAM for input and output buffer within one PE, and each buffer has 4 banks contributing to a data rate of 64B/cycle. The input map buffer is a 4KB SRAM separately whose data rate is 32B/cycle. We equipped SIGHT with 256 PEs in total, and there is a 32KB global buffer with 128B/cycle data rate. We use four PCIe 4.0 lanes to connect the accelerator and the host, providing 16GB/s off-chip bandwidth in total.

Benchmarks: For better understanding how the model structure and complexity are sensitive to our fault-tolerant scheme, we consider two typical types of CNN, VGG and ResNet, for evaluation, where VGG stands for a plain and straightforward network and ResNet has a residual structure [1], [52]. We also choose models with different depths, including VGG-11/16 and ResNet-18/34. We apply the public implementations from GitHub as baselines [53], [54]. The dataset we use is CIFAR-10, which consists of 60000 32$\times$32 color images in 10 classes.

TABLE II
THE ACCURACY RESULTS OF THE RESISTANCE-AWARE QUANTIZATION. MODELS ARE QUANTIZED INTO DIFFERENT PRECISION FOR DIFFERENT TYPES OF RESISTANCE DISTRIBUTION. FOR POWER AND EXPONENTIAL DISTRIBUTION, DIFFERENT PARAMETERS ARE CHOSEN.

		ResNet-18			ResNet-34			VGG-11			VGG-16		
Accuracy w/ Full Precision		**94.78**			**94.74**			**92.23**			**93.58**		
Quantization Bit		2-bit	3-bit	4-bit	2-bit	3-bit	4-bit	2-bit	3-bit	4-bit	2-bit	3-bit	4-bit
Deviated Linear	δ=0.10	93.47	94.68	94.82	93.39	94.37	94.65	85.94	91.21	91.86	89.23	93.03	93.38
Power	β=$\sqrt{2}$	94.36	94.60	94.73	94.40	94.48	94.59	90.63	91.89	92.15	92.80	93.27	93.48
	β=2	94.38	94.67	94.79	94.14	94.57	94.66	90.64	91.98	92.14	92.72	93.29	93.46
	β=3	94.31	94.51	94.68	94.19	94.56	94.54	90.35	91.34	91.97	92.43	93.11	93.52
Exponential	α=$\sqrt{2}$	93.40	94.71	94.69	93.36	94.59	94.61	85.18	91.91	91.92	88.09	93.30	93.38
	α=2	94.66	94.67	94.72	94.27	94.45	94.52	90.86	91.52	91.55	93.04	93.14	93.24
	α=3	94.45	94.38	94.39	94.14	94.14	94.09	90.15	90.11	90.06	92.81	92.83	92.81

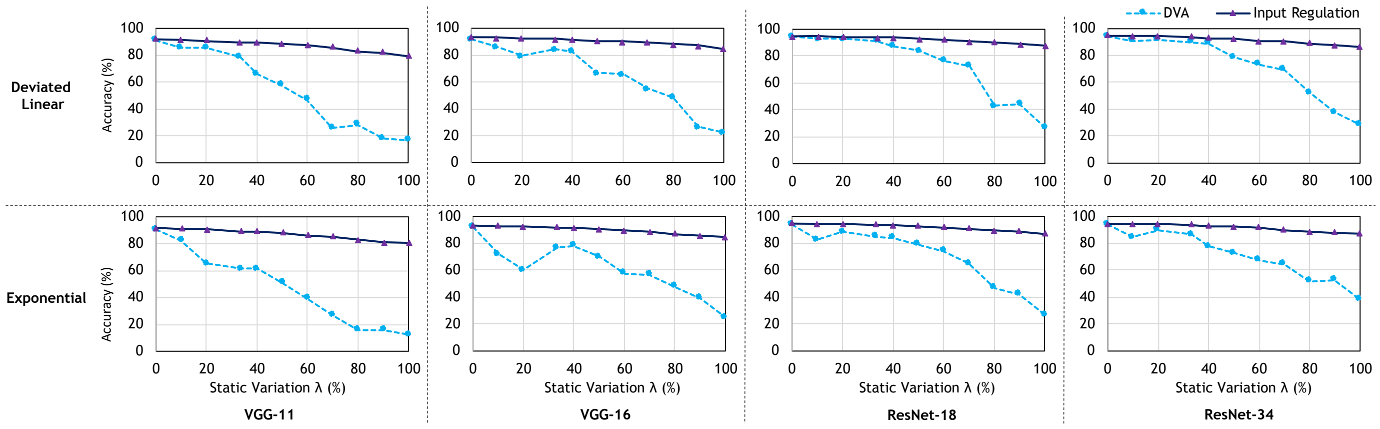


Fig. 10. The efficacy of input regulation against the static variations, compared with device-variation-aware (DVA) training [15]. Deviated-linear and exponential quantization are shown. The x-axis represents the variance of variations, and the bigger the worse. The y-axis represents the accuracy.

Among them, 50000 images are used for training and 10000 images are used for testing [55].

B. Accuracy Results

1) Resistance-Aware Quantization: We quantize and fine-tune the four pre-trained models with the three mentioned resistance distributions. For a better sensitivity study, we quantize them into 2, 3, and 4 bits as current RRAM devices are unlikely to have very high precision. We also select different parameters for different distributions and choose the base of power distribution and the index of exponential distribution to be $\sqrt{2}$,2,3.

As shown in TABLE II, the accuracy result under full precision can be found in the top line while the accuracy after quantization is below them. We find that: (a) For most cases, we achieve no accuracy loss or insignificant accuracy loss ($\sim$1%), demonstrating the efficacy of our resistance-aware quantization; (b) The overall results for ResNet models are better than VGG ones. We resume the accuracy of ResNet models back to 94% as original for almost all the cases, except for exponential quantization to 2 bits where the accuracy is 93%. Meanwhile, VGG models suffer more accuracy loss as usually 1-2% degradation is introduced, especially for VGG-11. This tells the ResNet structure is more robust than a VGG-like plain network when applying the non-linear quantization; (c) The exponential distribution with a small index plus low quantization width will degrade the performance noticeably. As shown, when using parameter α=$\sqrt{2}$ and quantizing the model with 2-bit precision, the accuracy of ResNet models drops 1% while the accuracy of VGG-11/VGG-16 decreases to 85.18% and 88.09%. This is because in such cases, the quantized values have a smaller range and thus are not enough to represent the NN models. Also, higher quantization precision generally makes quantized values more representative, which explains that it basically occurs at the low-bit quantization.

2) Input Regulation: To evaluate the efficacy of input regulation, we inject variations with increasing variances to see how models get affected by static variations, where a larger variance indicates a worse variation. We take the deviated-linear and exponential quantization methods for all the four models and recover the accuracy by regulating the input.

The results are shown in Fig. 10, where we compare the accuracy between the device-variation-aware (DVA) training [15] (shown as dotted lines) and our proposed input regulation (shown as solid lines). We observe that: (a) With the input regulation, we keep the accuracy against dropping from variations. Even when injecting the variation as large as 100%, the model delivers less than 10% accuracy loss most of the time; (b) The input regulation outperforms the DVA with an average accuracy gain of 25.2%. The DVA appears effective in a smaller variation range when $\lambda < 50\%$, but the accuracy degrades significantly after that. Specifically, our input regulation achieves an accuracy gain of 13.0% when $0 < \lambda \leq 50\%$ and an accuracy gain of 43.7% when $50 < \lambda \leq 100\%$, compared with the DVA. (c) The DVA shows stochastic characteristics, as in some cases such as the VGG-16, smaller variation may cause larger accuracy loss. This is because the real variation scenario is unpredictable. Although the model is trained to be more robust, the later injected variation could cause huge accuracy loss. On the opposite, our

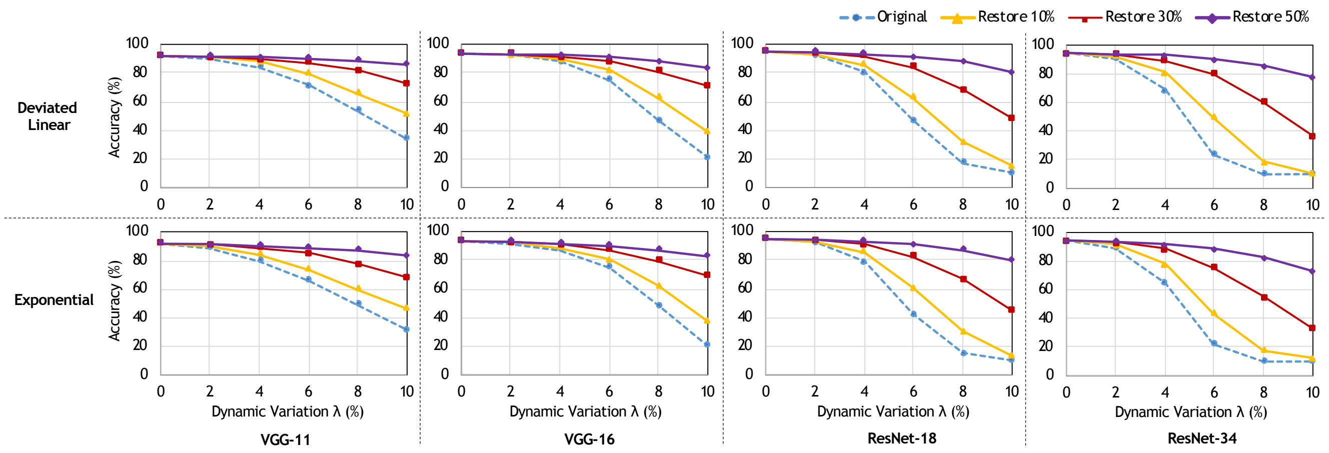


Fig. 11. The efficacy of selective weight restoration against the dynamic variations. Deviated-linear and exponential quantization are shown. The x-axis represents the variance of variations, and the bigger the worse. The y-axis represents the accuracy.

input regulation is obtained for a specific variation distribution and thus achieves better performance.

3) Run-Time Weight Restoration: Similar to the static variation, we inject dynamic variations with increasing variances to the model and evaluate the deviated-linearly and exponentially quantized models. We also restore different percentages of weights to see the trade-off between the restoration overhead and model accuracy.

As shown in Fig. 11, the dotted line presents how the accuracy drops as we injected larger dynamic variations, and three solid lines with different markers present the recovery of accuracy after the weight restoration. From the results we find that: (a) As a larger dynamic variation leads to lower model accuracy, the ResNet models suffer more from the dynamic variation. For the deviated linear model with an 8% dynamic variation, the accuracy of ResNet-18/34 drops rapidly to 17.47%/10.43%, while the VGG-11/16 only drops to 54.05%/46.86%. This may be caused by the single-directional drifting for dynamic variations, meaning the weight drifting will accumulate layer by layer. Therefore, deeper networks with more complex structures may have worse accuracy. This hypothesis also stands for the same model with different depths, as VGG-11 and ResNet-18 appear to perform better than VGG-16 and ResNet-34; (b) The weight restoration notably improves the accuracy under dynamic variations. For VGG models, we notice that restoring 30% weight recover 30-50% accuracy when injecting 8% dynamic variations, while further restoring 50% weight would almost resume the whole accuracy. Meanwhile, restoring 10% weight seems not that helpful as it only recovers 10% accuracy. Besides, for ResNet models, it seems that we need to restore about half of the weights to fully recover the accuracy. This can also be explained by the variation accumulating in the NN model since ResNet is much deeper than others. (c) As we take a look into the results for deviated-linear and exponential quantization, there is no much difference between them, implying that quantization methods are not affecting the model accuracy as much as the model itself.

C. Hardware Results

As our fault-tolerant framework is built on top of existing RRAM-based accelerators, we mainly discuss the overhead introduced by SIGHT in the hardware evaluation. We first provide baseline results and present the performance overhead, including inference latency and energy consumption. Then we show the area and power breakdown of SIGHT. Finally, we do a sensitivity study on how the performance of SIGHT is affected by various hardware configurations.

TABLE III
THE BASELINE RESULTS INCLUDING THE LATENCY, ENERGY CONSUMPTION, AND RUN-TIME PERFORMANCE FOR FOUR NN MODELS. THE RESULTS OF ONE IMAGE IS SHOWN.

Model	# Active Xbar	Latency (ms)	Energy (nJ)	Performance (TOP/W)
VGG-11	1164	0.18	3,883	27.03
VGG-16	1840	0.35	8,243	24.18
ResNet-18	1392	0.86	15,766	35.23
ResNet-34	2636	1.34	29,531	19.63

1) Performance: TABLE III shows the baseline results for the four models where our fault-tolerant framework is not enabled. We present the number of active crossbars, latency, energy consumption and run-time performance. Besides, we set 2-bit weight and 8-bit activation for inference and directly map the model with a batch size equal to 1. We first find that ResNet-34 consumes most crossbar resources and other models are using a comparative number of crossbars. Also, the two ResNet models take a much longer time for execution and consume more energy than VGG models. This is because ResNet models have more convolutional layers that require long repeated computation over them. Finally, the run-time performance of the four models is about 20-30 TOP/W.

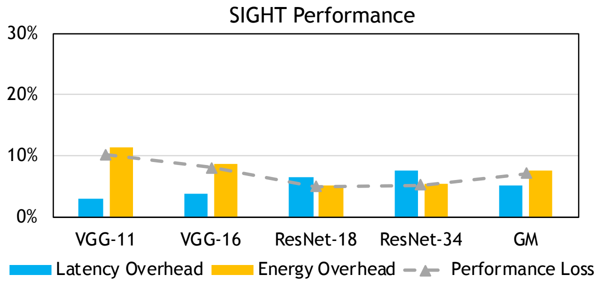


Fig. 12. The performance of SIGHT. The blue bar shows the execution latency while the yellow bar shows the energy consumption, and the dotted line represents the performance.

We show the results of SIGHT in Fig. 12, which are represented by the percentage of overhead compared with the baseline. We set the weight refreshing frequency to 4 images as the weight would be re-loaded every 4 images. First, SIGHT causes a small amount of overhead which is typically less than 10%. The average latency overhead is 5.23%, which is mainly introduced by the regulation array and weight refreshing. The geometric mean of energy overhead is 7.75%, which is slightly larger than latency overhead due to a large amount of RRAM writing, and the overall performance overhead is 7.14%. Second, from the NN model perspective, we find that VGG models have larger energy overhead but smaller latency overhead than ResNet models. This is because VGG models have three large FC layers, and they require more RRAM crossbars but essentially perform less computation, compared with convolutional layers. Therefore, the energy overhead comes from refreshing FC layers in VGG, which is relatively larger than ResNet. On the other hand, as convolutional layers require more computation, its latency suffers more from the regulation array as the array increases data loading latency, so the ResNet appears to have a larger latency overhead.

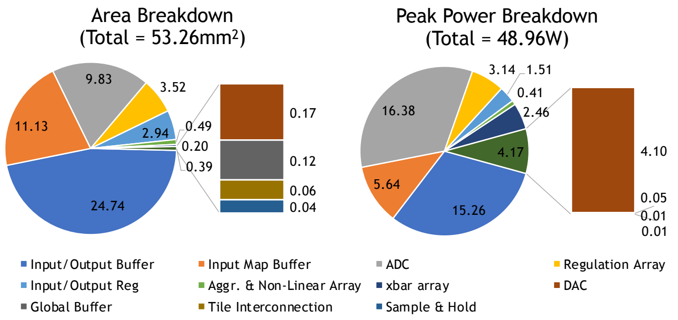


Fig. 13. The area and power breakdown of SIGHT.

2) Area/Power Analysis: We present the area and power breakdown in Fig. 13. As shown, the total area of SIGHT is 53.26 mm^2 and the peak power is 48.96 W. The largest components include the SRAM buffer and ADC, where the input/output buffer takes 24.74 mm^2, the input map buffer takes 11.13 mm^2, and ADC takes 9.83 mm^2. Our add-on components of regulation arrays introduce 3.52 mm^2 area overhead, which is 6.6% of the whole area. The input/output registers for DAC/ADC take 2.94 mm^2 area while other components including RRAM itself, logic unit, DAC and so on occupy a negligible area in the whole architecture.

From the peak power perspective, the SRAM buffer and ADC still consume the most power, where the input/output buffer takes 15.26 W, the input map buffer takes 5.64 W, and ADC takes 16.38 W. The regulation array consumes 3.14 W peak power, which is 6.4% of the whole system. The DAC and RRAM crossbar take another noticeable fraction with 4.10 W and 2.46 W peak power, respectively.

3) Sensitivity Study: To better understand the trade-off in hardware configuration, we also change the design parameters in SIGHT to see how its performance is sensitive to different architecture settings.

Sensitive to Input/Weight Resolution: As shown in Fig. 14, we fix the model to 2-bit weight/8-bit activation, and tune the activation/weight precision respectively to see how the performance overhead changes. First, from the left part where the weight is fixed to 2-bit, we find that with higher activation

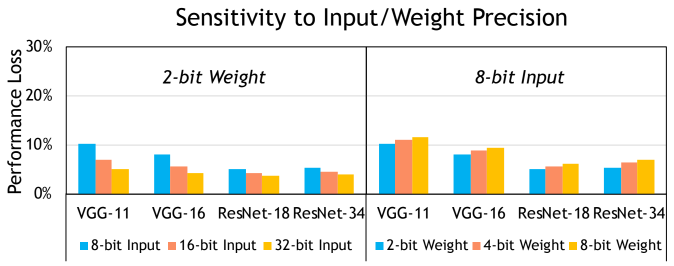


Fig. 14. The performance sensitivity to input and weight precision. The left part fixes weight to 2-bit while the right part fixes input to 8-bit.

precision, the performance overhead is actually reducing. This is because when increasing the activation precision, we do not necessarily need more RRAM crossbars but perform computation over the same RRAM crossbar repeatedly. Then, the proportion of energy consumption in weight refreshing is then decreased relatively, so we have less overall performance loss. On the other hand, when fixing the input precision to 8-bit, higher weight precision would increase the performance loss, as observed in the right part of Fig. 14. Because more RRAM crossbars are needed in such a case and thus more energy overhead is introduced when refreshing the weight.

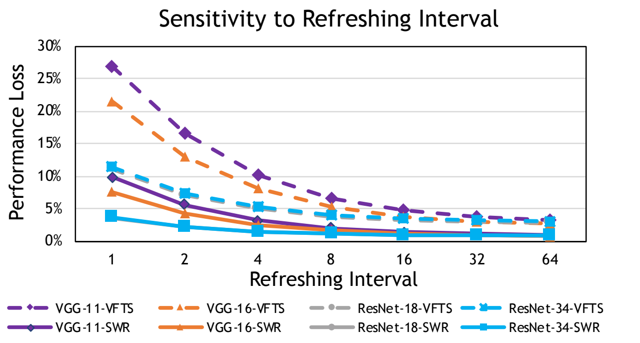


Fig. 15. The performance sensitivity of selective weight restoration (SWR) to different refreshing intervals from refreshing per 1 image to per 64 images, compared with variation-free tuning scheme (VFTS) [25].

Sensitive to Refreshing Interval: Since various RRAM devices emerge based on different materials, they may exhibit unique retention character and thus require different refreshing frequency. As shown in Fig. 15, we present the performance loss of selective weight restoration (SWR, solid lines) when ranging the refreshing interval from every 1 image to every 64 images, with the comparison to variation-free tuning scheme (VFTS, dotted lines) [25]. Our SWR causes much less performance degradation that is less than 10%, compared with VFTS that could lead to near 30% performance loss when refreshing the weight frequently. Also, the performance loss can be reduced significantly by increasing the refreshing interval, especially when the interval is less than 8. The benefit mainly comes from the reduction of relative energy consumption of weight refreshing. These observations demonstrate that the proposed SWR enjoys more benefits when the RRAM device is more vulnerable to dynamic variations.

VII. Discussion

A. Other Emerging Non-Volatile Memories

Many other non-volatile memory technologies emerge besides RRAM, such as Phase Change Memory (PCM) and magnetoresistive random access memory (MRAM).MRAM uses Magnetic Tunnel Junction (MTJ) to store the binary information with low and high cell resistance shown in different ferromagnetic directions [56], while PCM leverages the large resistance gap between crystalline (low resistance) and amorphous (high resistance) phases of the phase change material [57]. We mostly focus on the metal–oxide RRAM [20] because it generally shows longer endurance of 10^{12} [58], [59] than PCM whose endurance is about 10^6-10^9 [60], and MRAM used to have binary states instead of multiple states with higher density [56]. Besides, researchers also found that PCM may also exhibit similar non-ideal issues like variation as RRAM [57], and our techniques can also be applied in a PCM-based architecture.

B. Non-ideal Factors in Other Components

Our SIGHT architecture consists of not only the RRAM but also other components which may also introduce computation error caused by the nonideality, such as the ADC and non-linear logic. For the ADC, based on the mature CMOS processing technology, recent work has achieved the resolution of 12-14 bits and a high SNR of more than 70dB [61], [62]. This indicates that the noise inside the ADC is negligible compared with the large variation that exists in RRAM as shown in Fig. 10. For the non-linear activation in the SIGHT, since current CNNs mostly use ReLU as the activation function that simply turns negative values to zeros [63], it would not introduce any nonideality in the digital side.

VIII. Related Work

RRAM-based NN Accelerator: ISAAC [5] is a memristor-based CNN accelerator that uses RRAM crossbar to complete multiplying and accumulating. PRIME [4] is an RRAM-based main memory architecture that we can use for both data storage and NN inference. PipeLayer [64] is an accelerator targeting both the inference and training which improves the pipeline design. Ji proposed a programmable RRAM-based accelerator and designed a system stack including the compiler and programming model [65]. Ankit proposed an NN training architecture with RRAM-based outer product [66]. Meanwhile, various simulation tools are also proposed for RRAM-based architecture, such as MemTorch [67].

Enhancing RRAM's Reliability: Li introduced a verify-after-write approach to address the variation problem that tunes the RRAM cell more accurately against the variation [24]. Cheng further improved this solution with a RESET-free approach [25]. Chen proposed a mapping scheme for NN parameters to avoid important weights being in the variation-affected RRAM cells [14]. Yun proposed training the NN model with injected noise to enhance the robustness to RRAM variations [15]. For the endurance problem, Cai presented a row-wise update scheme and reduced the number of writes for RRAM-based training [49]. Also, researchers have made lots of efforts on hard errors [68], [69], which mainly refers to the stuck-at-fault where the resistance of one RRAM cell gets stuck and cannot be tuned anymore. For example, Xia proposed a matrix remapping scheme that leverages redundant RRAM crossbars to avoid the stuck cells [68]. Liu proposed a NN retraining algorithm with the awareness of RRAM defects to solve the problem of stuck-at-faults [69].

IX. Conclusion

In this paper, we present SIGHT, a SynergIstic alGorithm-arcHitecture fault-Tolerant framework, to holistically address the reliability issues in RRAM devices. Specifically, we consider three major types of faults for RRAM computing: non-linear resistance distribution, static variation, and dynamic variation. From the algorithm level, we propose a resistance-aware quantization to compel the neural network parameters to follow the exact non-linear resistance distribution as RRAM and introduce an input regulation technique to compensate for RRAM variations. We also propose a selective weight refreshing scheme to address the dynamic variation issue that occurs at run-time. Finally, we propose a *general* and *low-cost* architecture accordingly for supporting our fault-tolerant scheme. Our evaluation demonstrates almost no accuracy loss for our fault-tolerant algorithms, and the SIGHT architecture incurs performance overhead as little as 7.14%.

References

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[2] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 580–587.

[3] M. Sundermeyer, R. Schlüter, and H. Ney, "Lstm neural networks for language modeling," in *Thirteenth annual conference of the international speech communication association*, 2012.

[4] P. Chi, S. Li, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, and Y. Xie, "Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 27–39, 2016.

[5] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. P. Strachan, M. Hu, R. S. Williams, and V. Srikumar, "Isaac: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," *ACM SIGARCH Computer Architecture News*, vol. 44, pp. 14–26, 2016.

[6] B. Li, P. Gu, Y. Shan, Y. Wang, Y. Chen, and H. Yang, "Rram-based analog approximate computing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2015.

[7] X. Hong, D. J. Loy, P. A. Dananjaya, F. Tan, C. Ng, and W. Lew, "Oxide-based rram materials for neuromorphic computing," *Journal of materials science*, vol. 53, no. 12, pp. 8720–8746, 2018.

[8] Y. Wu, B. Lee, and H.-S. P. Wong, "Ultra-low power al 2 o 3-based rram with 1μa reset current," in *Proceedings of 2010 International Symposium on VLSI Technology, System and Application*. IEEE, 2010, pp. 136–137.

[9] Y. Hosoi, Y. Tamai, T. Ohnishi, K. Ishihara, T. Shibuya, Y. Inoue, S. Yamazaki, T. Nakano, S. Ohnishi, N. Awaya *et al.*, "High speed unipolar switching resistance ram (rram) technology," in *2006 International Electron Devices Meeting*. IEEE, 2006, pp. 1–4.

[10] C.-Y. Chen, H.-C. Shih, C.-W. Wu, C.-H. Lin, P.-F. Chiu, S.-S. Sheu, and F. T. Chen, "Rram defect modeling and failure analysis based on march test and a novel squeeze-search scheme," *IEEE Transactions on Computers*, vol. 64, no. 1, pp. 180–190, 2014.

[11] B. Traoré, P. Blaise, E. Vianello, L. Perniola, B. De Salvo, and Y. Nishi, "Hfo 2-based rram: Electrode effects, ti/hfo 2 interface, charge injection, and oxygen (o) defects diffusion through experiment andab initiocalculations," *IEEE Transactions on Electron Devices*, 2015.

[12] W. Chien, Y. Chen, K. Chang, E. Lai, Y. Yao, P. Lin, J. Gong, S. Tsai, S. Hsieh, C. Chen *et al.*, "Multi-level operation of fully cmos compatible wox resistive random access memory (rram)," in *2009 IEEE International Memory Workshop*. IEEE, 2009, pp. 1–2.

[13] S. R. Lee, Y.-B. Kim, M. Chang, K. M. Kim, C. B. Lee, J. H. Hur, G.-S. Park, D. Lee, M.-J. Lee, C. J. Kim *et al.*, "Multi-level switching of triple-layered taox rram with excellent reliability for storage class memory," in *2012 Symposium on VLSI Technology*. IEEE, 2012.

[14] L. Chen, J. Li, Y. Chen, Q. Deng, J. Shen, X. Liang, and L. Jiang, "Accelerator-friendly neural-network training: Learning variations and defects in rram crossbar," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*. IEEE, 2017, pp. 19–24.

[15] Y. Long, X. She, and S. Mukhopadhyay, "Design of reliable dnn accelerator with un-reliable reram," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2019, pp. 1769–1774.

[16] S.-S. Sheu, M.-F. Chang, K.-F. Lin, C.-W. Wu, Y.-S. Chen, P.-F. Chiu, C.-C. Kuo, Y.-S. Yang, P.-C. Chiang, W.-P. Lin *et al.*, "A 4mb embedded slc resistive-ram macro with 7.2 ns read-write random-access time and 160ns mlc-access capability," in *2011 IEEE International Solid-State Circuits Conference*. IEEE, 2011, pp. 200–202.

[17] J. Woo, K. Moon, J. Song, M. Kwak, J. Park, and H. Hwang, "Optimized programming scheme enabling linear potentiation in filamentary hfo 2 rram synapse for neuromorphic systems," *IEEE Transactions on Electron Devices*, vol. 63, no. 12, pp. 5064–5067, 2016.

[18] X. Guan, S. Yu, and H.-S. P. Wong, "On the switching parameter variation of metal-oxide rram—part i: Physical modeling and simulation methodology," *IEEE Transactions on electron devices*, 2012.

[19] X. Guan, S. Yu, and H. P. Wong, "On the variability of hfox rram: From numerical simulation to compact modeling," in *Proc. Workshop Compact Models*, 2012, pp. 815–820.

[20] H.-S. P. Wong, H.-Y. Lee, S. Yu, Y.-S. Chen, Y. Wu, P.-S. Chen, B. Lee, F. T. Chen, and M.-J. Tsai, "Metal–oxide rram," *Proceedings of the IEEE*, vol. 100, no. 6, pp. 1951–1970, 2012.

[21] Y. Y. Chen, L. Goux, S. Clima, B. Govoreanu, R. Degraeve, G. S. Kar, A. Fantini, G. Groeseneken, D. J. Wouters, and M. Jurczak, "Endurance/retention trade-off on hfo_2 metal cap 1t1r bipolar rram," *IEEE Transactions on electron devices*, pp. 1114–1121, 2013.

[22] C. Cagli, D. Ielmini, F. Nardi, and A. L. Lacaita, "Evidence for threshold switching in the set process of nio-based rram and physical modeling for set, reset, retention and disturb prediction," in *2008 IEEE International Electron Devices Meeting*. IEEE, 2008, pp. 1–4.

[23] Y. Y. Chen, M. Komura, R. Degraeve, B. Govoreanu, L. Goux, A. Fantini, N. Raghavan, S. Clima, L. Zhang, A. Belmonte *et al.*, "Improvement of data retention in hfo 2/hf 1t1r rram cell under low operating current," in *IEEE International Electron Devices Meeting*. IEEE, 2013, pp. 10–1.

[24] B. Li, P. Gu, Y. Shan, Y. Wang, Y. Chen, and H. Yang, "Rram-based analog approximate computing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 1905–1917, 2015.

[25] M. Cheng, L. Xia, Z. Zhu, Y. Cai, Y. Xie, Y. Wang, and H. Yang, "Time: A training-in-memory architecture for rram-based deep neural networks," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 5, pp. 834–847, 2018.

[26] J. Lin, L. Xia, Z. Zhu, H. Sun, Y. Cai, H. Gao, M. Cheng, X. Chen, Y. Wang, and H. Yang, "Rescuing memristor-based computing with non-linear resistance levels," in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2018, pp. 407–412.

[27] C. Lammie, O. Krestinskaya, A. James, and M. R. Azghadi, "Variation-aware binarized memristive networks," in *26th IEEE International Conference on Electronics, Circuits and Systems*, 2019, pp. 490–493.

[28] H. D. Beale, H. B. Demuth, and M. Hagan, "Neural network design," *Pws, Boston*, 1996.

[29] W. Zhao, M. Moreau, E. Deng, Y. Zhang, J.-M. Portal, J.-O. Klein, M. Bocquet, H. Aziza, D. Deleruyelle, C. Muller *et al.*, "Synchronous non-volatile logic gate design based on resistive switching memories," *IEEE Transactions on Circuits and Systems*, no. 2, pp. 443–454, 2013.

[30] L.-E. Yu, S. Kim, M.-K. Ryu, S.-Y. Choi, and Y.-K. Choi, "Structure effects on resistive switching of al/tio_x/al devices for rram applications," *IEEE electron device letters*, vol. 29, no. 4, pp. 331–333, 2008.

[31] U. Russo, D. Ielmini, C. Cagli, and A. L. Lacaita, "Filament conduction and reset mechanism in nio-based resistive-switching memory (rram) devices," *IEEE Transactions on Electron Devices*, pp. 186–192, 2009.

[32] D. Ielmini, "Modeling the universal set/reset characteristics of bipolar rram by field-and temperature-driven filament growth," *IEEE Transactions on Electron Devices*, vol. 58, no. 12, pp. 4309–4317, 2011.

[33] L. Xia, P. Gu, B. Li, T. Tang, X. Yin, W. Huangfu, S. Yu, Y. Cao, Y. Wang, and H. Yang, "Technological exploration of rram crossbar array for matrix-vector multiplication," *Journal of Computer Science and Technology*, vol. 31, no. 1, pp. 3–19, 2016.

[34] D. Fujiki, S. Mahlke, and R. Das, "In-memory data parallel processor," *ACM SIGPLAN Notices*, vol. 53, no. 2, pp. 1–14, 2018.

[35] J. Lin, S. Li, X. Hu, L. Deng, and Y. Xie, "Cnnwire: Boosting convolutional neural network with winograd on reram based accelerators," in *Proceedings of the Great Lakes Symposium on VLSI*, 2019, pp. 283–286.

[36] Z. Zhu, J. Lin, M. Cheng, L. Xia, H. Sun, X. Chen, Y. Wang, and H. Yang, "Mixed size crossbar based rram cnn accelerator with overlapped mapping method," in *ICCAD*, 2018.

[37] F. Alibart, L. Gao, B. D. Hoskins, and D. B. Strukov, "High precision tuning of state for memristive devices by adaptable variation-tolerant algorithm," *Nanotechnology*, vol. 23, no. 7, p. 075201, 2012.

[38] C. Nail, G. Molas, P. Blaise, G. Piccolboni, B. Sklenard, C. Cagli, M. Bernard, A. Roule, M. Azzaz, E. Vianello *et al.*, "Understanding rram endurance, retention and window margin trade-off using experimental results and simulations," in *2016 IEEE International Electron Devices Meeting (IEDM)*. IEEE, 2016, pp. 4–5.

[39] D. Ielmini, F. Nardi, C. Cagli, and A. L. Lacaita, "Trade-off between data retention and reset in nio rrams," in *2010 IEEE International Reliability Physics Symposium*. IEEE, 2010, pp. 620–626.

[40] A. Fantini, L. Goux, R. Degraeve, D. Wouters, N. Raghavan, G. Kar, A. Belmonte, Y.-Y. Chen, B. Govoreanu, and M. Jurczak, "Intrinsic switching variability in hfo 2 rram," in *2013 5th IEEE International Memory Workshop*. IEEE, 2013, pp. 30–33.

[41] S. Ambrogio, S. Balatti, Z. Q. Wang, Y.-S. Chen, H.-Y. Lee, F. T. Chen, and D. Ielmini, "Data retention statistics and modelling in hfo 2 resistive switching memories," in *2015 IEEE International Reliability Physics Symposium*. IEEE, 2015, pp. MY–7.

[42] B. Gao, J. Kang, H. Zhang, B. Sun, B. Chen, L. Liu, X. Liu, R. Han, Y. Wang, Z. Fang *et al.*, "Oxide-based rram: Physical based retention projection," in *2010 Proceedings of the European Solid State Device Research Conference*. IEEE, 2010, pp. 392–395.

[43] Y. Xu, Y. Wang, A. Zhou, W. Lin, and H. Xiong, "Deep neural network compression with single and multiple level quantization," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[44] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[45] S. Wu, G. Li, F. Chen, and L. Shi, "Training and inference with integers in deep neural networks," *arXiv preprint arXiv:1802.04680*, 2018.

[46] E. H. Lee, D. Miyashita, E. Chai, B. Murmann, and S. S. Wong, "Lognet: Energy-efficient neural networks using logarithmic computation," in *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2017, pp. 5900–5904.

[47] D. Miyashita, E. H. Lee, and B. Murmann, "Convolutional neural networks using logarithmic data representation," *arXiv preprint arXiv:1603.01025*, 2016.

[48] S. Jain, A. Sengupta, K. Roy, and A. Raghunathan, "Rxnn: A framework for evaluating deep neural networks on resistive crossbars," *IEEE TCAD*, 2020.

[49] Y. Cai, Y. Lin, L. Xia, X. Chen, S. Han, Y. Wang, and H. Yang, "Long live time: improving lifetime for training-in-memory engines by structured gradient sparsification," in *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*. IEEE, 2018, pp. 1–6.

[50] X. Dong, C. Xu, Y. Xie, and N. P. Jouppi, "Nvsim: A circuit-level performance, energy, and area model for emerging nonvolatile memory," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 31, no. 7, pp. 994–1007, 2012.

[51] S. Thoziyoor, N. Muralimanohar, J. H. Ahn, and N. P. Jouppi, "Cacti 5.1," Technical Report HPL-2008-20, HP Labs, Tech. Rep., 2008.

[52] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[53] C.-Y. Fu, "pytorch-vgg-cifar10," 2019. [Online]. Available: https://github.com/chengyangfu/pytorch-vgg-cifar10

[54] K. Jordan, "PyTorch-ResNet-CIFAR10," 2018. [Online]. Available: https://github.com/KellerJordan/ResNet-PyTorch-CIFAR10

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

[56] X. Dong, X. Wu, G. Sun, Y. Xie, H. Li, and Y. Chen, "Circuit and microarchitecture evaluation of 3d stacking magnetic ram (mram) as a universal memory replacement," in *2008 45th ACM/IEEE Design Automation Conference*. IEEE, 2008, pp. 554–559.

[57] H.-S. P. Wong, S. Raoux, S. Kim, J. Liang, J. P. Reifenberg, B. Rajendran, M. Asheghi, and K. E. Goodson, "Phase change memory," *Proceedings of the IEEE*, vol. 98, no. 12, pp. 2201–2227, 2010.

[58] M.-J. Lee, C. B. Lee, D. Lee, S. R. Lee, M. Chang, J. H. Hur, Y.-B. Kim, C.-J. Kim, D. H. Seo, S. Seo *et al.*, "A fast, high-endurance and scalable non-volatile memory device made from asymmetric ta 2 o 5- x/tao 2- x bilayer structures," *Nature materials*, vol. 10, no. 8, pp. 625–630, 2011.

[59] C.-W. Hsu, I.-T. Wang, C.-L. Lo, M.-C. Chiang, W.-Y. Jang, C.-H. Lin, and T.-H. Hou, "Self-rectifying bipolar tao x/tio 2 rram with superior endurance over 10 12 cycles for 3d high-density storage-class memory," in *2013 Symposium on VLSI Technology*. IEEE, 2013, pp. T166–T167.

[60] M. K. Qureshi, J. Karidis, M. Franceschini, V. Srinivasan, L. Lastras, and B. Abali, "Enhancing lifetime and security of pcm-based main memory with start-gap wear leveling," in *2009 42nd Annual IEEE/ACM international symposium on microarchitecture*, 2009, pp. 14–23.

[61] A. Abumurad and K. Choi, "Increasing the adc precision with oversampling in a flash adc," in *2012 IEEE 11th International Conference on Solid-State and Integrated Circuit Technology*. IEEE, 2012, pp. 1–4.

[62] Y. Chiu, P. R. Gray, and B. Nikolic, "A 14-b 12-ms/s cmos pipeline adc with over 100-db sfdr," *IEEE Journal of Solid-State Circuits*, 2004.

[63] R. Zaheer and H. Shaziya, "Gpu-based empirical evaluation of activation functions in convolutional neural networks," in *2018 International Conference on Inventive Systems and Control*. IEEE, 2018.

[64] L. Song, X. Qian, H. Li, and Y. Chen, "Pipelayer: A pipelined reram-based accelerator for deep learning," in *2017 IEEE International Symposium on High Performance Computer Architecture*, 2017, pp. 541–552.

[65] Y. Ji, Y. Zhang, X. Xie, S. Li, P. Wang, X. Hu, Y. Zhang, and Y. Xie, "Fpsa: A full system stack solution for reconfigurable reram-based nn accelerator architecture," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019, pp. 733–747.

[66] A. Ankit, I. El Hajj, S. Agarwal, M. Marinella, M. Foltin, J.-P. Strachan, D. S. Milojicic, W.-M. W. Hwu, K. Roy *et al.*, "Panther: A programmable architecture for neural network training harnessing energy-efficient reram," *IEEE Transactions on Computers*, 2020.

[67] C. Lammie, W. Xiang, B. Linares-Barranco, and M. R. Azghadi, "Memtorch: An open-source simulation framework for memristive deep learning systems," *arXiv preprint arXiv:2004.10971*, 2020.

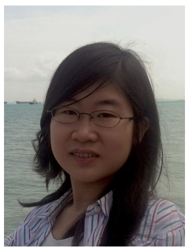
[68] L. Xia, W. Huangfu, T. Tang, X. Yin, K. Chakrabarty, Y. Xie, Y. Wang, and H. Yang, "Stuck-at fault tolerance in rram computing systems," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 2017.

[69] C. Liu, M. Hu, J. P. Strachan, and H. Li, "Rescuing memristor-based neuromorphic design with high defects," in *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE, 2017, pp. 1–6.

Jilan Lin Jilan Lin received his B.S. degree in 2018 from the Department of Electronic Engineering at Tsinghua University, Beijing, China, and he is currently a PhD student at UCSB supervised by Prof. Yuan Xie. His research mainly focuses on machine learning, domain-specific architectures, and in-memory processing.

Cheng-Da Wen Cheng-Da Wen received the B.S. in Economics from National Central University, Taoyuan, Taiwan and he is currently a master student in Computer Science and Information Engineering, National Cheng Kung University, Tainan, Taiwan. His current research interests include the reliability of resistive random access memory (RRAM) and circuit design for artificial intelligence.

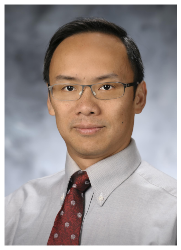
Xing Hu received the B.S. degree from Huazhong University of Science and Technology, Wuhan, China, and Ph.D. degree from University of Chinese Academy of Sciences, Beijing, China, in 2009 and 2014, respectively. She is currently a Postdoc at the Department of Electrical and Computer Engineering, University of California, Santa Barbara, CA, USA. Her current research interests include emerging memory system, and domain-specific hardware computing. She publishes over 30 papers in major conferences and journals including MICRO, HPCA, ASPLOS, DAC, DATE, TCAD, TVLSI, TACO, etc. She serves as reviewer for a number of journals and conferences.



Tianqi Tang Tianqi Tang received her B.S. degree in 2014 and her M.S. degree in 2017 from the Department of Electronic Engineering in Tsinghua University, Beijing, China, and she is currently a Ph.D. student in UCSB under the supervision of Prof. Yuan Xie. Her research mainly focuses on hardware modeling, domain specific accelerator, and processing-in-memory.

Ing-Chao Lin Ing-Chao Lin (M'09–SM'14) received the M.S. degree in computer science from the National Taiwan University, Taipei, Taiwan, and the Ph.D. degree from the Computer Science and Engineering Department, Pennsylvania State University, State College, PA, USA, in 2007. From 2007 to 2009, he was with Real Intent, Inc., Sunnyvale, CA, USA, and since 2009, he has been with the Department of Computer Science and Information Engineering, National Cheng Kung University, Tainan, Taiwan, where he is currently a Full Professor. He was a visiting scholar at University of California, Santa Barbara in 2015, and he was a visiting scholar at Academia Sinica in 2017. His current research interests include very large-scale integration design and computer-aided design for nanoscale silicon, energy-efficient reliable system design, and computer architecture. He has authored or co-authored more than 50 in referred journals and conference papers and he has served on the technical program committee of many prestigious technical conferences, such as ASP-DAC, ICCAD, ICCD, and GLSVLSI. He was the recipient of the 2015 Excellent Young Researcher Award from the Chinese Institute of Electrical Engineering and the 2016 Best Young Professional award (Formally GOLD) from the IEEE Tainan Section. He was awarded the Humboldt Fellowship for Experienced Researcher by the Alexander von Humboldt Foundation, Germany in 2019. He has been a senior member of the IEEE since 2015, and he has been a senior member of the ACM since 2016. He is the contest Chair of the 2020 CAD Contest at ICCAD.

Yu Wang Yu Wang (S05-M07-SM14) received the BS and PhD (with honor) degrees from Tsinghua University, Beijing, in 2002 and 2007. He is currently a tenured professor with the Department of Electronic Engineering, Tsinghua University. His research interests include brain inspired computing, application specific hardware computing, parallel circuit analysis, and power/reliability aware system design methodology. He has authored and coauthored more than 200 papers in refereed journals and conferences. He has received Best Paper Award in ASPDAC 2019, FPGA 2017, NVMSA 2017, ISVLSI 2012, and Best Poster Award in HEART 2012 with 9 Best Paper Nominations (DATE18, DAC17, ASPDAC16, ASPDAC14, ASPDAC12, 2 in ASPDAC10, ISLPED09, CODES09). He is a recipient of DAC under 40 innovator award (2018), IBM X10 Faculty Award (2010). He served as TPC chair for ICFPT 2019 and 2011, ISVLSI2018, finance chair of ISLPED 2012-2016, track chair for DATE 2017-2019 and GLSVLSI 2018, and served as program committee member for leading conferences in these areas, including top EDA conferences such as DAC, DATE, ICCAD, ASP-DAC, and top FPGA conferences such as FPGA and FPT. Currently, he serves as the associate editor of the IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, the IEEE Transactions on Circuits and Systems for Video Technology, the Journal of Circuits, Systems, and Computers, and Special Issue editor of the Microelectronics Journal. He is now with ACM Distinguished Speaker Program.



Yuan Xie Dr. Xie received B.S degree from Tsinghua University, MS and PhD degree from Princeton University. He is IEEE Fellow, ACM Fellow, and AAAS Fellow. He has published 3 books, 100+ journals, and more than 200 refereed conference papers, and holds 6 patents. His research interests include VLSI design, EDA, computer architecture, embedded systems, with a focus on application-driven and technology-driven novel circuits/architectures and design methodologies. He is a recipient of NSF Career award, IEEE Computer Society Edward J. McCluskey Technical Achievement Award, and many best paper awards.