Long Live TIME: Improving Lifetime and Security for NVM-Based Training-in-Memory Systems

Yi Cai, Yujun Lin, Lixue Xia, *Student Member, IEEE*, Xiaoming Chen, *Member, IEEE*, Song Han, Yu Wang, *Senior Member, IEEE*, and Huazhong Yang, *Fellow, IEEE*

***Abstract*—Nonvolatile memory (NVM)-based training-in-memory (TIME) systems have emerged that can process the neural network (NN) training in an energy-efficient manner. However, the endurance of NVM cells is disappointing, rendering concerns about the lifetime of TIME systems, because the weights of NN models always need to be updated for thousands to millions of times during training. Gradient sparsification (GS) can alleviate this problem by preserving only a small portion of the gradients to update the weights. However, conventional GS will introduce nonuniform writes on different cells across the whole NVM crossbars, which significantly reduces the excepted available lifetime. Moreover, an adversary can easily launch malicious training tasks to exactly wear-out the target cells and fast break down the system. In this article, we propose an efficient and effective framework, referred as *SGS-ARS*, to improve the lifetime and security of TIME systems. The framework mainly contains a structured GS (SGS) scheme for reducing the write frequency, and an aging-aware row swapping (ARS) scheme to make the writes uniform. Meanwhile, we show that the back-propagation mechanism allows the attacker to localize and update fixed memory locations and wear them out. Therefore, we introduce Random-ARS and Refresh techniques to thwart adversarial training attacks, preventing the systems from being fast broken in an extremely short time. Our experiments show that when TIME is programmed to train ResNet-50 on ImageNet dataset, 356× lifetime extension can be achieved without sacrificing the accuracy much or incurring much hardware overhead. Under the adversarial environment, the available lifetime of TIME systems can still be improved by 84×.**

***Index Terms*—Gradient sparsification, lifetime, neural networks, training-in-memory, wear-leveling.**

Manuscript received September 30, 2019; revised December 10, 2019; accepted February 3, 2020. Date of publication February 28, 2020; date of current version November 20, 2020. This work was supported in part by the National Key Research and Development Program of China under Grant 2017YFA0207600, in part by the National Natural Science Foundation of China under Grant 61832007, Grant 61622403, and Grant 61621091, in part by the Beijing National Research Center for Information Science and Technology, and in part by the Beijing Innovation Center for Future Chips. The work of Xiaoming Chen was supported by the Beijing Academy of Artificial Intelligence. This article was recommended by Associate Editor R. S. Chakraborty. *(Corresponding author: Yu Wang.)*

Yi Cai, Yu Wang, and Huazhong Yang are with the Department of Electronic Engineering, Beijing National Research Center for Information Science and Technology, Tsinghua University, Beijing 100084, China (e-mail: yu-wang@tsinghua.edu.cn).

Yujun Lin and Song Han are with the Department of EECS, Massachusetts Institute of Technology, Cambridge, MA 02139 USA.

Lixue Xia is with the Department of Cloud Intelligence, Alibaba Group, Beijing 100022, China.

Xiaoming Chen is with the State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China.

Digital Object Identifier 10.1109/TCAD.2020.2977079

I. Introduction

Deep learning has revolutionized the artificial intelligence (AI) technologies in recent years [1]–[3]. With neural network (NN) architectures becoming increasingly deep and wide, the training of state-of-the-art NNs has put a tremendous pressure on the hardware. Due to the high complexity, the NN training is mostly performed on high-performance GPUs and then move the well-trained models to the edge devices for *inference*. However, there is an increasing demand for empowering learning ability for the edge computing, because 1) many applications usually require online learning [4] to enhance the adaptability and 2) it is insecure and inefficient to off-load data and heavy weights to the cloud due to the bandwidth limitation and privacy protection concerns. Therefore, fast and the energy-efficient computing platforms for NN training are highly demanded, especially for edge devices with tight hardware resource and energy budgets. However, as the inherent memory wall exists in the von Neumann architecture, and the feature size of integrated circuit technology is approaching the physical limit [5], researchers are seeking for emerging technologies to replace CMOS-based processors for higher energy efficiency and performance.

Nonvolatile memories (NVMs), such as resistive random-access memory (RRAM) and phase-change memory (PCM), have many advantages, including high density, low power consumption, and suitability for implementing the crossbar structure to perform matrix-vector multiplications efficiently. Therefore, NVMs have been studied to build training-in-memory (TIME) systems for accelerating the NN training. For instance, the TIME [6] architecture and PipeLayer [7] are both emerged as RRAM-based accelerators for the training of CNNs, achieving over 100× energy efficiency improvement and 42.45× speedup than GPU-based implementations, respectively. The PCM-based NN training platform also achieves 280× and 100× better efficiency and throughput than the most-recent GPUs [8]. All these studies demonstrate a promising path to energy-efficient acceleration of training a variety of NNs by using NVM arrays.

However, the limited endurance of NVM devices becomes a main hurdle for the NVM-based TIME systems. While the endurance of state-of-the-art NVM devices covers a wide range from 10^6 to 10^{12} [9]–[12], they usually demonstrate much lower endurance cycles when constructing cross-point arrays [13] and multivalue computing systems [14]. By applying the widely known stochastic gradient descent (SGD) optimizer, updates of the weight parameters are incurred in every iteration, yielding a write operation on each cell in each update cycle. For example, ResNet-50 [15] needs approximately 5×10^5 iterations to be fully trained on the ImageNet dataset [16]. With an endurance limit of 10^7, NVM-based TIME systems can only execute the training task for $10^7/(5 \times 10^5) = 20$ times. Moreover, large NNs and complicated datasets (such as GoogLeNet [17] trained on ImageNet) usually require millions of iterations to train. Such limited endurance is far insufficient to support large-scale and long-term NN training.

A straightforward idea of improving the lifetime is to reduce the write amount and frequency on the NVM devices. Regarding the NN training application, it is required that updating as fewer weights as possible throughout the training process. On the algorithm side, many researches have demonstrated the feasibility of optimizing NN models with gradient sparsification (GS) [18]. GS accumulates smaller gradients, and only uses larger ones to update the weights in every iteration. The deep gradient compression (DGC) method [19] can drop off 99.9% of the gradients and utilize only the top-0.1% gradients with the largest magnitude, without sacrificing the accuracy. This enlightens us that through GS, the number of writes to NVM cells can be reduced by three orders of magnitudes. As a result, ideally, the lifetime of TIME systems can be enhanced to 1000× longer.

Unfortunately, in our experiments with conventional GS, we observe a severe unbalanced writes among different positions of the weight matrix. It stands to reason that frequently updated weights may have significant effects on the feature extraction. Since an update on a weight value corresponds to a write operation on an NVM cell, the frequently written cells will, undesirably, wear out much quicker than the rest. The work in [20] has proved that only 10% broken cells will lead to the failure of the whole system and cause substantial degradation on the NN performance. Therefore, the write uniformity across NVM crossbars is of paramount importance. Moreover, GS introduces additional computation overhead, mainly introduced by top-k selection. To find the gradients with the top-k largest magnitude out of n elements, the most efficient algorithm has the time complexity of $O(n \log_2 k)$. While in large NNs, if all gradients are involved, such time consumption will greatly slow down the pace of training. This drives us to find a faster and low-overhead way to select the important gradients for weight updates.

Moreover, the lifetime extension should be guaranteed under both a secure and adversarial environment. The training of NNs will necessarily incur write operations on certain cells since the essence of training is to tune the weight parameters. Thus, an attacker can launch a malicious training task that contains carefully designed NN architecture, dataset, and configuration to repeatedly write fixed positions and cause the TIME system fast broken. The wear-leveling techniques can alleviate the aging problem significantly, while it would fail if the techniques are leaked to the attackers. Therefore, the protection solution against adversarial attacks must be taken into account.

This article aims to improve the lifetime and security of NVM-based TIME systems. Specifically, the main contributions of this article are listed as follows.

1) We propose structured GS (SGS) by selecting structured gradients to update weights. Row-wise and element-wise sparsification are introduced for gradient matrices with the different number of rows.
2) We propose an aging-aware row swapping (ARS) method to balance the write count across all rows in crossbars, by which the write unbalance can be efficiently mitigated. The tradeoffs for balancing the overhead and the write uniformity are also discussed to select the optimal hyper-parameters of ARS.
3) We analyze the security vulnerability for the TIME system with limited endurance. We also introduce Random-ARS and Refresh techniques to defend against malicious attacks and enhance the reliability of TIME under adversarial settings.
4) We make thorough experiments to evaluate the effectiveness of the proposed framework. Our experiments demonstrate that the lifetime of TIME can be extended to 356× the original with a negligible overhead and small accuracy sacrifice ($\approx 1\%$) if programmed to process the training of ResNet-50 on ImageNet. Meanwhile, our framework can simultaneously improve the available lifetime by 84× under adversarial settings.

II. Preliminaries and Related Work

A. NVM-Based Neural Network Training

The NVM devices can build efficient computing-in-memory neural computing systems. An NVM cell is an element which has multiple analogue states, and each state can represent a number. Multiple cells can construct crossbars to perform efficient analog matrix-vector multiplications at the location of memory to avoid the data moving. If we map a *matrix* on the conductances of NVM cells in the crossbar, and transform a *vector* as the input voltage signals, the NVM crossbar can perform analog matrix-vector multiplications efficiently. According to Ohm's law and Kirchhoffs current law, the relationship between the input voltages and output current can be represented as: $i_{\text{out}}(z) = \sum_{j=1}^{M} g(z,j) \cdot v_{\text{in}}(j)$, where v_{in} is the input voltage vector (denoted by $j = 1, 2, \ldots, M$), i_{out} is the output current vector (denoted by $z = 1, 2, \ldots, N$), $g(z,j)$ represents the conductance of NVM which in zth row and jth column of the crossbar ($M \times N$). The matrix-vector multiplications dominant the main computation in the convolutional and fully connected (FC) layers, leading to tremendous opportunities for accelerating NN computing by using the NVM crossbars.

The essence of NN training is to search the optimal weight parameters to better fit the inputs and labeled outputs. Mapped on the NVM-based TIME, it incurs the tuning of resistance states (also referred to as write/program operations) of the NVM devices correspondingly. After fetching the gradients of weights in every iteration, the voltage drivers will apply voltage pulses (with various amplitudes or duration) to tune the resistance states to expected values. Thus, in-memory training of NN models can be implemented.

B. Gradient Sparsification

GS was first proposed to resolve the communication bottleneck in distributed NN training by reducing the communication data size by sending only the significant gradients for the

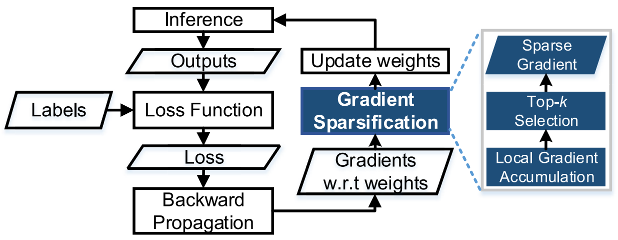


Fig. 1. Flow of training NNs with GS.

weight update [18], [19], [21], as shown in Fig. 1. Two simple heuristics for significance are the gradient magnitude [18], [19] and the ratio of gradient magnitude to the weight magnitude [21]. To avoid losing information, the rest of the gradients are accumulated locally and eventually become large enough to transmit. GS for distributed training is also practical for NN training with a single node. SGD performs the following update:

$$F(w) = \frac{1}{|\chi|}\sum_{x\in\chi} f(x,w),\ \ w_{t+1} = w_t - \eta\frac{1}{b}\sum_{x\in\mathcal{B}_t}\nabla f(x,w_t) \quad (1)$$

where χ is the training dataset, w are the weights of a network, $f(x,w)$ is the loss computed from samples $x \in \chi$, η is the learning rate, and $\mathcal{B}_t$ is a mini-batch of size b sampled from χ at iteration t. Consider the weight value $w^{(i)}$ of ith position in flattened weights w. After T iterations, we have

$$w_{t+T}^{(i)} = w_t^{(i)} - \eta T \cdot \frac{1}{bT}\left(\sum_{\tau=0}^{T-1}\sum_{x\in\mathcal{B}_{t+\tau}}\nabla^{(i)} f(x, w_{t+\tau})\right). \quad (2)$$

Equation (2) shows that local gradient accumulation can be considered as increasing the batch size from b to bT (the first summation over the iteration τ), where T is the length of the sparse update interval between two iterations at which the gradient of $w^{(i)}$ is adopted. Local gradient accumulation ensures the convergence of training with sparse gradients. The sparsity of gradients is able to achieve 99.9% without any loss of accuracy [19]. Therefore, it is promising to substantially reduce the update times by introducing GS in the NN training.

C. Compensation for the Limited Endurance

In addition to the algorithmic innovations, previous work has also intensively discussed the methods for compensating the limited endurance of NVM-based memory system and neural computing system.

Fault-Tolerant Techniques: Fault-tolerant techniques have been proposed to prolong the lifetime and enhance the reliability of TIME or neural computing systems. Typically, redundant or spare units are preset and utilized to replace the wear-out cells. The approach proposed in [22] makes full use of all available mapping space to tolerant stuck-at faults of the RRAM cells. While this article aims at the inference of NN, and with high complexity on algorithm mapping since the models only need be deployed for once, which is not applicable for training. The work in [20] proposes a fault-tolerant training method to reduce the impact of faults in RRAM cells, so that the system availability can still be guaranteed even if errors occur during training, thereby extending the lifetime. They achieve 10× lifetime extension, which is, however, not satisfactory enough for long-term stability.

Write Reduction Techniques: Many work has proposed to prolong the lifetime of NVM-based systems by eliminating the redundant write. For example, in NVM-based main memory applications, *Flip-N-write* scheme [23] has been proposed to reduce the number of bit-flips when writing new data. While in NN application, each NVM cell usually store multiple bits, and every updating will incur a tuning operation on each cell. Therefore, such methods do not fit well with NN training.

Wear-Leveling Techniques: Wear-leveling techniques have been widely studied for endurance-limited memories, such as commonly seen FLASH-based storage devices, PCM, etc. The frequently utilized approaches are to set lifetime management modules to record the written times and rearrange the storage of data [24]–[26]. Periodically, the frequently written memory rows will be remapped to the least written rows (LWRs). While such methods will introduce additional storage overhead, since the registers for write counting are set to record the aging status of all segments, and the mapping tables are required to keep track of the mappings of logical address and physical address. Approaches with less overhead have been also discussed. For example, the *Start-Gap* wear leveling technique proposed in [27] used tow registers to record the *Start* and *Gap* locations when moving the memory rows. However, this method will lose the effectiveness when the write operations concrete in a spatial close region, because the approach can only move a heavy-written row to its neighboring rows. Although all the above techniques perform well on the memory systems, they all have their own shortcomings. The training of NN is always a black box for the trainers, i.e., the updates of the weights are unpredictable but show certain spatial and temporal concentrations. Thus far, the wear-leveling method for in-memory NN training is still in lack.

Programming Optimizations: Previous work has also explored the optimization of conductance state switching to increase the endurance cycles of RRAM [14]. They argue that shrinking the analog switching window under weak tuning pulses can increase the endurance cycles of RRAM by more than five orders of magnitude than the full window switching under strong tuning pulses. However, the nonlinearity and dynamic range of RRAM will degrade, which leads to smaller on/off ratio and the sacrifice of the accuracy of NN training.

Security Protection: Prior work has discussed the security vulnerability of NVM main memory systems, while few studies have discussed the security problem of TIME systems. The simplest way to attack the endurance-limited memories is repeatedly writing a same position [27], [28]. Although wear-leveling methods can alleviate the problem, an obfuscation of data mapping is required to prevent the attackers from knowing the exact physical address of data [28]. Further, online detection [29] can recognize malicious write stream and adopt adaptive wear-leveling to reduce the overhead.

Despite the effectiveness of the above studies for compensating the limited endurance, they have their own shortcomings, and cannot be directly transferred. While they provide promising guidance to design protection solutions for the TIME systems. Besides, many of them are orthogonal to this article, and can be combined to obtain a better improvement.

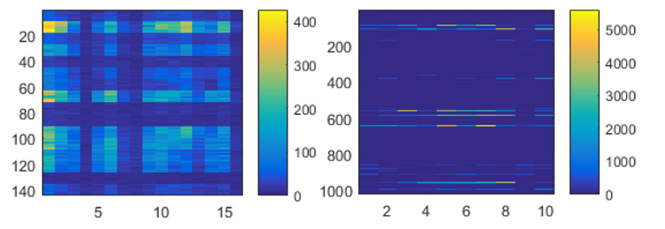


Fig. 2. Overall write distribution of RRAM crossbars. Left: the second CONV layer of ResNet-20. Right: the last FC layer of VGG-11. Total training iteration counts are both 64 000.

III. Motivational Examples

A. Shortcomings of Conventional GS

Theoretically, the conventional GS seems very likely to reduce the total number of write operations. However, two undesired phenomena occur in our simulations of training ResNet-20 and VGG-11 on the CIFAR-10 dataset, which are highly unfriendly to the NVM crossbar and make the improvement on lifetime far less appealing than expected.

Unbalanced Writes: Different positions of FC weight matrices or convolutional (CONV) kernels usually do not share the equal chance to be updated in the conventional GS. After mapping weights to NVM crossbars, it leads to the unbalanced writes on NVM cells. Fig. 2 shows the overall write distributions of two sample NVM crossbars. Each one corresponds to a weight matrix of a layer in the network models. Both of the two NN models are trained for 64 000 iterations. The left one demonstrates the write distribution of the second convolutional layer of ResNet-20 ($3 \times 3 \times 16 \times 16$, reshaped as 144×16), and the right shows the last FC layer of VGG-11 (1024×10).

The distribution maps show a severe unbalance in the total write times throughout the whole matrix. With 99.9% gradient sparsity, the expected write times shall be $64000 \times 0.1\% = 64$ if with an ideal uniformity. Though, in ResNet-20, some cells are written for up to over 400 times, and some are written even less than ten times. It gets even worse in the FC layer of VGG-11 as the maximum of write times surges to 5595. The FC layers show much severer nonuniformity than CONV layers caused by the dense connections. In this occasion, frequently written cells will wear out much more quickly than expected, following by soft faults or stuck-at-faults (SAFs). Therefore, a write-balanced solution is required.

Overhead of Gradient Top-k Selection: Sorting operations are needed to find out the most significant gradients to update the weights, which incurs prohibitively expensive cost. As stated before, the time complexity of selecting top-k from n numbers approaches $O(n \log_2 k)$. Taking the largest CONV layer of VGG-11 as an example, the weights are shaped as $3\times3\times512\times512$. If all $n = 2359296$ gradients are involved for searching the top 0.1% gradients ($k = n \times 0.1\% = 2359$) with the largest magnitude, the computation amount increase by $O(10^7)$. Such large extra cost drives us to design a better sparsification method with less additional overhead.

B. Vulnerability

The TIME systems are also vulnerable to the adversarial attacks. Analogues to attacking NVM-based main memories, the way to attack a TIME system is to force the optimizer repeatedly update a fixed physical location. In NN training application, to exactly force the system to update the target location, the attackers can utilize the nature of gradient generation mechanism. Since the gradients with respect to weights are dependent of both the back-propagated gradients and the inputs, the weight gradients can be finely controlled by specifically designing the inputs. Taking a simplest FC layer as an example, the computation of FC layer is a matrix-vector multiplication. Note the cost function as J, the input feature map of layer l as x_l, the output of layer l as y_l, the forward pass and the gradient generation with respect to the weights w_l can be represented as following:

$$y_j^l = \sum_{i=1}^{N} x_i^l \cdot w_{i,j}^l, \quad \frac{\partial J}{\partial w_{i,j}^l} = \frac{\partial J}{\partial y_j^l} \cdot \frac{\partial y_j^l}{\partial w_{i,j}^l} = \frac{\partial J}{\partial y_j^l} \cdot x_i^l. \tag{3}$$

Through this, an attacker can continuously write the same address by specifically designing the input and the back-propagated loss under the GS mechanism. For example, if the attack goal position is (i, j), the process can set x_i^l and δy_j^l as nonzeros and the others as zeros. Therefore, the gradient located in (i, j) will obviously the largest because it is the only nonzero number in the matrix.

Algorithm 1 Example of Attacking TIME

```
Require: Attack position: $(i, j)$
Require: Generated data and label: $x$, $Label$
Require: Initialized FC weight: $W$, where $W_{i,j} = 1$, $\mathbb{1}(W)=1$
Require: Loss function $J(Pred, Label) = \sum(Pred - Label)^2$
Require: Optimization function $W = W - \mu * \delta W$ w/ gradient sparsification
Require: Learning rate: $\mu = 0.5$
1: while 1 do
2:     Input sample $x$ where $x_i$ = -$W_{i,j}$, $x_k$ = 0 if $k \neq i$
3:     Input target $Label$ where $Label_j$ = 1, $Label_k$ = 0 if $k \neq j$
4:     $Pred$= $W \cdot x$
5:     $\delta Pred = 2 * (Pred - Lable)$
6:     $\delta W = x \cdot \delta Pred^T$
7:     Update the weight $W = W - 0.5 * \delta W$
8: end while
```

We show the malicious process can fast disable fixed locations by the example illustrated in Algorithm 1. We assume that the attacker's goal is destroying the cell located in (i, j) of the cross-point NVM array. Then a simple network and training task can be constructed. Assuming the size of the NVM crossbar as (M,N), an FC layer with M input neurons and N output neurons will be constructed and spread onto the crossbar. Then, Algorithm 1 can force the system continuously write on the goal position. The loop will iteratively switch the $W_{i,j}$ between 1 and -1. Hence, each update will precisely program the target cell between a high-resistance state and low-resistance state. Moreover, the attacks can be extended to simultaneously attack multiple crossbar arrays via multilayer network and multithreading technique.

The motivations behind the malicious process include: an authorized user may deliberately break down the system before the quality assurance expires to get a chance to replace a brand new device; or an unauthorized user (e.g., hackers or competitors) may launch malicious attacks to achieve their illegal purposes. Therefore, a comprehensive solution to prolong the lifetime of TIME systems shall not only ensure the effectiveness under normal use but also thwart the threats under adversarial settings.

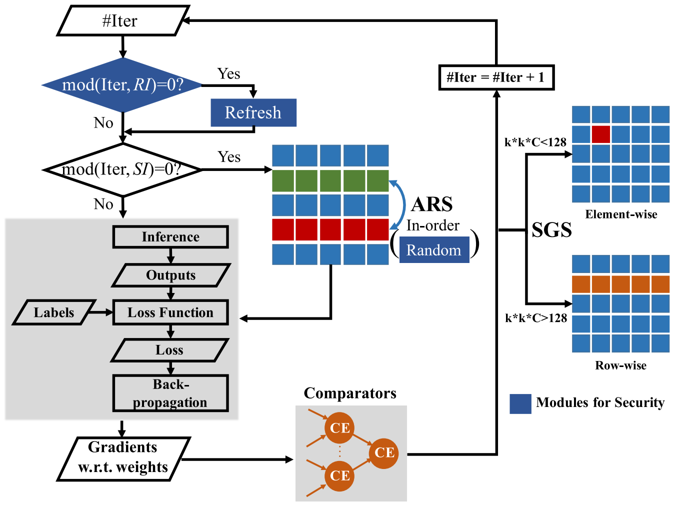


Fig. 3. Overall SGS-ARS framework for improving lifetime of TIME with: 1) SGS for efficiently reducing the update operations; 2) ARS for making the writes uniform; and 3) Random-ARS and Refresh for protecting the TIME systems.

IV. SGS-ARS Framework for TIME

Motivated by the above observations, we propose a simple but effective framework to extend the lifetime of TIME systems under both secure and adversarial settings, referred as the SGS-ARS framework. The overall framework is shown in Fig. 3. Among the main components, the SGS and ARS are carefully designed to reduce the write times and mitigate the unbalanced writes, which will be introduced in Sections V and VI, respectively, in detail. In addition, we introduce random ARS and Refresh mechanisms to enhance the security by obfuscating the mapping of weights in Section VII.

The whole process goes as follows. At the beginning of each iteration, the decisions to perform ARS and Refresh are made based on whether the current iteration number is a multiple of ARS interval [swap interval (SI)] or Refresh interval (RI), followed by the normal inference and backpropagation pass to obtain the gradients. Then the gradients with respect to the weights are sent to the comparator elements (CEs) to select the one with the largest magnitude. Subsequently, SGS is applied to update the weights. A row count threshold (RCT) is used to partition the NN layers into two types, which is set to 128 in our implementation. If the row count of the weight matrix of a layer is less than RCT, the element-wise sparficiation will be applied; otherwise, the row-wise sparsification will be adopted to update weights. These steps will be repeated until the configured maximum number of training iterations.

V. Structured Gradient Sparsification

As observed in Fig. 2, write unbalance introduced by the concentration of sparse updates on weight matrices can be considered as a dual character, since it exhibits a structured pattern. This structured concentration of updated locations facilitates the remapping of weights. In the meantime, structured write operations on the NVM crossbar can be fully parallelized. Inspired by these natural characteristics of GS and NVM crossbar structure, we propose the SGS to overcome the drawbacks of directly applying conventional GS. SGS not only adapts well to NVM by sparsifying the gradients in an crossbar-friendly structured pattern but also significantly reduces the complexity of the sparsification by changing the way to select the gradients in need for the weights update. Fig. 4(a) illustrates the channel-wise, row-wise, and element-wise SGS. To ensure sufficient sparsity, two parts are introduced in this article: 1) the row-wise sparsification and 2) the element-wise sparsification.

A. Row-Wise Sparsification

Fig. 4(b) illustrates the row-wise structured sparsification process. When mapping on NVM crossbars, both FC and CONV layer are treated as matrix–matrix multiplication, since the kernels of CONV layer are reshaped from 4-D tensors ($k \times k \times C \times N$) to matrices ($k^2C \times N$), where k is the kernel size, C is the number of input channels, and N is the number of output channels. Row-wise SGS selects an entire row of weight matrix where the max gradient magnitude lies. Due to the backpropagation computation trait of TIME systems, one row of the gradients can be obtained in each cycle [6], and these gradients are calculated in parallel. Then the maximum gradient magnitude in the row will be popped out. After the last row of gradients is finished, we immediately get the index of the row which contains the maximum magnitude of the whole gradient matrix, and update the corresponding row of weights. Accordingly, the sparsity of row-wise SGS will be $1 - 1/(k^2C)$. Row-wise sparsification is naturally favorable for the crossbar structure, since the crossbar supports writing cells in one row in parallel [30], [31]. It helps significantly reduce the writing cycles for the weight update and enforces the write distribution to be uniform in rows.

In the meantime, we introduce a hyper-parameter N_{update} in the SGS algorithm to determine the number of rows per iteration. A lower N_{update} indicates a higher sparsity of the gradients. Despite that, an increasing sparsity of the gradients will definitely incur less weight updating overhead and enable a longer lifetime, because the number of write operations per iteration decreases, it may also lead to a deteriorating or slowing convergence. The NN models will converge more slowly in the early training stage when applying a larger sparsity, because at the beginning, the NN models can be optimized along various directions as the weights are randomly initialized. And the convergence will gradually catch up the pace of dense gradients in the later stage because of the smaller gradients and more explicit optimization direction. Therefore, we also empirically discover the tradeoffs of balancing the convergence and the overhead in Section VIII.

B. Element-Wise Sparsification.

In small weight matrices with very few rows, the sparsity of row-wise sparsification will be much lower than desired. For example, the first CONV layer of most CNNs has kernel tensors shaped as $k \times k \times 3 \times N$, where k usually ranges from 3 to 7, as the input images commonly have three channels (RGB). In this scenario, the gradient sparsity of row-wise sparsification will be $1 - 1/(3k^2)$. When $k = 3$, only 96.3% sparsity is achieved. To increase the sparsity, a finer-grained sparsification scheme should be exploited. As shown in Fig. 4(c), the element-wise sparsification follows a similar process to the row-wise. The difference is that the index of the column where

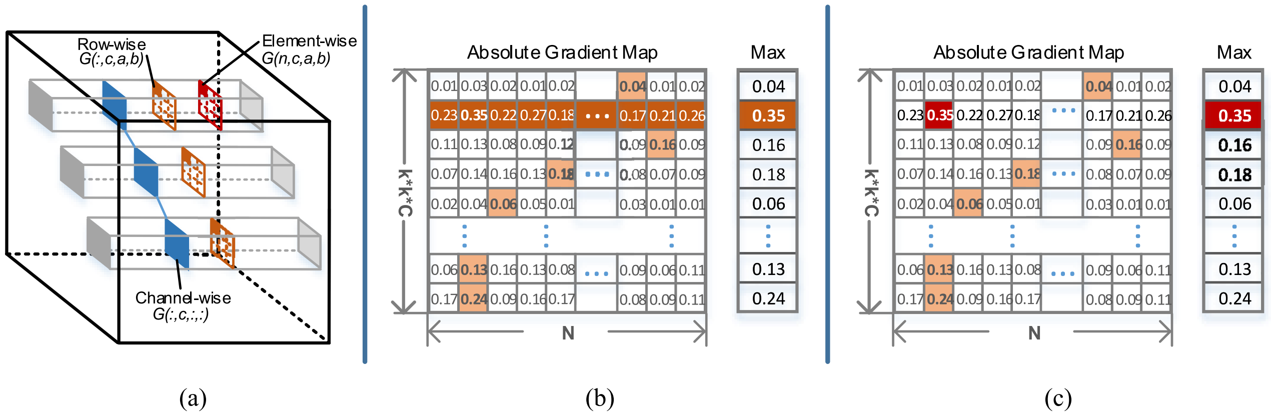


Fig. 4. Proposed SGS. Gradients can be split into multiple groups. (a) Illustrates the channel-wise, row-wise and element-wise GS. (b) and (c) Selection process of row-wise and element-wise sparsification.

the maximum gradient magnitude lies is also calculated along with the index of the row, and we only update the corresponding location in the weight matrix. Consequently, the gradient sparsity will rise to $1 - 1/(k^2C \times N)$.

C. Complexity Analysis

The complexity of row-wise and element-wise SGS mainly concentrate on maximum value selection. With a kernel size of $k^2C \times N$, selecting the gradients with the largest magnitude in all rows will be operated for k^2C times; and selecting the largest one from these gradients will be operated for one time. Thus, the total operation count will be $k^2C \times N + k^2C$. If operating serially, the time complexity will be $O(k^2C \times (N + 1)) \approx O(n)$ $(n = k^2C \times N)$, which is far less than $O(n \log_2 s)$ as conventional GS which picks the top-s from n numbers. And if operating in parallel, the time complexity will be reduced to $O(\log_2 n)$.

VI. Aging-Aware Row Swapping

Although the row-wise SGS ensures the cells in one row to share the same write times, the write distribution is still extremely unbalanced among rows. However, the row-wise operation fits the horizontal stripe pattern of the write distribution shown in Fig. 2. It intensifies the concentration of writes and thus is favorable for row swapping to balance the write-load. Therefore, we propose the ARS approach when processing the training tasks, to dynamically adjust the weight mapping at a small extra overhead.

A. Basic Process

In the training with sparse gradients, if some locations are updated much more often than others, it is reasonable to assume that they will be updated frequently in the following training iterations because their significance on extracting the features has been highlighted. So if we conduct a remapping and swap the rows which are mostly written with the rows which are least written, the write times across whole crossbars will be prospectively more balanced.

Fig. 5 shows the basic process of row swapping. Registers are placed to count the write times of all cells. As the cells in one row share the same write times, only M registers are needed to record the ages in a crossbar of size $M \times N$. If the maximum training iteration number is set to T, the bit-width of a counter register only need to be $\log_2(T)$ at most. Thus, the total memory requirement of write counters will be $M \log_2(T)$. Moreover, an ARS interval (noted as SI) is set to control the frequency of row swapping, and a variable R is set to decide the number of swapped rows in each ARS. Every SI iterations, the most-written R rows and the least-written R rows are picked out, respectively; then swap the largest one with the smallest one, the second largest one with the second smallest one, and so forth. In theory, such in-order swapping can maximize the uniformity, while it is not robust to adversarial attacks because the row mapping relationship determined by the ARS can be easily cracked. We further discuss the security problem in Section VII.

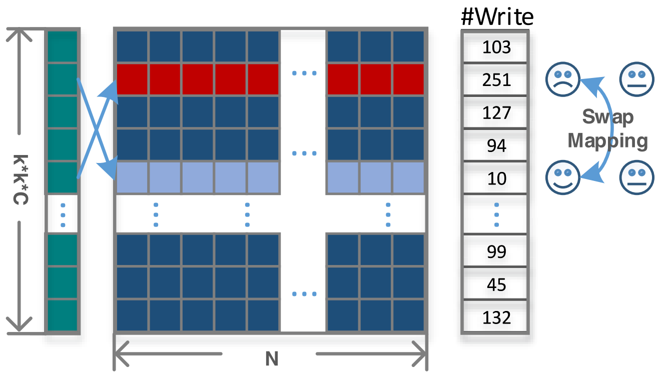


Fig. 5. Basic process of ARS.

B. Trade-Offs

As stated above, the process of ARS is in control of two hyper-parameters, the ARS Interval SI and the number of swapped rows R in each ARS operation, and will introduce extra overhead. First, the swapping of two RRAM crossbar rows needs to read out the original weights on these rows, and then write them back to each other's positions, respectively. In other words, an ARS incurs a read operation and a write operation on each cell in the selected rows. Second, performing the ARS during training requires not only an interruption but also the rescheduling of the input data addressing, since the weight mapping has been changed.

Performing ARS more frequently certainly results in more balanced writes. Since each row swapping needs to rewrite the weights mapped on corresponding rows, it will stop being profitable to continue to decrease the interval SI, when the SI reaches a threshold value. Moreover, the configuration of R also should take into consideration the tradeoff between hardware overhead and write-load balance. In our implementation, we experimentally find a optimal pair of (SI, R), which is presented in Section VIII.

C. Address Mapping

It is noticeable that the ARS algorithm will swap the weight positions continuously throughout the training process, which changes the mapping addresses of the weight parameters. To ensure the computational correctness, the ARS scheme requires additional architectural support for sending the propagated feature maps and gradients to their correct positions. This raises a challenge that there must be a connection router module to connect the NVM crossbars. Xia *et al.* [20] proposed to eliminate the router overhead by reordering the neurons, i.e., if swapping the ith and jth row of the nth layer's weights, the ith and jth column of the $(n-1)$th layer's weights shall be swapped simultaneously. However, two problems arise under the reordering mechanism. First, supporting both row- and column-oriented parallel write will incur significant overhead. Despite multidimensional-access memories [32] have been developed to enable the write along both the horizontal and vertical axes, they require duplication or alteration to the driving circuitry in the interfaces. Without the multidimensional write mechanism, the reordering will consume a lot of cycles. Second, the reordering has a great impact on the flexibility of CONV weights accommodation, because moving one column in the $(n-1)$th CONV layer will reorder $k \times k$ rows in the nth (the next) CONV layer. This will degrade the efficiency of the ARS algorithm because the swapping must be performed at the channel granularity.

In most NVM-based systems, the arrays communicate with each other by utilizing buffers and NoCs instead of directly physical connection. We propose to resolve the addressing problem by introducing address mapping look-up tables (AM-LUTs). The AM-LUTs will be set to record the mapping relationship of the original row number and the swapped row number. The AM-LUTs will be maintained throughout the training process. Before the data being transmitted to the next layer, the actual address (row number) shall be transformed according to the AM-LUTs. Because the ARS algorithm only swaps the mapping of weights inside the same layer. Hence, assuming the row number of a neural layer as R_{all}, the memory occupation for the AM-LUTs will reach $R_{\text{all}} \times \log_2 R_{\text{all}}$ bits.

The desirable characteristics of NVM crossbars for NN training include the dimensional scalability that can always hold whole parameters of the neural layers. However, the NVM crossbars are impossibly scaled up as excepted, owing to the IR-drop, sneak-path, manufacturing technology constraints, etc. Previous work has discussed the splitting mechanisms to enable the mapping of large weight matrix onto the size-limited cross-point NVM arrays. We integrate the same splitting strategy as proposed in [33]. Therefore, when mapping the weights onto multiple crossbars, there may be spare rows that are not utilized. The spare rows will be utilized in the swapping operations to fully disperse the write across the whole arrays. Therefore, the total rows involved in the ARS scheme will be $\lceil k^2 C/M \rceil * M$ in CONV layers and $\lceil N_i/M \rceil * M$ in FC layers, where k represents the kernel size and C represents the input channel count in CONV layers, and N_i represents the input neuron count in FC layers.

VII. Defense Against the Attacks

Although the SGS-ARS framework prospectively brings significant enhancement to the lifetime compared with training with dense gradients, the endurance-limited NVM-based TIME systems are still vulnerable to the adversarial attacks. Thus far, we only consider the normal training workloads, e.g., the training of ResNet or VGG, etc. However, the TIME systems will be threatened by malicious NN training. An adversary who knows about the working mechanism of SGS-ARS framework can easily design an attack that stresses target rows and cause them to quickly reach the endurance limit, thereby making the entire TIME system failed. In this section, we discuss a possible adversarial attack model, and extend the SGS-ARS framework to enhance the availability of TIME systems under such malicious attacks.

A. Problem Formulation

The goal of attack is to repeatedly write the same target physical rows under the SGS-ARS framework. Therefore, the attackers should finish two basic processes: one is to localize the target physical row, and the other is to force the system to write this row. We denote the former process as "localization" and the latter one as "targeting," and we will explain how to implement these two process as following.

Targeting can be implemented by using the same mechanism as introduced in Section III. To locate the target row and update it, we need to maximize the gradients of the weights in this row. According to (3), when setting the x_t^l as nonzeros and the others $(i = 1, 2, \ldots, N, i \neq t)$ as zeros, the gradients can be controlled as $g_{t,:}^l \neq 0, g_{i,:}^l = 0, (i \neq t)$. Under the SGS mechanism, the row t will be updated because the absolute values of the t row gradients with respect to the weights are always the largest ones in the matrix. Therefore, the attacker will succeed to force the system to update the target row.

Localization is required to track the target row after the ARS operations. We assume that the mapping relationship of the algorithmic weight address and the actual physical address is invisible to the user. Without wear-leveling techniques, the attackers only need to specify a row at the beginning and then iterative write on it. However, the ARS technique is applied to make the write traffic uniform, so the mapping relationship of the algorithmic weights and the physical address keeps changing. Therefore, to precisely attack the target row after the swapping, the attacker will *track* the weights that are mapped on the target row. As shown in Fig. 6, the process in each swap interval can be divided into two stages to realize the goal. We first define several notations to illustrate the overall process: the most-written (also the attack target) row R_t, the weights

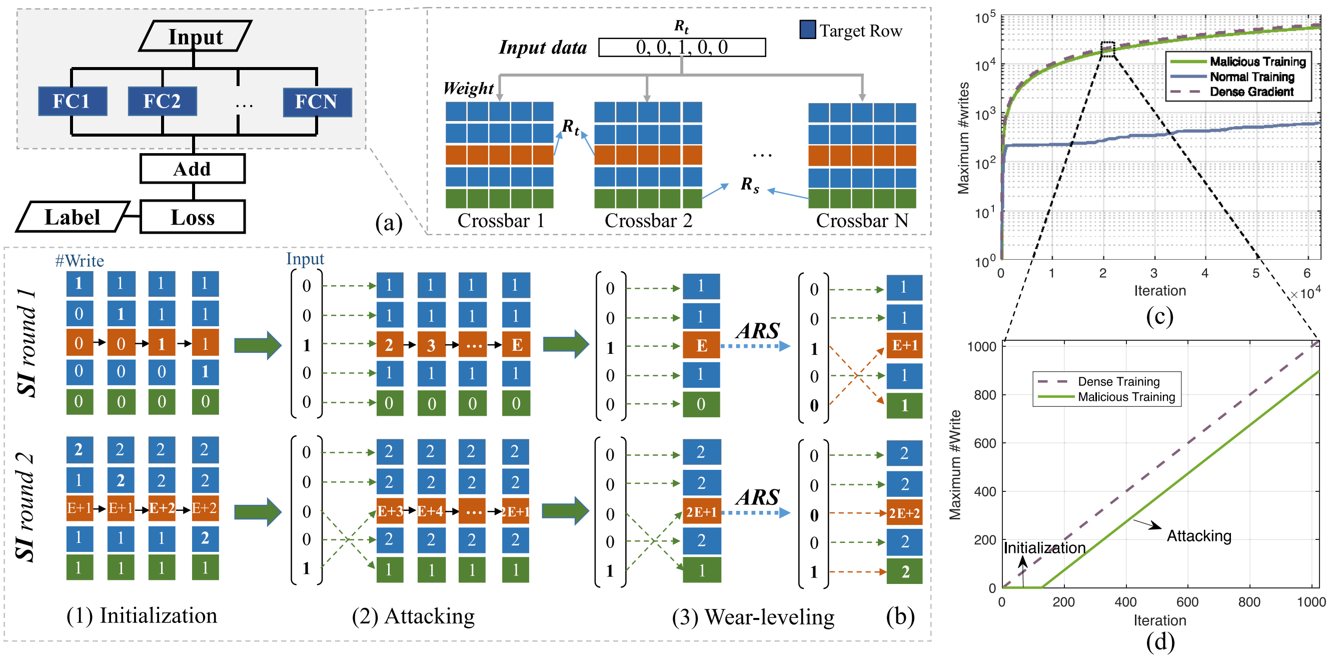


Fig. 6. Example adversarial attacking method, in which the crossbar size is set as $(M, N) = (128, 128)$, and the SI as 1024. (a) Adversarial network architecture can spread the adversarial attacking targets among all the available crossbars by setting the FC layers with M input neurons and N output neurons. (b) Stages of attacking the target row in an ARS round, where $E = SI - M + 2 = 898$. (c) Maximum write times versus the training iteration. (d) Example maximum write times within an ARS round.

W_t which are initially mapped on R_t, the LWR R_s, and the weights W_s which are initially mapped on R_s.

Stage 1 (Initialization): To attack R_t, the attacker must know which weights are currently mapped on it. Therefore, the process first launches an initialization on the write counter. Because the ARS always swaps the most written rows (MWRs) with the LWRs, the attacker only needs to distinguish two rows, one with the most written times, i.e., R_t, and the other with the least written times, i.e., R_s, then the weights W_s will certainly be swapped with W_t when performing the ARS. Therefore, in the initialization phase, the process will first write all the rows except for R_s for 1 time, and never write R_s, so that the weight position can be exactly tracked.

Stage 2 (Attacking): The next stage of the malicious process will launch repeat writing on R_t until the SI arrives. Every time when performing the ARS operation, the weights (W_t) will always migrate to where W_s lies, and similarly, the weights (W_s) will be remapped to where W_t lies. Therefore, in the odd ARS intervals, to force the system to update the physical address R_t, the following weight updating will be simply performed on W_t. And in the even ARS rounds, the weights mapped on R_t will be W_s. Thus, the updating will be performed on W_s, thereby still triggering the write on R_t. Repeatedly, the attackers can always update on the same physical row R_t. Since each swapping operation incurs a write on both the MWRs and the LWRs, the write counter of R_s will increase one. Thus, within every swap interval, the above two stages will be performed.

Hence, such adversarial process can be utilized by an attacker to fast break the system even only with a common user authorization. Kindly noted that the attacks are not only effective under our framework but also can be adjusted to disable the other similar wear-leveling techniques if they are leaked. Through designing the training data and network architecture, the attackers can easily realize targeted weight updates, thereby enabling repeat writing to targeted rows. In this scenario, the time to make a target row fail can be given

$$T_{\text{fail}} = \text{Endurance} \times T_{\text{unit}} \times \frac{SI}{SI - (M-2) + 1} \tag{4}$$

where T_{unit} is defined as the time period of performing an iteration of training. The fraction item indicates that every SI round, there must be $M - 2$ writes to finish the initialization and one additional write will be incurred due to the ARS. By deploying the network architecture on the TIME system as shown in Fig. 6, the attacker is able to spread the malicious process on all the memory crossbars. Besides, because all the FC layers are independent of each other, the malicious processes can be parallel. The process can be fast because the training batch size can be configured as one, thus processing an input can incur an update. Assume $T_{\text{unit}} = 0.0005$ s,[1] and the endurance of NVM cells as 10^7 cycles, the attackers will be able to fail a target row within 1.5 h. Therefore, a defence solution is highly demanded to enhance the reliability. Motivated by this, we extend the framework to enhance the attack difficulty, thereby preventing the TIME systems from being fast broken by the malicious attacks.

B. Protection Solution

To ensure the reliability of the TIME systems under adversarial settings, we extend our SGS-ARS framework to enhance the ability of defending the malicious attacks as mentioned

[1]The time consumption is estimated based on the 42.25× speedup than GPU reported in [7]. We run the attack process on a GPU, which consumes approximately 0.02 second per iteration.

above. To protect the TIME systems from being threatened by the malicious attacks, we need to enhance the difficulty of at least one of the two steps of the malicious process: 1) localization and 2) targeting. For better convergence of the NN training, larger gradients often contribute more to the convergence speed. Therefore, it is always preferred to update the weights with the largest gradients. From the perspective of algorithm, it is more prospective to defend the attacks by obfuscating the mapping relationship, thereby preventing the attackers from accurately locating the target row.

Random-ARS: Randomness in swapping operations can confuse the attacker's judgement of the target physical address. It increases the difficulty of tracking a specific row during training as the mapping physical address of the weights will be unpredictable. Motivated by this, we introduce a randomized ARS scheme (referred to as Random-ARS) to enhance the attack difficulty. Recall that the ARS scheme always swaps the MWRs to the LWRs. We can randomize the swapping operations by randomly constructing a random mapping relationship of the MWRs and LWRs, instead of swapping by following the order of the written times. Theoretically, assuming the swapping row number as R, i.e., changing the mapping of weights in the most-written R rows and the least-written R rows, the attack overhead will increase by R times. Therefore, the expected time to failure of the TIME system under adversarial setting will be extended to $T_{\text{fail}} * R$ as the malicious write will be randomly distributed across the R rows.

Refresh: Although that the Random-ARS can alleviate the threats of being attacked, the attackers still have opportunities to break down the system by tracking a group of target rows (R rows). Our goal is to completely prevent the attackers from tracking the actual physical mapping of the weights. Ideally, the expected fail time should approach $T_{\text{fail}} * M$. Motivated by this, we further extend the framework by dynamically refreshing the mapping relationship periodically to avoid the mapping information leaked.

The protection solution will introduce additional overhead and new security risk from the following two aspects. On the one hand, the Refresh operation shuffles all the weight mapping, thereby obfuscating the addresses to prevent the attackers from exactly tracking the target row. Refresh incurs large energy and time overhead, because the Refresh remaps all rows of weights and the ARS only swaps the selected R rows. Therefore, the frequency RI of applying Refresh should be controlled within a reasonable range. Actually, there is no need to perform the Refresh operations frequently. As our goal is distributing the write stress of the malicious attack to all rows, an appropriate interval of performing Refresh is $R * SI$ iterations. As within this interval, the expected write time on each row is SI, which is consistent with the protection goal. On the other hand, random number generators (RNGs) will be required to generate the randomness for the Random-ARS and Refresh operations. The randomness should be invisible and unpredictable because the essence of our approaches is to prevent the mapping relationship from being tracked. Previous work has intensively discussed about the randomness quality, efficiency, and security of RNGs from the generation methods [34], [35] and hardware [36], [37]. Meanwhile, the countermeasures against possible information leakage attacks

TABLE I
CONFIGURATIONS OF THE EXPERIMENTAL PARAMETERS

Parameter	Value
Endurance Cycle	10^7
Crossbar Size	256x256
Swap Interval (SI)	1024
Swapped #Row/SI (R)	32
Refresh Interval (RI)	32768

(e.g., side-channel attacks [38]) can be utilized to protect the generated randomness. Therefore, this article assumes that the attackers cannot extract or predict the randomness and the AM-LUT information.

VIII. EVALUATION

In this section, we evaluate the effectiveness of our SGS-ARS framework and the ability of defending against malicious attacks. We evaluate the framework from several key metrics, including the accuracy influence, the tradeoffs, the write uniformity, the security under adversarial attacks, and the performance overhead for deploying the framework.

A. Experiment Setup

Benchmark: We construct the training of VGG-16 [1], ResNet-20 [15], and MobileNet-v1 [39] network architectures with the CIFAR dataset for the performance evaluation, and ResNet-50 trained on the ImageNet dataset is designed for validating the large-scale training performance. We also contrast the performance with four existing methods. The "baseline" method refers to training without GS; the "conventional GS" method refers to the conventional GS; the "FT-Train" method refers to the Fault-Tolerant Training method proposed in [20]; and the "Start-Gap" method proposed in [27].

Evaluation Metric: Our experiments demonstrate the effectiveness of the framework from four aspects. First, the performance of trained NN models should be guaranteed, because it is necessary for an NN training system to provide a satisfying accuracy for the users. Second, recalling that the main motivation is to enhance the lifetime of TIME systems, we also evaluate the lifetime extension brought by the framework. We define the "lifetime" as the fastest time that one row in the NVM crossbars reach the endurance limit, assuming there is no spare rows or crossbars and no fault-tolerant module which enables normal functionality even with a reasonable fault ratio, because these methods are orthogonal to the proposed framework and can be combined. Besides, the variation on the endurance characteristic is not considered. All the NVM cells are regarded with a same endurance. Third, we also discuss the security under adversarial settings, and evaluate the effectiveness of the extension to defend the malicious attacks. Fourth, we evaluate the additional overhead introduced by the architectural support of SGA-ARS framework.

Configurations: In default, the configurations of the hardware and resources are listed as Table I. We set the swap interval SI and swapped row number R as (1024, 32) to tradeoff the overhead based on the observation in the experiments in Section VIII-C. Besides, the other parameters are decided based on the commonly used hardware settings and actual limitations.

TABLE II
EXPERIMENTAL RESULTS

Model	Dataset	Classification Accuracy				Sparsity of SGS (Ratio, N_{update})	#Max Writes		Lifetime Extension	
		SGS		Baseline						
		Top-1	Top-5	Top-1	Top-5		SGS-ARS	Baseline	SGS-ARS	FT-Train [20]
ResNet-20	Cifar10	91.5% (-0.6%)	-	92.1%	-	99.7%, 1	371	64124	173×	N/R[2]
	Cifar100	67.8% (+0.4%)	90.9% (+0.1%)	67.4%	90.8%	99.7%, 1	387	64124	166×	N/R
MobileNet	Cifar10	91.2% (-0.6)	-	91.8%	-	99.9%, 1	609	64124	105×	N/R
	Cifar100	66.7% (-0.9%)	89.3% (-0.1%)	67.6%	89.4%	99.9%, 1	603	64124	106×	N/R
VGG-16	Cifar10	92.6% (-1.4%)	-	94.0%	-	99.7%, 4	466	78200	168×	15×
	Cifar100	71.1% (-1.6%)	89.5% (-1.1%)	72.7%	90.6%	99.1%, 32[1]	1503	78200	52×	N/R
ResNet-50	ImageNet	75.1% (-1.0%)	92.4% (-0.5%)	76.1%	92.9%	99.8%, 1	1264	450450	356×	N/R

[1] The update numbers of the layers in VGG-16 are configured as $\min(R_{all} * 1\%, 32)$ when trained on CIFAR-100 dataset.
[2] N/R represents the data is not reported in [20]

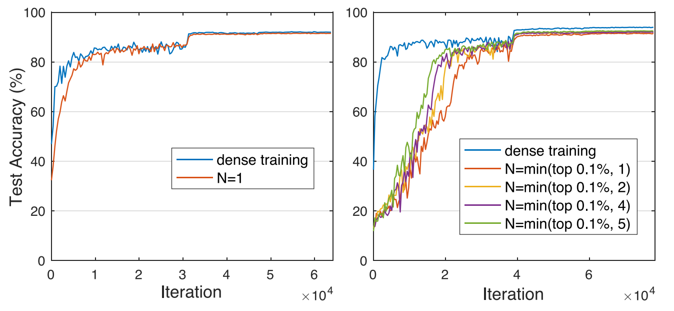


Fig. 7. Convergence curves of training NN models (left: ResNet-20; right: VGG-16) on the CIFAR-10 dataset with different gradient sparsity. "Dense training" represents the conventional training with dense gradients, and the others represent training with SGS algorithm, in which N represent the number of updated rows N_{update} per layer in each iteration.

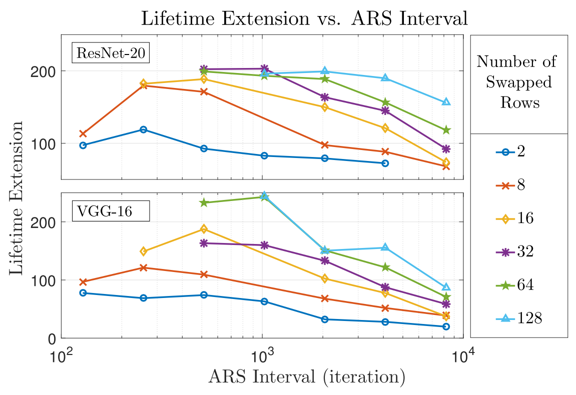


Fig. 8. Inverted U-shaped curve of lifetime extension versus ARS interval. Under different numbers of ARS swapped row, the lifetime extensions first rise, then saturate, and eventually decline, as the ARS interval increases.

B. Accuracy Influence

Accuracy: We demonstrate the performance of SGS-ARS methodology by evaluating the classification accuracy of the trained models. We select several classical network architectures and datasets as examples to show the accuracy. Table II shows that there is slight loss of top-1 accuracy on ResNet-20 and MobileNet-v1 with $N_{\text{update}} = 1$. However, VGG-16 requires less sparse gradients to converge better and achieve comparable accuracy. A similar phenomenon has also been observed in [40] which reasons that the VGGs place the critical weights across the whole layers. Thus, the weights updating needs to be applied over a wider range of weights. Besides, when it comes to the large scale dataset and deeper NN, the experiment of ResNet-50 only displays 1.0% loss of top-1 accuracy. Overall, SGS-ARS is able to acquire very close performance as the baseline.

Convergence Effectiveness: We contrast the convergence of the NN training with the SGS algorithm under different sparsity. As shown in Fig. 7, the convergence speed of training ResNet-20 with sparse gradients almost coincides with the training with dense gradients. As for VGG-16, two conclusions can be figured out from the results. First, a lower sparsity (i.e., updating more rows per iteration) demonstrates a faster convergence speed at the very beginning. This is an reasonable conclusion because a randomly initialized weights usually need to be tuned from multiple directions to adapt better for

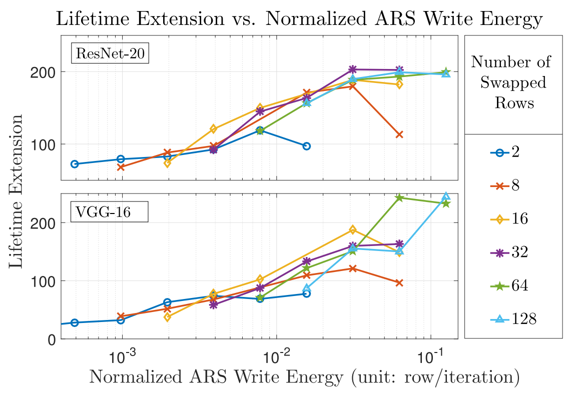


Fig. 9. S-shaped curve of lifetime extension versus normalized ARS write energy. Normalized ARS write energy is the average energy consumption over iterations caused by writing swapped rows, comparing to the energy consumed by writing one row. Under different numbers of ARS swapped row, the lifetime extensions first rise, and finally saturate, as the normalized ARS write energy increases.

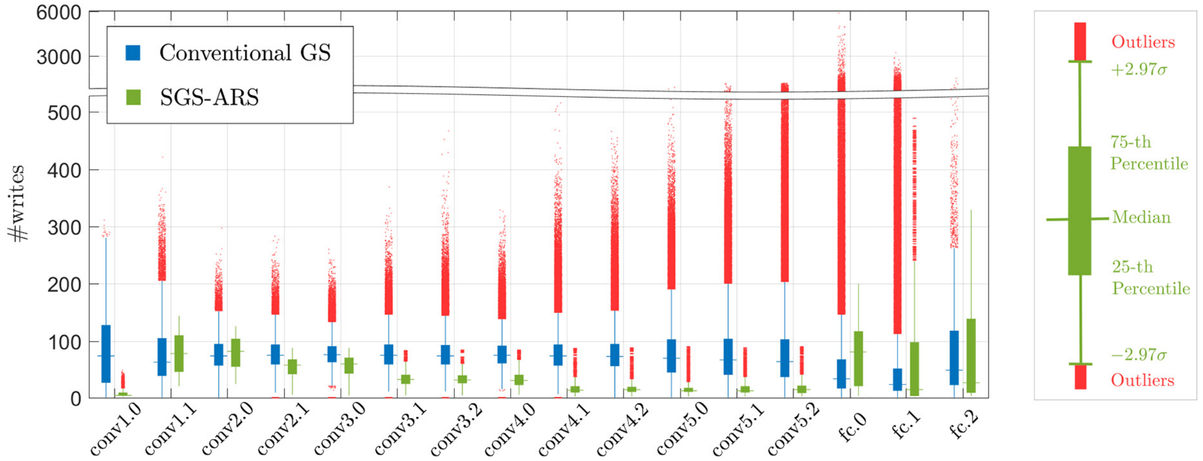


Fig. 10. Box plot of write times of all layers when training VGG16 on CIFAR-10 dataset. The color-filled box indicates the range of the centered half of statistics, and the whiskers show the 99.3% coverage if normally distributed. The outliers in red describe the degree of the unbalanced distribution.

the specialized dataset. Second, although the NN models converge slightly slower when applying a smaller N_{update}, they can catch up with the pace of training with denser gradients and eventually obtain comparable accuracy.

C. Lifetime Evaluation

Trade-Offs: We explore the tradeoffs by performing experiments with different configuration parameter sets (SI, R). The search space of ARS interval SI ranges from 10^2 to 10^4, and the number of swapped rows R in each ARS ranges from 2 to 128. Figs. 8 and 9, respectively, show the curves of lifetime extension versus ARS interval and normalized ARS write power under different configurations. As is shown in Fig. 9, the curves of lifetime extension versus SI are inverted U-shaped. To maximize the expected lifetime extension, the sets of configuration within the flat stage of the curves are preferred. However, while maintaining the lifetime extension, the larger SI is and the smaller R is, the less overhead will be introduced by ARS. Meanwhile, Fig. 9 shows S-shaped curves of lifetime extension versus normalized ARS write energy. The optimal set should also be within the flat stage of the curves but with less energy consumption. Therefore, we choose a relatively optimal set of (SI, R) as (1024, 32). This is a near-optimal generic configuration, although the performance may slightly vary in different models. All the following experiments utilize this set of configuration.

Write Distribution: The effectiveness of the proposed SGS-ARS approach in mitigating unbalanced writes is evaluated from two aspects: 1) the statistical distribution of write times and 2) the trend of maximum write time of layers during training process, compared to conventional GS and Baseline.

1) *Statistical Distribution:* As shown in Fig. 10, the box plot is adopted here to statistically analyze the distribution of the write times of all cells in CONV and FC layers of VGG-16 after 78 200 iterations training. Common statistical information is illustrated in the plot: first quartile, median, third quartile, ±2.7 variance, and outliers. For CONV layers, both the median and variance of the write times are much smaller with SGS-AGS. For FC layers, the median of write times is smaller, but third quartile is slightly larger, which indicates the distribution of write times becomes more left skewed to fewer writes. The range of outliers of both CONV and FC layers trained with SGS-AGS dramatically declines from hundreds and thousands to dozens, and even vanishes in some layers. This demonstrates that SGS-AGS not only significantly reduces the write times but also effectively resolves the unbalanced writes issue.

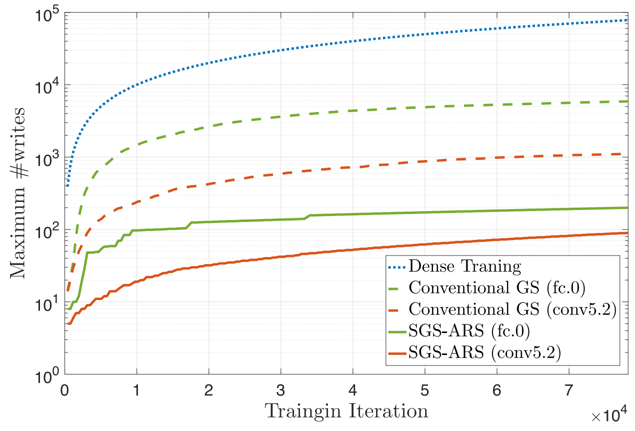


Fig. 11. Maximum write times of FC (fc.0) and CONV (conv5.2) layer when training VGG16 on Cifar10 dataset. Conventional GS reduce the write times by tenfold, while, the SGS-ARS framework achieves a reduction of writes by more than two orders of magnitude.

2) *Maximum Write Times:* Fig. 11 shows the maximum cell write times at each of the 78 200 iterations. The sample layers are, respectively, CONV (conv5.2) layer and FC (fc.0) layer of VGG-16. The write time of Baseline increases linearly, because without GS, all cells will be written in every iteration. The maximum write time of conventional GS increases much slower than the baseline, but still much faster than SGS. At the last iteration, the maximum write time of the FC layer by conventional GS is more than 29 times larger than by SGS. Overall, SGS reduces the maximum write time by around 160× and 12× compared to the baseline and conventional GS, respectively.

TABLE III
SIMULATED TIME AND MEMORY REQUIRED FOR TRAINING VARIOUS NNS ON THE NVM-BASED TIME SYSTEMS

Network[1]	Iteration	Time				Memory Occupation			
		FP+BP[2]	Update[3]	ARS[3]	Refresh[3]	#Weight	#Gradient	ARS-counter	AM-LUT
ResNet-20	64124	30.2 s	65.3 ms	6.43 ms	1.48 ms	0.27 M	0.54 M	18.4 KB	10.6 KB
VGG-16[4]	78200	64.5 s	155.2 ms	6.27 ms	6.2 ms	15.2 M	30.4 M	78.5 KB	56.6 KB
MobileNet-v1[5]	64124	33.4 s	91.4 ms	8.99 ms	8.4 ms	4.2 M	8.4 M	102 KB	79.6 KB
ResNet-50	450450	11941 s	1146 ms	112.9 ms	57.9 ms	25.5 M	51 M	126 KB	74 KB

[1] The first three networks are evaluated on the CIFAR-10 dataset, and the ResNet-50 is evaluated on the ImageNet dataset.
[2] is estimated based the performance reported in [7], with an average of 42.45x speedup than GPU. Therefore, we run the training tasks on a GPU (Nvidia GeForce RTX 2080), and divide the running time by 42.45x to obtain the estimated running time of TIME systems.
[3] We assume the write on a same row can be parallel, and the updates on different rows and different neural layers are serially executed. The read and write latency are set as 29.31 ns and 50.88 ns based on [7].
[4] We make slight modification on the VGG-16 network, in which the FC layers are compressed to 512x512, 512x512, and 512x10.
[5] The depth-wise convolution separates the channels for calculation. The row-wise SGS is equivalent to element-wise SGS in depth-wise CONV.

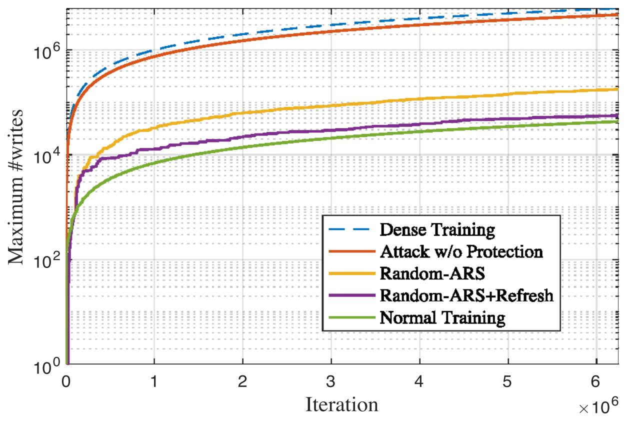


Fig. 12. Maximum write times of the training under different scenarios. Random-ARS can greatly alleviate the attack stress by 32x. Further, Refresh can continue reducing the growing speed of max write time by around 3x. Overall, 84x lifetime can be achieved under adversarial settings.

TABLE IV
ABLATION STUDY AND COMPARISON OF DIFFERENT WEAR-LEVELING METHODS WITH THE TRAINING OF RESNET-20 ON CIFAR-10

Method[1]	Time (ms)	Aging Counter	Addr. Memory	#Max Write	Vulnerable[2]
None	-	-	-	64124	Y
S	-	-	-	3306	Y
S+A	6.43	18.4KB	10.6KB	371	Y
S+RA	6.43	18.4KB	10.6KB	380	y
S+RA+Re	7.91	18.4KB	10.6KB	397	N
S+SG[3] [27]	6.43	0B	44B	2528	Y

[1] We denote the methods as: S for SGS, A for ARS, RA for Random-ARS, Re for Refresh, and SG for Start-Gap method [27].
[2] "Y" represents the method is vulnerable to malicious training; "y" represents the method can alleviate the threat of being attacked; and "N" represents the method can defend the attacks.
[3] To fairly compare, we increase the Start-Gap moving frequency to make the swapping times consistent with the ARS.

Lifetime Extension: Table II also shows the experimental results of lifetime extension under different models. When TIME is programmed for VGG-16 trained on CIFAR, at least 130× longer lifetime will be achieved, and 166× for ResNet-20. Compared to FT-Train, we also get around 10× extension on training of VGG-16. Moreover, on the larger model ResNet-50, 356× lifetime extension is achieved, which is in accordance with our expectation, since the sparsity of row-wise SGS will increase remarkably as the row number of weight matrix increases in the larger NNs.

D. Security Evaluation

We evaluate the security and reliability under adversarial setting. As shown in Fig. 12, without protection methods, the max write time will linear increase with the training iteration, approximately equivalent to training with dense gradients. While the Random-ARS can reduce the rising speed of maximum write times by around 32x, the effect of defense is still unsatisfactory, because it only spreads the pressure of one row into a group of rows (R rows). The Refresh technique can expand the attacked area to the entire array. The max write time grows much slower than that without protection, and closely to the normal training workloads, e.g., training ResNet-20 on CIFAR-10 dataset.

E. Performance Overhead

Memory Occupation: Both the SGS and ARS algorithms require additional memory space to record necessary historical information. As shown in Table III, the gradients take twice memory space than weights because under the GS, a duplication of the gradients is required to accumulate the preserved gradients which are not utilized to update the weights in present iteration. Kindly noted that the *momentum* mechanism also requires a copy of historical gradients. Thus, it will not introduce additional memory occupation for the GS. Moreover, compared with the weights, the ARS-counter and AM-LUT take only a small portion of memory. In the meantime, the system can also provide the memory space for these intermediate data by using NVMs, because the data are modified infrequently throughout the training.

Time Consumption: We show the time consumption of the main parts throughout the training process, without considering the possible latency introduced by scheduling. The SGS significantly reduces the update overhead as per iteration only needs to write a row of the weights in every layer. Moreover, since the ARS and Refresh operations are infrequently executed, they incur less than 1% of the overall time consumption, which is negligible in the training process.

Comparison: We make ablation study to show the overhead and performance differences in Table IV. The protection strategies (Random-ARS and Refresh) only slightly impact the write balance effectiveness. Moreover, we compare the performance with the Start-Gap wear-leveling method [27]. As mentioned before, Start-Gap only records the Start and Gap rows, thus saving significant memory overhead for buffering the ages and the address mapping tables. However, the lifetime extension is unsatisfactory because the swapping is aging-unaware, and can only move a row to its neighbor rows while the weight updating is spatial-concentrated.

IX. Conclusion

In this article, we propose an effective and efficient framework to improve lifetime and security of TIME with SGS and ARS, reducing the overall write times and ensuring the write balance throughout NVM crossbars simultaneously. Moreover, Random-ARS and Refresh techniques are proposed to thwart the malicious attacks. The experimental results show the proposed methods extend the lifetime of TIME for approximately two orders of magnitude with a negligible overhead, while maintaining almost the same NN performance. Under adversarial attacking, the proposed framework can still enhance the lifetime by 84×. In future work, we plan to further evaluate various potential security risks and attack methods (e.g., side-channel attacks), and design robust solutions to enhance the security of TIME systems.

References

[1] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014. [Online]. Available: arXiv:1409.1556.

[2] W. Liu *et al.*, "SSD: Single shot multibox detector," in *Proc. Eur. Conf. Comput. Vis.*, 2015, pp. 21–37.

[3] A. Karpathy and L. Fei-Fei, "Deep visual-semantic alignments for generating image descriptions," in *Proc. Comput. Vis. Pattern Recognit.*, 2015, pp. 3128–3137.

[4] S. C. H. Hoi, D. Sahoo, J. Lu, and P. Zhao, "Online learning: A comprehensive survey," 2018. [Online]. Available: arXiv:1802.02871.

[5] M. M. Waldrop, "The chips are down for Moore's law," *Nature*, vol. 530, no. 7589, p. 144, 2016.

[6] M. Cheng *et al.*, "Time: A training-in-memory architecture for memristor-based deep neural networks," in *Proc. 54th Annu. Design Autom. Conf.*, 2017, p. 26.

[7] L. Song, X. Qian, H. Li, and Y. Chen, "Pipelayer: A pipelined reram-based accelerator for deep learning," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit.*, 2017, pp. 541–552.

[8] S. Ambrogio *et al.*, "Equivalent-accuracy accelerated neural-network training using analogue memory," *Nature*, vol. 558, no. 7708, pp. 60–67, 2018.

[9] K. Beckmann, J. Holt, H. Manem, J. V. Nostrand, and N. C. Cady, "Nanoscale hafnium oxide rram devices exhibit pulse dependent behavior and multi-level resistance capability," *MRS Adv.*, vol. 1, no. 49, pp. 1–6, 2016.

[10] C. H. Cheng, A. Chin, and F. S. Yeh, "Novel ultra-low power RRAM with good endurance and retention," in *Proc. Symp. VLSI Technol.*, 2010, pp. 85–86.

[11] R. F. Freitas and W. W. Wilcke, "Storage-class memory: The next storage system technology," *IBM J. Res. Develop.*, vol. 52, no. 4.5, pp. 439–447, 2008.

[12] C.-W. Hsu *et al.*, "Self-rectifying bipolar TaO_x/TiO_2 RRAM with superior endurance over 10^{12} cycles for 3D high-density storage-class memory," in *Proc. Symp. VLSI Technol.*, 2013, pp. T166–T167.

[13] A. Grossi *et al.*, "Fundamental variability limits of filament-based RRAM," in *Proc. IEEE Int. Electron Devices Meeting*, 2016, pp. 4–7.

[14] M. Zhao *et al.*, "Characterizing endurance degradation of incremental switching in analog rram for neuromorphic systems," in *Proc. IEEE Int. Electron Devices Meeting (IEDM)*, 2018, pp. 1–4.

[15] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770–778.

[16] O. Russakovsky *et al.*, "ImageNet large scale visual recognition challenge," *Int. J. Comput. Vis.*, vol. 115, no. 3, pp. 211–252, 2015.

[17] C. Szegedy *et al.*, "Going deeper with convolutions," in *Proc. Comput. Vis. Pattern Recognit.*, 2015, pp. 1–9.

[18] A. F. Aji and K. Heafield, "Sparse communication for distributed gradient descent," in *Proc. Empir. Methods Nat. Lang. Process. (EMNLP)*, 2017, pp. 440–445.

[19] Y. Lin, S. Han, H. Mao, Y. Wang, and W. J. Dally, "Deep gradient compression: Reducing the communication bandwidth for distributed training," 2017. [Online]. Available: arXiv:1712.01887.

[20] L. Xia *et al.*, "Fault-tolerant training with on-line fault detection for RRAM-based neural computing systems," in *Proc. Design Autom. Conf.*, 2017, pp. 1–6.

[21] K. Hsieh, A. Harlap, N. Vijaykumar, D. Konomis, G. R. Ganger, and P. B. Gibbons, "Gaia: Geo-distributed machine learning approaching LAN speeds," in *Proc. 14th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, Boston, MA, USA, 2017, pp. 629–647.

[22] L. Xia *et al.*, "Stuck-at fault tolerance in RRAM computing systems," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 8, no. 1, pp. 102–115, Mar. 2018.

[23] S. Cho and H. Lee, "Flip-n-write: A simple deterministic technique to improve pram write performance, energy and endurance," in *Proc. 42th Annu. IEEE/ACM Int. Symp. Microarchit.*, 2009, pp. 347–357.

[24] A. Ben-Aroya and S. Toledo, "Competitive analysis of flash memory algorithms," *ACM Trans. Alg.*, vol. 7, no. 2, p. 23, 2011.

[25] E. Gal and S. Toledo, "Algorithms and data structures for flash memories," *ACM Comput. Surveys*, vol. 37, no. 2, pp. 138–163, 2005.

[26] P. Zhou, B. Zhao, J. Yang, and Y. Zhang, "A durable and energy efficient main memory using phase change memory technology," *ACM SIGARCH Comput. Archit. News*, vol. 37, no. 3, pp. 14–23, 2009.

[27] 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 *Proc. IEEE/ACM Int. Symp. Microarchit.*, 2009, pp. 14–23.

[28] N. H. Seong, D. H. Woo, and H.-H. S. Lee, "Security refresh: Prevent malicious wear-out and increase durability for phase-change memory with dynamically randomized address mapping," *ACM SIGARCH Comput. Archit. News*, vol. 38, no. 3, pp. 383–394, 2010.

[29] M. K. Qureshi, A. Seznec, L. A. Lastras, and M. M. Franceschini, "Practical and secure PCM systems by online detection of malicious write streams," in *Proc. IEEE 17th Int. Symp. High Perform. Comput. Archit.*, 2011, pp. 478–489.

[30] G. W. Burr *et al.*, "Large-scale neural networks implemented with non-volatile memory as the synaptic weight element: Comparative performance analysis (accuracy, speed, and power)," in *Proc. IEEE Int. Electron Devices Meeting*, 2015, pp. 1–4.

[31] L. Gao *et al.*, "Fully parallel write/read in resistive synaptic array for accelerating on-chip learning," *Nanotechnology*, vol. 26, no. 45, 2015, Art. no. 455204.

[32] S. George *et al.*, "MDACache: Caching for multi-dimensional-access memories," in *Proc. 51st Annu. IEEE/ACM Int. Symp. Microarchit. (MICRO)*, 2018, pp. 841–854.

[33] T. Tang, L. Xia, B. Li, Y. Wang, and H. Yang, "Binary convolutional neural network on RRAM," in *Proc. 22nd Asia South Pacific Design Autom. Conf. (ASP-DAC)*, 2017, pp. 782–787.

[34] F. James, "A review of pseudorandom number generators," *Comput. Phys. Commun.*, vol. 60, no. 3, pp. 329–344, 1990.

[35] X. Chen *et al.*, "Modeling random telegraph noise as a randomness source and its application in true random number generation," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 35, no. 9, pp. 1435–1448, Sep. 2016.

[36] H. Jiang *et al.*, "A novel true random number generator based on a stochastic diffusive memristor," *Nat. Commun.*, vol. 8, no. 1, p. 882, 2017.

[37] A. P. Johnson, R. S. Chakraborty, and D. Mukhopadyay, "An improved DCM-based tunable true random number generator for xilinx FPGA," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 64, no. 4, pp. 452–456, Apr. 2017.

[38] J. Fan, X. Guo, E. D. Mulder, P. Schaumont, B. Preneel, and I. Verbauwhede, "State-of-the-art of secure ECC implementations: A survey on known side-channel attacks and countermeasures," in *Proc. IEEE Int. Symp. Hardw. Orient. Security Trust (HOST)*, 2010, pp. 76–87.

[39] A. G. Howard *et al.*, "MobileNets: Efficient convolutional neural networks for mobile vision applications," 2017, [Online]. Available: arXiv:1704.04861.

[40] C. Zhang, S. Bengio, and Y. Singer, "Are all layers created equal?" 2019. [Online]. Available: arXiv:1902.01996.

Yi Cai received the B.S. degree in electronic engineering from Tsinghua University, Beijing, China, in 2017, where he is currently pursuing the Ph.D. degree with the Department of Electronic Engineering.

His research mainly focuses on deep learning acceleration and emerging nonvolatile memory technology.

Yujun Lin received the B.S. degree in electronic engineering from Tsinghua University, Beijing, China, in 2018. He is currently pursuing the Ph.D. degree with the EECS Department, Massachusetts Institute of Technology, Cambridge, MA, USA.

His research mainly focuses on efficient deep learning acceleration and machine learning-assisted hardware optimization.

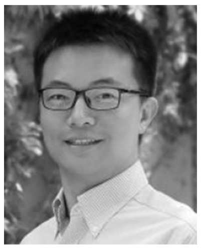
Lixue Xia (Student Member, IEEE) received the B.S. and Ph.D. degrees in electronic engineering from Tsinghua University, Beijing, China, in 2013 and 2018, respectively.

His research mainly focuses on energy efficient hardware computing system design and neuromorphic computing system based on emerging nonvolatile device.

Xiaoming Chen (Member, IEEE) received the B.S. and Ph.D. degrees in electronic engineering from Tsinghua University, Beijing, China, in 2009 and 2014, respectively.

He is currently an Associate Professor with the Institute of Computing Technology, Chinese Academy of Sciences, Beijing. His current research interests include electronic design automation and computer architecture design for processing-in-memory systems.

Dr. Chen was a recipient of the 2015 European Design and Automation Association Outstanding Dissertation Award and the 2018 DAMO Academy Young Fellow Award. He served on the Organization Committee of the Asia and South Pacific Design Automation Conference (ASP-DAC) 2020, the Technical Program Committees of Design Automation Conference 2020, the International Conference on Computer Aided Design 2019, ASP-DAC 2019, the International Conference on VLSI Design 2019 and 2020, the Asian Hardware Oriented Security and Trust Symposium 2018 and 2019, and the IEEE Computer Society Annual Symposium on VLSI 2018 and 2019.



Song Han received the B.S. degree from Tsinghua University, Beijing, China, in 2012, and the Ph.D. degree from Stanford University, Stanford, CA, USA, in 2017.

He is currently an Assistant Professor with the EECS Department, Massachusetts Institute of Technology, Cambridge, MA, USA. He proposed deep compression and efficient inference engine that impacted the industry. His research focuses on efficient deep learning computing.

Dr. Han has received the Best Paper Award in ICLR16 and FPGA17. He served on the Technical Program Committees of 24th IEEE International Symposium on High-Performance Computer Architecture and 38th International Conference on Computer Aided Design. He as an Area Chair of 7th International Conference on Learning Representations.

Yu Wang (Senior Member, IEEE) received the B.S. and Ph.D. (Hons.) degrees from Tsinghua University, Beijing, China, in 2002 and 2007, respectively.

He is currently a Tenured Professor with the Department of Electronic Engineering, Tsinghua University. He has authored and coauthored more than 200 papers in refereed journals and conferences. His research interests include brain inspired computing, application specific hardware computing, parallel circuit analysis, and power/reliability aware system design methodology.

Dr. Wang has received Best Paper Award in ASPDAC 2019, FPGA 2017, NVMSA 2017, and ISVLSI 2012, and Best Poster Award in HEART 2012 with Nine Best Paper Nominations (DATE18, DAC17, ASPDAC16, ASPDAC14, ASPDAC12, 2 in ASPDAC10, ISLPED09, and CODES09). He was a recipient of the DAC under 40 Innovator Award in 2018 and the IBM X10 Faculty Award in 2010. He currently serves as the Co-Editor-in-Chief for the ACM SIGDA E-Newsletter, an Associate Editor for 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 the Special Issue Editor for *Microelectronics Journal*. He served as a TPC Chair for ICFPT 2019 and 2011, and ISVLSI2018, the Finance Chair of ISLPED 2012–2016, the 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. He is currently with ACM Distinguished Speaker Program.

Huazhong Yang (Fellow, IEEE) received B.S. degree in microelectronics, and the M.S. and Ph.D. degrees in electronic engineering from Tsinghua University, Beijing, China, in 1989, 1993, and 1998, respectively.

In 1993, he joined the Department of Electronic Engineering, Tsinghua University, where he has been a Professor since 1998. He has been in charge of several projects, including projects sponsored by the National Science and Technology Major Project, 863 Program, NSFC, and several international research projects. He has authored and coauthored over 500 technical papers, seven books, and over 180 granted Chinese patents. His current research interests include wireless sensor networks, data converters, energy-harvesting circuits, nonvolatile processors, and brain inspired computing.

Dr. Yang was awarded the Distinguished Young Researcher by NSFC in 2000, the Cheung Kong Scholar by the Chinese Ministry of Education (CME) in 2012, the Science and Technology Award First Prize by China Highway and Transportation Society in 2016, and the Technological Invention Award First Prize by CME in 2019. He has also served as the Chair of Northern China ACM SIGDA Chapter Science 2014, the General Co-Chair of ASPDAC20, the Navigating Committee Member of AsianHOST18, and a TPC Member of ASP-DAC05, APCCAS06, ICCCAS07, ASQED09, and ICGCS10.