ESSENCE: Exploiting Structured Stochastic Gradient Pruning for Endurance-aware ReRAM-based In-Memory Training Systems

Xiaoxuan Yang, *Student Member, IEEE*, Huanrui Yang, *Student Member, IEEE*, Janardhan Rao Doppa, *Senior Member, IEEE*, Partha Pratim Pande, *Fellow, IEEE*, Krishnendu Chakrabarty, *Fellow, IEEE*, and Hai (Helen) Li, *Fellow, IEEE*.

***Abstract*—Processing-in-memory (PIM) enables energy-efficient deployment of convolutional neural networks (CNNs) from edge to cloud. Resistive random-access memory (ReRAM) is one of the most commonly used technologies for PIM architectures. One of the primary limitations of ReRAM-based PIM in neural network training arises from the limited write endurance due to the frequent weight updates. To make ReRAM-based architectures viable for CNN training, the write endurance issue needs to be addressed. This work aims to reduce the number of weight reprogrammings without compromising the final model accuracy. We propose the *ESSENCE* framework with an endurance-aware structured stochastic gradient pruning method, which dynamically adjusts the probability of gradient update based on the current update counts. Experimental results with multiple CNNs and datasets demonstrate that the proposed method can extend ReRAM's life time for training. For instance, with the ResNet20 network and CIFAR-10 dataset, *ESSENCE* can save the mean update counts of up to $10.29\times$ compared to the SGD method and effectively reduce the maximum update counts compared with No Endurance method. Furthermore, aggressive tuning method based on *ESSENCE* can boost the mean update count savings by up to $14.41\times$.**

***Index Terms*—Endurance, resistive random-access memory (ReRAM), processing-in-memory (PIM), structured gradient pruning.**

I. Introduction

RESISTIVE random-access memory (ReRAM)-based crossbars can perform efficient vector-matrix multiplications (VMMs), which can be utilized for accelerating convolutional neural networks (CNNs) with low energy cost [1], [2]. Hence, ReRAM-based architectures are suitable platforms for accelerating both CNN training and inference and are more energy- and area- efficient compared to their GPU counterpart [3]. In addition, they do not require expensive off-chip memory access due to their "in-memory" nature of computation. One advantage of inferencing with ReRAM-based PIM architectures is that the weights do not change, and VMM will not update the weight values while utilizing the weight stored in the crossbar [4]. However, CNN training involves multiple weight updates. Only a limited amount of prior research has investigated the technical challenges that arise during the weight update process. Repeated reprogramming of the ReRAM devices during the weight update stage will pose significant hurdles to the ReRAM system's reliability.

The endurance problem limits the number of allowable reprogrammings and remains an open challenge for the ReRAM-based PIM systems [5]. As the periodic and accurate gradient update in each iteration is necessary for CNN training, repeated rewrites of the weights are inevitable. Therefore, utilizing a ReRAM-based PIM system for CNN training requires addressing the problem of periodic weight reprogramming.

In this work, we aim to leverage gradient pruning to reduce the number of gradient updates and thus decrease the counts of weight reprogramming for ReRAM-based PIM systems. Gradient pruning based on the magnitude of gradient value is initially introduced to reduce the communication load for distributed learning [6], [7]. ReRAM-based PIM architecture like PipeLayer [3] implements dense gradient update and thus reprograms the entire weight crossbar in each training iteration. Long Time Live [8] employs gradient pruning and only updates sparse gradients. However, this approach incurs the unavoidable overhead of intermediate gradient storage.

We propose the endurance-aware stochastic gradient pruning strategy, which adjust the update probability based on the previous writing frequency. In our approach, we can reduce the maximum number of reprogramming for the weight matrix compared with No Endurance approach. Furthermore, we propose an endurance-aware structured gradient pruning method to avoid writing to all the cells in selected rows or columns simultaneously. Our approach can decrease the number of weight updates and balance the write accesses. Our design utilizes ReRAM-based counters to keep track of the write accesses in each ReRAM row or column and structurally adjusts the write probability according to the current number of reprogramming recorded in the counter. In summary, we propose the ESSENCE framework for neural network training, which exploits structured stochastic gradient pruning for endurance-aware ReRAM-based PIM systems.

The key contributions of this work are as follows:

- We present *ESSENCE*, an endurance-aware ReRAM-based in-memory training system. *ESSENCE* utilizes

This work is supported in part by the US National Science Foundation (NSF) under grants CNS-1955196, CCF-1725456, OAC-1910213, and CNS-1955353. *(Corresponding author: Xiaoxuan Yang)*

X. Yang, H. Yang, K. Chakrabarty, and H. H. Li are with the Department of Electrical and Computer Engineering at Duke University, Durham, NC, USA. (email: xy92@duke.edu)

J. R. Doppa and P. P. Pande are with the School of Electrical Engineering & Computer Science at Washington State University, Pullman, WA, USA.

structured stochastic gradient pruning for ReRAM crossbar structure.
- We demonstrate that *ESSENCE* can successfully reduce the update counts for the ReRAM-based in-memory training process. We further explore the update count map in the selected layer and illustrates that our method can remove the clusters in the tail distribution of the update count and effectively reduce the maximum update count.
- We propose the design of a ReRAM-based counter, which keeps track of the number of writes in a structural manner and introduces limited area overhead.
- We evaluate our method with various CNNs to show its effectiveness and explore the potential of aggressive tuning of update probability based on *ESSENCE*.

The remainder of this paper is organized as follows: Section II discusses related prior work and Section III further explains the preliminaries of ReRAM-based in-memory processing and gradient pruning . Section IV presents the proposed *ESSENCE* methodology. Section V provides the experimental results and analysis. Section VI concludes this paper.

II. Related Work

This section presents an overview of the state-of-the-art ReRAM-based PIM architectures and the associated endurance issue. Then, we discuss the update counts during the weight update stage in selected previous approaches for the ReRAM-based in-memory training system.

A. ReRAM-based PIM Designs

ReRAM-based PIM designs can speed up neural networks like CNN, recurrent neural network (RNN), generative adversarial network (GAN), and attention-based network for cutting-edge tasks like image categorization [3], activity detection [9], image generation [10], and natural language translation [11]. Neural network inferencing with ReRAM-based systems is a welled-studied topic, and thus the existing challenges of ReRAM-based PIM inferencing, such as IR-drop, programming noise, and thermal noise, have become the focus of current studies [12]. Potential solutions include compensation [13], device-variability-aware training [14], and stochastic-noise-aware training [15]. Therefore, utilizing ReRAM-based PIM designs for neural network inference is efficient and reliable. However, for ReRAM-based training tasks, the endurance problem is a major challenge.

B. Endurance Problem

Repeated reprogramming of the ReRAM crossbar is unavoidable for neural network training. Taking training on $50,000$ images from the CIFAR-10 dataset [16] with a batch size of 64 as an example, one training epoch involves 782 reprogramming of each ReRAM cell in weight crossbar. Thus, 300 training epochs will introduce 2.35×10^5 writes. Currently, the endurance for a ReRAM device ranges from 10^6 to 10^{12} cycles [17]–[20]. The endurance problem sets an upper bound for the available number of programming, but repeated and frequent programming is necessary to the ReRAM-based

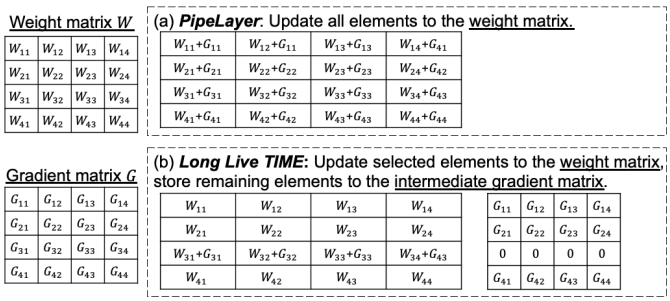


Fig. 1. Conceptual visualizations for the weight update stage in prior work: (a) PipeLayer. (b) Long Live TIME. The learning rate is omitted for simplicity.

neural network training. We need an efficient and intelligent strategy to balance these two parts.

Previous approaches try to propose endurance management methods with hardware-friendly programming strategy. Wen et al. [21] advocated utilizing a programming order of reset operations followed by set operations and shortening the reset operations for less significant bits to reconcile the endurance problem with the programming technique. Zhang et al. [22] also proposed a software-based strategy to bias the weights toward smaller values as well as a hardware mapping solution that takes into account the memristors' aging status. However, these approaches didn't fully exploit the potential of training neural networks with sparse gradients.

Recently, Giordano et al. [23] proposed CHIMERA framework that simplifies the features and gradients with low-rank approximation. In addition, CHIMERA used a large set to avoid frequent ReRAM updates. Note that utilizing a large set for training is always effective in reducing the number of updates for training. Besides, CHIMERA requires off-line training and the proposed low-rank training only shows its effectiveness with pre-trained models.

C. Representative Approaches for Updating Counts during Training

For training in-memory systems, we compare two representative approaches, namely PipeLayer [3] and Long Live TIME [8], and show the conceptual visualizations in Fig. 1. PipeLayer method involves calculating the gradients and updating the weights in each iteration. Here we assume that one crossbar size is $n \times n$. Since ReRAM crossbars function as memory subarray and computing subarray in the design, we assign one crossbar for the weights and another one for the gradients. And the update counts for both weight matrix and gradient matrix are equal to the crossbar size in each iteration. PipeLayer is a standard approach when we don't consider gradient sparsity. The disadvantage of this approach is that all elements in the weight matrix require reprogramming in every training iteration, and this frequent reprogramming compromises the reliability of the overall PIM system.

The second approach Long Live TIME reduces the number of writes for the weight update stage by value-based gradient pruning. For example, if only one row is selected for weight update, the weight update counts will be decreased to size n. However, it is worth noting that this method requires resetting the selected gradient row of size n and storing the remaining

TABLE I
THE UPDATE COUNTS IN THE CROSSBAR OF SIZE N×N FOR K TRAINING ITERATIONS, WHERE 1,2,...,K STANDS FOR THE ITERATION NUMBER, AND G/W STANDS FOR CROSSBAR FOR GRADIENT/WEIGHT MATRIX.

Iteration Index-Crossbar	1-G	1-W	2-G	2-W	...	k-G	k-W	Total
PipeLayer [3]	n^2	n^2	n^2	n^2	...	n^2	n^2	$2k \cdot n^2$
Long Live TIME [8]	n^2	n	$2n^2$	n	...	$2n^2$	n	$(2k-1) \cdot n^2 + k \cdot n$
ESSENCE (this work)	n^2	$c_1 \cdot n$	n^2	$c_2 \cdot n$	...	n^2	$c_k \cdot n$	$k \cdot n^2 + \sum_{i=1}^{n} c_i \cdot n$

gradient. At the next iteration, the new calculated gradient requires writing to an array of size n^2. Then, the remaining gradient will be accumulated to the new calculated gradient, and this step requires writing to a block of size $(n^2 - n)$. We summarize the results for these two methods in Table I.

Note that Long Live TIME method increases the number of writes for the gradient calculation by n^2 compared with PipeLayer and requires additional storage in ReRAM or DRAM. In the first case of storing in supplementary ReRAM crossbars, the resource requirement for weight and gradient matrix in one layer increases to $1.5\times$ compared with the PipeLayer approach because the size of the intermediate gradient matrix is the same as that of the weight matrix. In another case of storing the intermediate result in DRAM, DRAM resource consumes additional power and area and introduces inevitable communication overhead between DRAM and ReRAM in each training iteration. In total, Long Live TIME requires larger number of reprogramming involving the gradient and weight matrix compared with PipeLayer. Therefore, this method cannot solve the frequent reprogramming problem for ReRAM-based in-memory training systems. Thus, the proposed method needs to reduce the weight matrix access without increasing the gradient matrix access. In addition, we do not want to introduce overhead due to the intermediate gradient storage.

III. PRELIMINARIES: RERAM-BASED IN-MEMORY PROCESSING AND GRADIENT PRUNING

In this section, we present the background of in-memory processing using ReRAM-based crossbars. Next, we highlight the important design principles in the ReRAM-based gradient pruning and the formulation of stochastic gradient pruning.

A. In-memory Processing of Neural Network in ReRAM

Fig. 2(a) shows a fully-connected (FC) layer with the connection between input and output neurons. We denote the activation (or input) as x, the weight as W, the bias as b, and the output as y. The forward path computation is captured by the equation of $y = Wx + b$. When the forward path computation is finished, the loss L can be calculated with the targeted output and the actual output using the squared Euclidean norm (L2 norm) or the cross-entropy loss. In the backward path, the gradient with respect to the weight $\frac{\partial L}{\partial W}$ can be computed with the chain rule, following the equation of $\frac{\partial L}{\partial W} = \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial W}$. In this example, we focus on the VMM related to the weight, which is the dominant computation in the forward and the backward path. Accordingly, the ReRAM-based in-memory processing for neural networks should support the VMM with high efficiency.

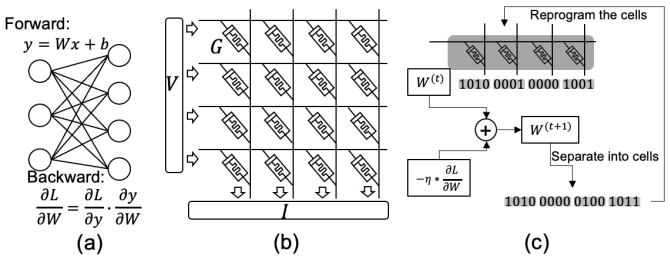


Fig. 2. ReRAM-based processing-in-memory system. (a) Forward propagation using the weight and the activation and backward propagation for the gradient with respect to weight in a neural network. (b) ReRAM-based vector-matrix multiplication computing. (c) An example of weight update in ReRAM-based system, where multiple cells represent one weight element. In this example, a 16-bit weight is separated into four 4-bit ReRAM cells.

ReRAM, as a nonvolatile memory device, can store conductance in each cell, as shown in Fig. 2(b). Mapping the fully-connected kernels to the ReRAM crossbars requires programming the weight in Fig. 2(a) into the ReRAM cells in Fig. 2(b). When mapping the convolution kernels to the ReRAM crossbars, the weights will be unrolled to two-dimensional representation in the first step. For instance, the convolution kernel has the size of $[in\_channel, kernel\_height, kernel\_width, out\_channel]$, and the unrolled kernel has the dimension of $[in\_channel \times kernel\_height \times kernel\_width, out\_channel]$. Then, we can map the unrolled kernels to ReRAM crossbars.

In the computation stage, the voltage will be supplied to the word line. For instance, during the forward path, the input vector is encoded as the voltage vector (denoted as V), and the value of the weight matrix is encoded as the conductance of the ReRAM crossbar (denoted as G). The ReRAM system performs the vector-matrix multiplication according to Kirchhoff's current law and Ohm's law [24]. Consequently, the output current accumulated through the bit line (denoted as I) equals the multiplication result of voltage and conductance.

In our PIM system, the VMM is computed in the analog domain. The peripheral circuits (including the digital-to-analog converter and the analog-to-digital converter) help in converting the signals from analog to digital and vice-versa. For the ReRAM-based in-memory training system, the backward path can also be implemented with the ReRAM-based VMM. In addition, training requires the weight update step, in which the gradient is added to the original weight. Thus, the reprogramming of the ReRAM crossbar is required. If the weight precision (prec_w) is larger than the cell resolution (res_cell), we should update multiple cells (i.e., $\lceil \text{prec}_\text{w}/\text{res}_\text{cell} \rceil$ cells) to refresh one weight element. In Fig. 2(c), we show an example of weight update in ReRAM-based system. Here, $W^{(t)}$ and $W^{(t+1)}$ represent the weight for time step t and $(t+1)$. Parameter η denotes the learning rate. In the example, each weight update starts from reading the conductance values from

the ReRAM cells. Then, the value of $W^{(t)}$ is achieved by shifting and adding the cell conductance. We compute the $W^{(t+1)}$ with the equation $W^{(t+1)} = W^{(t)} - \eta \cdot \frac{\partial L}{\partial W}$. After that, we split the new weight into cells and reprogram the cells. This loop from reading to reprogramming is necessary for the cell-level update in the ReRAM-based in-memory training system.

B. Design Principles of ReRAM-aware Gradient Pruning

As we discussed in Section II-C, the whole gradient matrix will be updated to the weight matrix if no gradient sparsity information is known. This motivates us to explore options to replace updating the whole gradient matrix with only updating the nonzero elements if our method can prune the gradient matrix to have more zero elements. The potential gain of this method lies in the pruning percentage of the gradient. Before discussing details of the ReRAM-based gradient pruning strategy, we want to address two important design principles.

First, we propose to eliminate the intermediate gradient that is required for the gradient sparsification [8]. Gradient sparsification helps in selecting the important gradient row (i.e., the gradient row with the largest magnitude) for weight update. However, the disadvantage of this method is storing the remaining gradients (i.e., gradient matrix except for the selected gradient row) for the next training iteration. Generally, we need to keep the intermediate gradient to ensure the convergence of the overall training process. However, the additional copy of the intermediate gradient adds to the storage and communication overhead. In contrast, we should separate the constructed gradient into zero and nonzero values. In the update stage, all nonzero values should be updated to the weight matrix so that the remaining gradients are zero elements. Therefore, our approach can successfully circumvent the intermediate gradient problem.

Second, we advocate utilizing the optimizer without momentum to avoid the area and latency overhead in ReRAM-based designs. Stochastic gradient descent (SGD) [25] is an effective gradient-based optimizer for neural network training. In addition, momentum [26] or adding momentum to SGD can include the history information to the gradient update, and it leads to faster convergence. Here, momentum is defined as the moving average of the gradient, and thus the storage and computation related to momentum will introduce additional overheads. If ReRAM-based designs manage to support the momentum calculation within the crossbar structures, momentum values should also be stored in ReRAM. Also, the addition of the momentum term and gradient term is required to derive the correct gradient update. Thus, the area and latency overhead related to the momentum is considerable. With an appropriate gradient pruning method, gradient term can have more zero elements, but there is no guarantee of sparsity in the momentum. As a result, including momentum in the update step may increase the number of updates in ReRAM cells. Therefore, we will take the SGD as a preferred optimizer.

C. Stochastic Gradient Pruning Formulation

In this part, we first explain the gradient calculation and the stochastic gradient pruning method. Next, we show that stochastic gradient pruning will not affect the expectation of the gradient value. This gradient pruning method serves as the basis of our framework.

During the training process, we want to find the optimal weight set to minimize the expectation of loss function, which can be written as follows,

$$minimize \quad \mathcal{F}(\mathbf{w}) = \mathbf{E}[\mathcal{L}(\mathbf{w}, \mathbf{x})], \tag{1}$$

where $\mathbf{x}$ denotes the input and $\mathbf{w}$ represents the weight. The gradient at iteration t is achieved by taking the derivative of the loss function $\mathcal{L}(\mathbf{w}, \mathbf{x})$. That is to say,

$$\mathbf{g} = \nabla_{\mathbf{w}} \mathcal{L}(\mathbf{w}, \mathbf{x}). \tag{2}$$

Here we denote the weight at training iteration t as $\mathbf{w}^{(\mathrm{t})}$ and the gradient at training iteration t as $\mathbf{g}^{(\mathrm{t})}$. The weight update formula for the weight at the next training iteration (i.e., $\mathbf{w}^{(\mathrm{t}+1)}$) is defined as:

$$\mathbf{w}^{(\mathrm{t}+1)} = \mathbf{w}^{(\mathrm{t})} - \eta * \mathbf{g}^{(\mathrm{t})}, \tag{3}$$

where the learning rate is denoted as η.

The expectation of stochastic rounding is unbiased, and thus stochastic rounding has been investigated for low-precision neural network training [27] and distributed learning [6]. We utilize the stochastic gradient pruning and round the gradient to its maximum absolute value or zero with the following formula:

$$\mathbf{sgp}(\mathbf{g}^{(\mathrm{t})}) = \mathbf{h}(\mathbf{g}^{(\mathrm{t})}) \circ b(\mathbf{g}^{(\mathrm{t})}) \tag{4}$$

$$\mathbf{sgp}(\mathbf{g}_k^{(\mathrm{t})}) = \begin{cases} \mathbf{h}(\mathbf{g}^{(\mathrm{t})}) & \text{with } p = \frac{|\mathbf{g}_k^{(\mathrm{t})}|}{max(abs(\mathbf{g}^{(\mathrm{t})}))} \\ 0 & \text{with } p = 1 - \frac{|\mathbf{g}_k^{(\mathrm{t})}|}{max(abs(\mathbf{g}^{(\mathrm{t})}))}, \end{cases} \tag{5}$$

$$\mathbf{h}(\mathbf{g}^{(\mathrm{t})}) = max(abs(\mathbf{g}^{(\mathrm{t})})) \cdot sign(\mathbf{g}_k^{(\mathrm{t})}), \tag{6}$$

where $\mathbf{sgp}$ denotes the gradient after stochastic pruning, operator $\circ$ denotes the element-wise multiplication, b denotes a binary indicator for pruning position, and $\mathbf{h}$ covers the maximum absolute value of gradient and the sign of each element. As shown in Equation (5), the probability of choosing the k^{th} element of binary indicator as value 1 or value 0 depends on the absolute value of the k^{th} element of gradient ($|\mathbf{g}_k^{(\mathrm{t})}|$). Note that with the Equations (4) (5) and (6), the expectation of $\mathbf{sgp}(\mathbf{g}^{(\mathrm{t})})$ equals the expectation of $\mathbf{g}^{(\mathrm{t})}$. Besides, with the Equations (1) and (2), we can get the formula of the expectation of the stochastic pruning gradient as

$$\mathbf{E}(\mathbf{sgp}(\mathbf{g}^{(\mathrm{t})})) = \mathbf{E}(\mathbf{g}^{(\mathrm{t})}) = \nabla_{\mathbf{w}^{(\mathrm{t})}} \mathcal{F}(\mathbf{w}^{(\mathrm{t})}), \tag{7}$$

which remains an unbiased gradient towards the minimization target described in Equation (1).

IV. Proposed *ESSENCE* Methodology

This section focuses on the development of the *ESSENCE* methodology. First, we propose the endurance-aware gradient pruning method, which can monitor the previous writing frequency and adjust the current writing probability. As the writing access of the ReRAM crossbar is either a column-wise or a row-wise operation, the structured stochastic gradient pruning is preferable. We introduce the *ESSENCE* method

and explain the endurance-aware structured gradient pruning approach. Finally, we present the overall design of *ESSENCE* and discuss the characteristics of the two components: the computation unit and the counter unit.

A. Endurance-Aware Gradient Pruning

In the stochastic gradient pruning, we prune the gradient based on its magnitude, and the sparsity of the gradient matrix can be increased. For instance, for ResNet20 network with the CIFAR-10 dataset, the average sparsity of the gradient matrix is approximately 71.64% across all layers, as shown in Fig. 3(a). As this pruning doesn't consider the endurance issue, we refer to the stochastic gradient pruning as the "No Endurance" gradient pruning approach. However, note that there is an aggregated usage of specific ReRAM cells for No Endurance approach, as shown in the blue line of Fig. 3(b). For instance, in the last convolution layer, i.e., layer with index 19, the maximum update counts reach 80.52×10^3, while the SGD method has the maximum update counts of 117.30×10^3. The ReRAM cells with the maximum update counts are very highly utilized, and may wear out faster compared to the cells with mean update counts. This result indicates that the mean update count saving from No Endurance method doesn't guarantee a similar-level improvement in terms of update counts for the most frequently updated cell.

Therefore, we introduce an endurance-related factor $\boldsymbol{\alpha}$ to indicate the update intensity of ReRAM cells. Here we denote the existing number of updates in one ReRAM cell as u. Then, similar to the binary indicator $b(\mathbf{g}^{(\mathrm{t})})$ in Equation (4) with Bernoulli distribution, the probability of update this cell in iteration t will be $p(\mathrm{t}) = f(u)$. And function f is the probability function of existing update counts. Then, in training iteration $\mathrm{t}+1$, we will have the existing number of updates in the ReRAM cell as u', and the value of u' is achieved with the following equation:

$$u' = \begin{cases} u+1 & \text{with } p = p(t) \\ u & \text{with } p = 1-p(t). \end{cases} \tag{8}$$

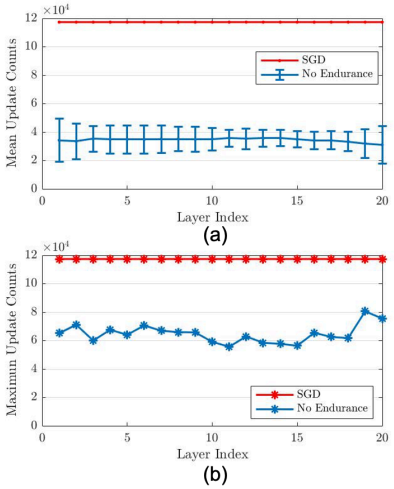


Fig. 3. Update counts for SGD, No Endurance methods. ResNet20 network with the CIFAR-10 dataset training for 300 epochs with the batchsize of 128. (a) Mean update counts in each layer with standard derivation shown as an error bar, (b) Maximum update counts in each layer.

For the iteration $\mathrm{t}+1$, we have

$$\begin{aligned} p(\mathrm{t}+1) = f(u') &= p(\mathrm{t})f(u+1) + (1-p(\mathrm{t}))f(u) \\ &= [f(u+1)+1-f(u)]f(u). \end{aligned} \tag{9}$$

As we want to keep the format of $p(\mathrm{t}+1)$ similar to $p(\mathrm{t})$ and the probability value smaller than 1, we define $f(u)$ as $1-\frac{u}{threshold+u}$. Parameter $threshold$ is learnable for different training tasks. Thus, we will have the endurance-related factor $\boldsymbol{\alpha}$ be represented as

$$\boldsymbol{\alpha}^{(\mathrm{t})} = \frac{1}{f(u^{(\mathrm{t})})} = 1 + \frac{u^{(\mathrm{t})}}{threshold}. \tag{10}$$

The endurance-related factor increases linearly with the existing update counts in ReRAM cells as shown in Equation (10). With the proposed endurance-related factor, we can reduce the probability of update large-$\boldsymbol{\alpha}$ cells. Here, we propose the endurance-aware stochastic gradient pruning strategy (referred to as **esgp**) as following:

$$\mathbf{esgp}(\mathbf{g}^{(\mathrm{t})}, \boldsymbol{\alpha}^{(\mathrm{t})}) = \mathbf{h}(\mathbf{g}^{(\mathrm{t})}, \boldsymbol{\alpha}^{(\mathrm{t})}) \circ (b(\mathbf{g}^{(\mathrm{t})}) \cdot b(\boldsymbol{\alpha}^{(\mathrm{t})})) \tag{11}$$

$$\mathbf{h}(\mathbf{g}^{(\mathrm{t})}, \boldsymbol{\alpha}^{(\mathrm{t})}) = \boldsymbol{\alpha}^{(\mathrm{t})} \cdot max(abs(\mathbf{g}^{(\mathrm{t})})) \cdot sign(\mathbf{g}^{(\mathrm{t})}) \tag{12}$$

$$P(b(\mathbf{g}_k^{(\mathrm{t})}) = 1) = \frac{|\mathbf{g}_k^{(\mathrm{t})}|}{max(abs(\mathbf{g}^{(\mathrm{t})}))} \tag{13}$$

$$P(b(\boldsymbol{\alpha}_k^{(\mathrm{t})}) = 1) = \frac{1}{\boldsymbol{\alpha}_k^{(\mathrm{t})}}. \tag{14}$$

Here we omit the probability functions when each binary indicator equals to 0, because they are easy to obtain from the Equations (13) and (14). Therefore, the probability of update of one cell will be determined by both the amplitude of the gradient and the existing update counts. Following the steps of Equation (7), we can derive the endurance-aware stochastic pruning gradient expectation as

$$\mathbf{E}(\mathbf{esgp}(\mathbf{g}^{(\mathrm{t})})) = \mathbf{E}(\mathbf{g}^{(\mathrm{t})}) = \nabla_{\mathbf{w}^{(\mathrm{t})}} \mathcal{F}(\mathbf{w}^{(\mathrm{t})}), \tag{15}$$

which does not influence the training convergence towards the optimization target.

We apply the endurance-aware stochastic gradient pruning to the training process. In Fig. 4, we collect the maximum update counts of each layer across different training epochs and compare the No Endurance and Endurance-aware approaches. For No Endurance method, we can observe that the linear increase in terms of maximum update counts in Fig. 4(a). In this case, controlling the maximum update counts or decreasing the gap between the worst case and the average

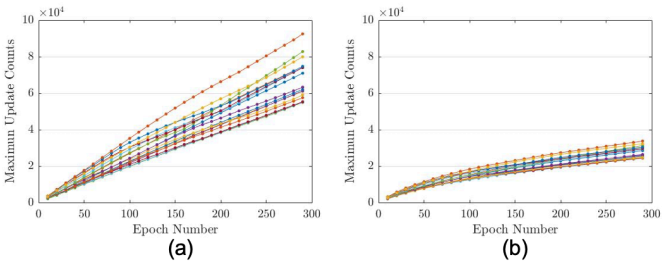


Fig. 4. Maximum update counts of ReRAM cells during the training process. Different lines in the figure represent different layers from the ResNet20 network. (a) No Endurance method, (b) Endurance-aware method.

case is not possible. However, with the endurance-aware stochastic pruning method, we can successfully control the slope of maximum update counts, as shown in Fig. 4(b). We set the same range for $y-$axis in Fig. 4(a) and (b) and find that Endurance-aware method largely reduces the maximum update counts compared with No Endurance method.

In summary, this case study shows the effectiveness of endurance-aware stochastic gradient pruning for ReRAM-based in-memory training systems. This endurance-aware method can automatically decrease the gap between the maximum update counts and average update counts. More importantly, our method can transform the expected sparsity of stochastic gradient pruning in the software approach to a similar level of update savings in hardware approach. The advantage of our method over previous approaches is that our method eliminates the remaining gradient matrix and avoids the additional storage.

B. ESSENCE: Endurance-Aware Structured Stochastic Gradient Pruning

For the No Endurance and Endurance-aware approaches, the pruned gradient position is related to the amplitude of the gradient and the endurance-related factor, and these approaches only offer random pruning. As we write to ReRAM crossbars either row-wise or column-wise manner, we incorporate the gradient pruning in a structured manner. Updating one row or column can be avoided if the gradient matrix has all zeros in that line. Besides, the endurance-aware gradient pruning with random positioning is impractical because the area overhead of setting one endurance-related factor for each ReRAM cell is large. On the other hand, setting the endurance-monitoring factor for each ReRAM row or column, will significantly reduce the overhead. Therefore, we develop the endurance-aware structured stochastic gradient pruning method with the structural position indicator and minimize the overhead of the endurance-related factor.

The proposed method includes the structural information by revising the endurance-related factor. We introduce a structured endurance-related factor $\boldsymbol{\alpha}_{\mathrm{s}}$ to indicate the intensity of ReRAM update in a structured manner. Here we denote the existing number of update in one ReRAM line as u_{s}, and all elements in this line share the same u_{s}. And the relationship of $\boldsymbol{\alpha}_{\mathrm{s}}$ and u_{s} follows the equation of

$$\boldsymbol{\alpha}_{\mathrm{s}}{}^{(\mathrm{t})} = \frac{1}{f(u_{\mathrm{s}}{}^{(\mathrm{t})})} = 1 + \frac{u_{\mathrm{s}}{}^{(\mathrm{t})}}{threshold}. \tag{16}$$

Here we propose the endurance-aware structured stochastic gradient pruning strategy (referred to as **essgp**) as following:

$$\mathbf{essgp}(\mathbf{g}^{(\mathrm{t})}, \boldsymbol{\alpha}_{\mathrm{s}}{}^{(\mathrm{t})}) = \mathbf{h}(\mathbf{g}^{(\mathrm{t})}, \boldsymbol{\alpha}_{\mathrm{s}}{}^{(\mathrm{t})}) \circ (b(\mathbf{g}^{(\mathrm{t})}) \cdot b(\boldsymbol{\alpha}_{\mathrm{s}}{}^{(\mathrm{t})})) \tag{17}$$

$$\mathbf{h}(\mathbf{g}^{(\mathrm{t})}, \boldsymbol{\alpha}_{\mathrm{s}}{}^{(\mathrm{t})}) = \boldsymbol{\alpha}_{\mathrm{s}}{}^{(\mathrm{t})} \cdot max(abs(\mathbf{g}^{(\mathrm{t})})) \cdot sign(\mathbf{g}^{(\mathrm{t})}) \tag{18}$$

$$P(b(\mathbf{g}_k^{(\mathrm{t})}) = 1) = \frac{|\mathbf{g}_k^{(\mathrm{t})}|}{max(abs(\mathbf{g}^{(\mathrm{t})}))} \tag{19}$$

$$P(b(\boldsymbol{\alpha}_{\mathrm{s}}{}^{(\mathrm{t})}) = 1) = \frac{1}{\boldsymbol{\alpha}_{\mathrm{s}}{}^{(\mathrm{t})}}. \tag{20}$$

Note that the binary indicator $b(\boldsymbol{\alpha}_{\mathrm{s}}{}^{(\mathrm{t})})$ in Equation (17) keeps track of the existing update counts of ReRAM and provides structural information. Because the elements in one selected row or column will share one binary indicator for structured pruning position, we don't have an element-wise representation for the endurance-related factor. We illustrate our method in **Algorithm 1**.

Algorithm 1 Endurance-aware Structured Stochastic Gradient Pruning Method

0: Randomly initialize model weights $\mathbf{W}^{(0)}$;
0: Set total iteration T and learning rate η;
0: Set update counter $u_{\mathrm{s}}{}^{(0)}$;
0: **for** $t = 0, \ldots, T$ **do**
0: Sample batch $\mathcal{B}$ from training set;
0: Compute SGD gradient $\mathbf{g}^{(\mathrm{t})} = \nabla L_{\mathcal{B}}(\mathbf{W}^{(\mathrm{t})})$;
0: Compute endurance-related factor $\boldsymbol{\alpha}_{\mathrm{s}}{}^{(\mathrm{t})}$ with Equation (16);
0: Compute the endurance-aware structured gradient $\mathbf{essgp}(\mathbf{g}^{(\mathrm{t})}, \boldsymbol{\alpha}_{\mathrm{s}}{}^{(\mathrm{t})})$ with Equation (17);
0: Update the weights with the endurance-aware structured gradient and update the update counter $u_{\mathrm{s}}{}^{(\mathrm{t})}$;
return model weights $\mathbf{W}^{(\mathrm{T})}$ and update counter results $u_{\mathrm{s}}{}^{(\mathrm{T})}$.
=0

In the *ESSENCE* framework, both the gradient information and the structural endurance-related factor influence the update probability. If the amplitude of the gradient is low or if the existing update number is large, the probability of further updating the cell will be reduced. We present the update count saving results from our *ESSENCE* method in Section V-B.

Inspired by the deep gradient sparsification [7], we find that the pruning method can be aggressive such that only the largest row in a matrix is updated in each iteration. Therefore, we propose to design a more aggressive tuning method for *ESSENCE*, which further reduces the update probability for the ReRAM cells. However, we will not employ the aggressive strategy at the start of the training process since we want to ensure the correct training convergence trend. In addition to the gradient information, we can collect training accuracy during the training process and prepare for the aggressive tuning. When the training accuracy (t_acc) is larger than one training threshold (t_thres), we use the aggressive technique to reduce the update probability for all ReRAM cells and avoid over fitting in the training process. The equation of this aggressive method based on Equation (20) is as follows:

$$P(b(\boldsymbol{\alpha}_{\mathrm{s}}{}^{(\mathrm{t})}, t\_acc) = 1) = \frac{1}{\lceil t\_acc - t\_thres \rceil \cdot \boldsymbol{\alpha}_{\mathrm{s}}{}^{(\mathrm{t})}}, \tag{21}$$

where the probability of update the ReRAM crossbars will be reduced if the training accuracy exceeds the training threshold by 1%. We show the update count saving results from aggressive tuning with different thresholds in Section V-E.

C. Overall Design of ESSENCE

The *ESSENCE* design, which comprises of two loops, is represented in Fig. 5. This design is developed to realize the ReRAM-based in-memory training with the endurance-aware structured gradient pruning method. The left loop deals with the computation in ReRAM-based system and the implementation of weight update in a structured manner, while

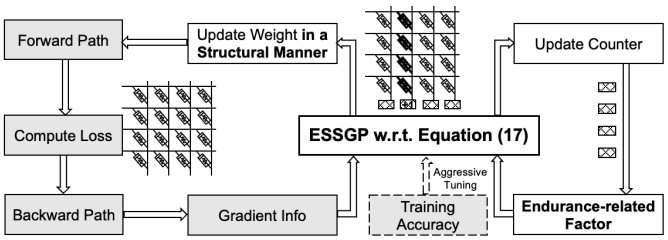


Fig. 5. *ESSENCE* framework that enables the endurance-aware structured stochastic gradient pruning. The grey blocks represent the normal training process for neural networks, the white blocks highlight the proposed *ESSENCE* design components, the dashed block are required for the aggressive tuning. The weight matrix is shown as the black ReRAM cells, and the updated cells in the weight matrix are denoted with the bold lines.

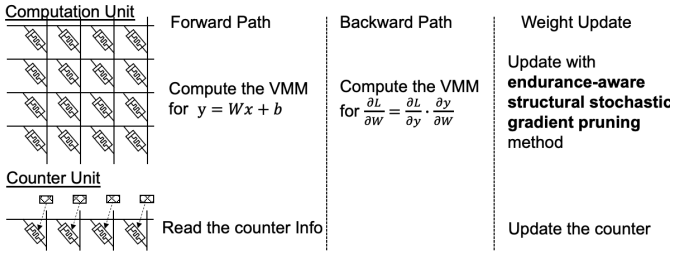


Fig. 6. The stage descriptions of computation unit and ReRAM-based counter unit.

the right loop keeps track of the write access number to the weight matrix. The essgp method provides a bridge for these two loops and considers both the gradient information and the endurance-aware factor. The forward path and backward path follow the ReRAM-based in-memory processing process as explained in Section III-A. The gradient information is propagated from the backward path and used for generating structured sparse gradient. One essential feature is that the components in these two loops can be executed simultaneously, ensuring that training latency is not compromised. In the aggressive tuning setting, the training accuracy is also fed into the center unit.

We summarize the weight update stage of *ESSENCE* as update structured sparse elements to the weight matrix without intermediate gradient storage. Therefore, following our design principles, we save update counts compared with dense gradient update and avoid introduce ReRAM storage overhead.

In addition, we separate the tasks of computation unit and counter unit according to the stages of neural network training, as shown in Fig. 6. Since the details of the computation unit have been described, we will concentrate on the counter unit analysis. We propose ReRAM-based counter for this design. Suppose we set the direction of counter as column-wise. In the case that multiple cells (in our case, with four ReRAM cells and the cell resolution of 4-bit per cell) are utilized to represent one weight element (with the resolution of 16-bit), we should set one counter for one weight column. For the ReRAM-based counter, one counter is made up of 4 cells. The update process for ReRAM-based counter is similar to that of ReRAM-based weight cells. If the selected column is updated in one iteration, we add one to the counter information and reprogram the new value to the counter. The upper bound of number stored in the counter will be $(2^{16} - 1)$. Due to the promising structured sparsity provided by *ESSENCE*, the counter number will not reach the upper bound in the training process.

Furthermore, we show the the tasks for computation and ReRAM-based counter parts in Fig. 6 and discuss two possible patterns for distributing the tasks into different stages. We name the two patterns as conventional pattern and efficient pattern. In the conventional pattern, where the read operation of counter unit and the compute VMM operation of computation unit can be implemented simultaneously in the forward path stage; Then, in the backward path, the gradient data is produced by the computation unit; Afterwards, in the weight update stage, we will update the weight with the help of the *ESSENCE* method and the counter information. However, we find that we can save the design overhead of ReRAM-based counter unit by sharing the ADC with the computation unit. Therefore, in the efficient pattern, we can read the counter information after the VMM computation instead of implementing the counter read and VMM computation simultaneously. This pattern will not influence the overall latency since the counter unit is idle during the backward path. We will adopt the efficient pattern as our preferred design choice because it can help save the area.

V. Experimental Results and Analysis

In this section, we first discuss the experimental setup. We also present details of neural networks under consideration, datasets, and ReRAM designs. Next, we present the experimental results, highlight the advantages of *ESSENCE*, and analyze the counter overhead. Finally, we analyze the results of the aggressive tuning strategy.

A. Experimental Setup

We evaluate *ESSENCE* with the following representative neural networks: ResNet20 [28] and VGG19 with batch normalization (VGG19-BN) [29] on the CIFAR-10 and CIFAR-100 datasets [16]. There are $50,000$ training images and $10,000$ testing images in the CIFAR-10 dataset, which are divided into 10 classes. The CIFAR-100 dataset has the same number of training and testing images as the CIFAR-10 dataset, but these images are grouped into 100 classes. We select the common training approach SGD with momentum (momentum is set as 0.9) as a pure software-based baseline. The inferencing accuracy results of the ResNet20 and VGG19-BN network on the CIFAR-10 dataset are 92.15% and 94.02%, respectively. We include the SGD algorithm without momentum and the update strategy of PipeLayer as the hardware-based baseline method, and denote this baseline as SGD. For the gradient pruning methods, we also exclude the momentum term to guarantee that the ReRAM-based in-memory processing system does not store intermediate gradients. For all the experiments, the learning rate is configured to follow the cosine decay and initialized as 0.25. In total, 300 epochs are required for the training process. The batch size for both training and inferencing is 128. Note that extending the batch size can reduce the training iterations in one training epoch and

TABLE II
RERAM PROCESSING ELEMENT DESIGN DETAILS.

Name	Spec.	Number	Power (mW)	Area (mm^2)
DAC	1bit	8×128	4	0.00017
ReRAM Crossbar	4bit/cell 128×128	8	2.4	0.0002
ADC	8bit,1.2GSps	8	16	0.0096
S+H		8×128	0.01	0.00004
IR	2KB	1	1.24	0.0021
OR	256 B	1	0.23	0.00077

TABLE III
EXPERIMENT RESULTS ON THE CIFAR-10 DATASET.

(a) ResNet20

Methods	Mean Update Counts (Savings)	Accuracy
SGD	117.30×10^3 (1×)	90.67%
No Endurance	33.26×10^3 (3.52×)	90.49%
Essence Row	11.45×10^3 (10.24×)	90.51%
Essence Column	11.40×10^3 (10.29×)	90.93%

(b) VGG19-BN

Methods	Mean Update Counts (Savings)	Accuracy
SGD	117.30×10^3 (1×)	92.66%
No Endurance	25.64×10^3 (4.57×)	92.43%
Essence Row	11.45×10^3 (10.24×)	92.61%
Essence Column	11.58×10^3 (10.13×)	92.41%

thus reduce the update counts of ReRAM cells for the whole training process. However fine tuning the batch size is not the focus of this work. We report the mean update counts for ReRAM cells, update count savings compared with baseline, and the inferencing accuracy.

We set the weight, activation, and gradient precision as 16, 16, and 16 bit. In addition, we report the configurations for the processing element in the ReRAM-based in-memory training system in Table II. As each ReRAM cell in our setup has a resolution of 4 bit, we need 4 cells to represent one weight. The peripheral circuits in *ESSENCE* are adopted from the existing ReRAM-based PIM design [2].

B. ESSENCE Results

1) Results on the CIFAR-10 dataset: We show the ResNet20 and VGG19 network results on the CIFAR-10 dataset, with SGD, No Endurance, Essence Row, and Essence Column methods, in Table III. Our results demonstrate that *ESSENCE* provides promising mean update savings without degrading the accuracy. For instance, with the ResNet20 network, Essence Column can have on average 10.29× update savings and achieve better inferencing accuracy compared with SGD. For these two CNNs, *ESSENCE* framework helps in extending the average life time by up to 10×.

In addition, we illustrate the mean update counts across layers of ResNet20 network in Fig. 7. The mean update and standard derivation results of Essence Column (yellow line) overlap with that of Essence Row (purple line), which matches our result in Table III that these two methods achieve similar saving. We find that the standard derivations of Essence

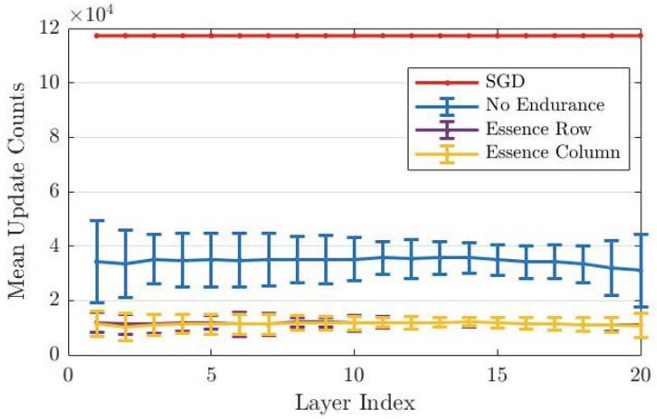


Fig. 7. Mean update counts for SGD, No Endurance, Essence Row, Essence Column methods. ResNet20 network with the CIFAR-10 dataset training for 300 epochs with the batch size of 128. The standard derivation is represented as an error bar.

methods are always smaller than No Endurance method. The standard derivation of Essence reduces as the writing probability of high utilization cells is reduced as shown in the Equation (20). In addition, our observation confirms the effectiveness of *ESSENCE* in reducing mean update counts and balancing write accesses to ReRAM crossbars during training. Furthermore, we discuss the problem of maximum update count and compare our method with No Endurance approach in the next part.

We further analyze the training time required by ESSENCE and SGD+PipeLayer. Note that the training time consists of the computation time in the forward and backward paths, the writing time in the gradient update stage, and the time overhead introduced by the ESSENCE design. Our method can significantly reduce the writing time in the gradient update stage and the overall time saving of *ESSENCE* is 3.18× and 3.17× for the ResNet20 and VGG19-BN network with the CIFAR-10 dataset compared with the SGD+PipeLayer approach. For training with the CIFAR-100 dataset, ESSENCE method achieves 2.80× and 2.72× overall time saving compared with the SGD+PipeLayer approach on the ResNet20 and VGG19-BN network respectively.

2) Analysis of the Tail of Update Count Distribution: As indicated in Fig. 3, the maximum update number is maximum in the last convolution layer, and thus we analyze the update count distribution of this layer and show the effectiveness of *ESSENCE*. First, we look into the distribution of update counts for the No Endurance Method in Fig. 8(a). Here we define the tail of the distribution as the area beyond the third standard derivation (denoted as $\mu+3\sigma$, where μ is the mean and σ is the standard derivation). In our case, the tail of the update count distribution indicates the heavy utilization of the crossbar.

In the next step, we look at the positions of the high utilization cells and try to figure out whether we can decrease the percentage in the tail of distribution with our method. In the No Endurance method, 1.13% of ReRAM cells fall into the tail distribution. Fig. 8(d) illustrates that the high utilization cells form several clusters in the last convolution layer in

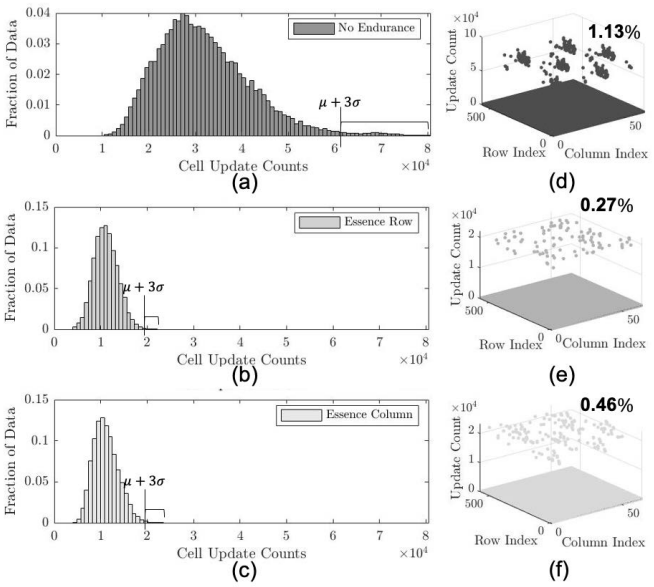


Fig. 8. Left: Update count distribution in the last convolution layer in the ResNet20 network: (a) No Endurance, (b) Essence Row, (c) Essence Column. Right: The positions of the high utilization cells in the last convolution layer in the ResNet20 network: (d) No Endurance, (e) Essence Row, (f) Essence Column. The number in each figure represents the tail percentage of each distribution.

the ResNet20 network. Moreover, the clusters are along the ReRAM columns. The reason for these clusters is that the last convolution layer is connected to the fully-connected layer. The importance of the input neurons in the fully-connected layers are different, i.e., frequent gradient updates on selected neurons are necessary because the amplitude of gradient is relatively large compared with other neurons. Even though the tail percentage of the ReRAM cells is not considerable, the influence of these clusters may bring degradation to the training accuracy. If the high utilization cells are wear out, the conductance values stored in the cells can be faulty. The ReRAM-based PIM designs accumulate the current along the columns for the VMM computation, and thus concurrent faults in one column will accumulate errors in that column. Therefore, the clusters of high utilization cells in the last convolution layer can introduce high fault rate for the computation results in this layer and the final result, which consequently degrades the performance of the neural network.

With the endurance-aware structural stochastic gradient pruning, we have the distributions of update counts in Fig. 8(b) and (c). The mean value in the distribution is shifted to the left compared to the no-endurance method, which confirms that endurance-aware method is effective in reducing the mean update counts and prolonging the life cycles for ReRAM cells. Then, our experimental results show that row counter and column counter approaches can decrease the tail percentage of distribution to only 0.27% and 0.46%. Furthermore, high utilization cells, as shown in Fig. 8(e) and (f), are randomly distributed across the ReRAM crossbar and thus will not introduce the cluster problem. Therefore, *ESSENCE* can improve the overall performance in ReRAM cells as well as save maximum update counts in the worst-case scenario.

C. Design Overhead Evaluation

1) ReRAM-based counter overhead: In our design, we propose to use the ReRAM-based counter to keep track of the number of updates in the ReRAM crossbars. The baseline for the ReRAM-based counter is the CMOS-based counter design. To have a fair comparison, we assume that the counter is column-wise for both cases. In addition, we set the same range of counters. We evaluate the area overhead over the space of 128×128 ReRAM crossbar as shown in Table II. Our results demonstrate that the overhead for a ReRAM-based counter design is only 0.78%, whereas a CMOS-based counter introduces 2.68% area overhead.

In our setting, we need one set of counters for one weight matrix. For instance, in the last convolutional layer in the ResNet20 network, we will have the unrolled kernel of size $[64 \times 3 \times 3, 64]$. We find that matrix row number is in general larger than matrix row number in the 2-dimensional kernel for ReRAM implementation. This indicates that the required counter number for column-wise setting is smaller. Based on our results in Table III(a), the mean update savings and accuracy results for column-wise and row-wise with the ResNet20 network are similar. And the column-wise result with the VGG19-BN network is better in terms of update savings. Therefore, to decrease the overhead of the counter design, we are in favor of the column method.

2) Graident pruning overhead: The additional part in Fig. 5 helps with the endurance-aware structured stochastic gradient pruning. The part of generating stochastic gradient is implemented with digital circuits. We first realize a pseudo number generator with linear-feedback shift register. For the probability in Equation (19), we compare the amplitude of $|\mathbf{g}_k^{(\mathrm{t})}| \cdot 2^n$ against $max(abs(\mathbf{g}^{(\mathrm{t})}))$ multiplied by a random number in the range of 0 to $2^n - 1$. The first part is achieved with a left shift, and the second part is achieved with a multiplier. We select the gradient value ($\mathbf{essgp}(\mathbf{g}^{(\mathrm{t})}, \boldsymbol{\alpha}_{\mathrm{s}}{}^{(\mathrm{t})})$) via a mux. We determine the power and area overheads with Synopsys Design Compiler. The result shows that the structured stochastic gradient pruning components require 0.50% power and 5.13% area overhead.

D. Analysis of ESSENCE Hyperparameters and Scalability

1) Parameter $threshold$ to tradeoff accuracy and update counts: The $threshold$ in Equation (16) is a tunable parameter in our design. If this threshold is set to approach the positive infinity, the endurance-related factor equals to 1. Then, our method converges to No Endurance method. That is to say, our method is a more flexible method compared with No Endurance method. We present the threshold tuning result on the VGG19-BN network on the CIFAR-10 dataset in Fig. 9. Here, we utilize *ESSENCE* with the column counter. Even though our endurance-aware structural pruning method is an unbiased method in terms of the expectation of gradient, it introduces variance to the training process. Therefore, when the threshold is $10,000$, the accuracy result of *ESSENCE* is 91.86%, which suffers from 0.80% degradation compared with the SGD method. Then, with the increase of the threshold, the accuracy results can outperform the SGD method with effective update count savings. In the case of $threshold$ of

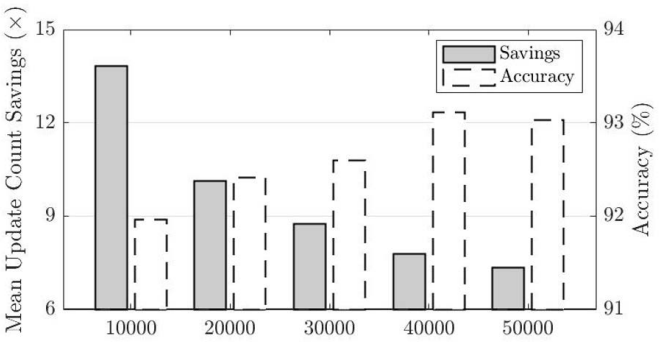


Fig. 9. Threshold tuning of *ESSENCE* with the VGG19-BN network on the CIFAR-10 dataset.

TABLE IV
EXPERIMENT RESULTS ON THE CIFAR-100 DATASET.

(a) ResNet20

Methods	Mean Update Counts (Savings)	Accuracy
SGD	117.30×10^3 (1×)	68.07%
No Endurance	35.69×10^3 (3.29×)	67.55%
Essence Column	17.90×10^3 (6.55×)	67.40%

(b) VGG19-BN

Methods	Mean Update Counts (Savings)	Accuracy
SGD	117.30×10^3 (1×)	73.26%
No Endurance	26.83×10^3 (4.37×)	72.20%
Essence Column	19.65×10^3 (5.97×)	71.45%

$40,000$ or $50,000$, we can achieve higher accuracy as well as update count savings. If our major concern is preserving the accuracy, larger $threshold$ is preferred. In the circumstance that the goal is to boost the update count savings, we can choose the $threshold$ as $20,000$, and the result has only negligible inferencing accuracy loss of 0.25%.

2) Scalability of ESSENCE method: To prove the scalability of *ESSENCE*, we also test on the CIFAR-100 dataset. Again, according to the counter overhead evaluation, we adopt *ESSENCE* with column counter method. With the ResNet20 network, our *ESSENCE* method can achieve the count savings of $6.55\times$ compared with the SGD method; Also, in the VGG19-BN network, our *ESSENCE* method outperforms the SGD by $5.97\times$. We are also sacrificing more inferencing accuracy on the CIFAR-100 dataset. The gradient pruning method, even without the endurance-aware part, suffers from the accuracy loss. The reason is that the classification task becomes more complex due to the aggregated classes. Therefore, the scalability of *ESSENCE* framework is limited by the complexity of classification tasks, but *ESSENCE* still provides an effective method in prolonging the life cycles for ReRAM-based in-memory training system.

The mean update counts shown in Table III and IV are achieved by taking the mean value of all cells (weight parameters). The parameter numbers in ResNet20 and VGG19-BN are different, but our results (the mean number of updates among all cells in the systems) are similar from different networks.

TABLE V
ACCURACY RESULTS UNDER RERAM WRITE VARIATION IN RESNET20 NETWORK.

Method	0.5%	1.0%	1.5%	2.0%
SGD	82.79%	37.40%	23.95%	17.03%
Essence Column	89.71%	88.21%	83.07%	76.73%

The mean update counts results are directly related to the training iteration, batch size, and the effect of gradient pruning. The update counts results is not correlated with the number of network weight parameter. Therefore, the scalability of the proposed technique is not limited by the network structure.

We also conducted experiments with the VGG16 network to confirm the effectiveness of *ESSENCE* on the small-scale model structure. We demonstrate that ESSENCE can achieve $12.98\times$ and $5.08\times$ savings on mean updates for CIFAR-10 and CIFAR-100 datasets respectively. Moreover, there is no accuracy loss compared to the SGD method.

3) ESSENCE Performance under write variation: We further investigate the influence of write variation because it may lead to accuracy degradation. Our *ESSENCE* approach follows the PipeLayer training strategy and doesn't include the write variation in the gradient update stage.

Note that both advanced programming techniques and mature fabrication processes have contributed to reducing ReRAM variation. With homogeneous switching [30] approach, ReRAM variation in the switching process could be mitigated [31]. Besides, with fine modulation of pulse amplitude, the evaluated ReRAM variation is only 5% [32]. In addition, the variation can be reduced to 4.2% with fabricated TiO_2 devices [33]. Furthermore, the writing scheme innovation can help achieve a 0.5% variation level in 4-bit ReRAM devices [34].

To make our work comprehensive, we add one set of experiments to show the influence of writing variation on the baseline and the proposed method. We set variation levels as $0.5\%, 1.0\%, 1.5\%$, and 2.0%. The experimental results are shown in TABLE V.

Our proposed *ESSENCE* method only updates structured sparse gradient to the matrix, and thus the influence of write variation can be reduced compared with the SGD approach. For instance, in each iteration of the SGD approach, all elements in each weight matrix have to be rewritten, and thus the programming variation of ReRAM will influence all elements. Therefore, stable training with the SGD approach cannot be realized under certain level of ReRAM variation. As shown in TABLE V, under 1.0% variation, SGD approach fails to converge to a high accuracy. However, *ESSENCE* only updates parts of the weight matrix, and the inferencing accuracy doesn't degrade as much as the SGD approach. The accuracy improvement achieved by *ESSENCE* under variation level of 2.0% is 59.7% compared with SGD. Therefore, *ESSENCE* is more tolerant to ReRAM write variation.

E. Aggressive Tuning Strategy based on ESSENCE

In an initial experiment, we implement the Essence Column aggressive tuning of a threshold of 87.5% method in the

TABLE VI
AGGRESSIVE TUNING RESULTS WITH VARIOUS TRAINING THRESHOLDS ON THE CIFAR-10 DATASET.

(a) ResNet20

Threshold	Start Ep.	Mean Update Counts (Savings)	Accuracy
87.50%	110	8.80×10^3 (13.33×)	89.59%
90.00%	180	9.57×10^3 (12.26×)	90.22%
92.50%	217	10.32×10^3 (11.37×)	90.45%
95.00%	253	11.06×10^3 (10.61×)	90.71%

(b) VGG19-BN

Threshold	Start Ep.	Mean Update Counts (Savings)	Accuracy
87.50%	20	5.33×10^3 (22.47×)	90.72%
90.00%	40	5.68×10^3 (20.65×)	91.15%
92.50%	108	7.19×10^3 (16.31×)	91.14%
95.00%	168	8.14×10^3 (14.41×)	91.85%
97.50%	210	10.40×10^3 (11.28×)	92.44%

ResNet20 network. This method starts aggressive tuning at epoch 110. However, we notice that the final training accuracy with this threshold is 93.72%, which implies that the training is unsuccessful. To improve the training accuracy, we adjust the threshold for the aggressive method to 90.00%, 92.50%, 95.00%. We don't include the threshold of 97.50% for ResNet20 since the training accuracy cannot reach this number through 300 training epochs. Results in Table VI(a) show that the aggressive tuning further decreases the update counts and maintains the inferencing accuracy. Next, we switch the network to VGG19-BN and collect results in Table VI(b). Since the VGG19-BN network has a larger number of parameters than ResNet20, the training accuracy reaches the targeted threshold at an earlier epoch. For instance, with threshold 90.00% and 95.00%, we can start aggressive tuning at epoch 40 and 168, respectively. As a result, the aggressive method shows promising mean update count savings in the VGG19-BN network.

The aggressive method has to trade off between mean update counts and the inferencing accuracy. In other words, the aggressive method has to balance the endurance issue and the performance target. In the case that we set the tolerable accuracy loss compared with SGD as 1.00%, the proposed method helps in further boosting the mean update count savings from 10.29× to 12.26× for ResNet20, and from 10.13× to 14.41× for VGG19-BN.

F. Comparison with respect to Existing Work

Considering that the endurance for a ReRAM device ranges from 10^6 to 10^{12} cycles, we set the available cycle of ReRAM writes as 10^9. To make a fair comparison, we include two baselines: batch training and single instance training. More specifically, batch training sets the batch size as 128 and updates the gradients after the computation of each batch, while single instance training utilizes the single instance for training and updates the gradients after the computation of a single instance. Note that batch training serves as the baseline for this paper.The comparison results are shown in TABLE VII. The proposed *ESSENCE* method exploits the potential of structured stochastic gradient pruning for in-memory training systems and outperforms existing methods.

Our method can outperform the representative approaches mentioned in Section II-C because this work can reduce the update count during the gradient update stage without compromising the update count in the gradient calculation stage. In addition, our method takes advantage of the potential sparsity of gradient matrix in neural network training and transforms the sparsity into the update count saving, and thus achieves promising update saving results compared with hardware-based solutions [21], [22]. Furthermore, the batch training in the ESSENCE method also provides benefits to the update count saving because frequent gradient updates for each training instance can be effectively avoided, and our method can achieve better performance compared with the pre-train-based method [23]. More importantly, our hardware-software co-design solution is orthogonal to mapping-based solutions, such as [22]. The update saving counts can be further improved with the consideration of *ESSENCE* and detailed mapping optimization.

This work is the first-of-its-kind structured stochastic gradient pruning work. There are existing work targeted at stochastic activation, weight, and gradient pruning [6], [35]–[37]. Prior work ReCOM supports sparse DNN processing in ReRAM-based design with a focus on the weight sparsity [38]. However, this work focuses on the gradient pruning because of the endurance issue of ReRAM device and further proposes the structured gradient pruning because of the preferred programming method of ReRAM crossbar is row-wise and column-wise.

VI. CONCLUSION

We have presented *ESSENCE* framework for ReRAM-based in-memory training system. The endurance-aware structured gradient pruning method can save the mean update counts of up to 10.29× compared with SGD method and effectively reduces the maximum update counts compared with No Endurance method. Our framework includes the design of ReRAM-based counters, which keeps track of crossbar accesses in a structural manner. Moreover, we have explored the potentials of aggressive tuning method based on *ESSENCE* to boost the mean update count savings by up to 14.41×.

TABLE VII
UPDATE COUNT SAVINGS COMPARISON OF EXISTING METHODS AND ESSENCE.

	ReNew [21]	Aging-aware [22]	CHIMERA [23]	ESSENCE (this work)
Compared to batch training	2.83x	11.00x	2.21x	14.41x
Compared to single instance training	362x	1408x	283x	3252x

REFERENCES

[1] P. Chi *et al.*, "PRIME: A novel processing-in-memory architecture for neural network computation in ReRAM-based main memory," in *43rd ACM/IEEE Annual International Symposium on Computer Architecture, ISCA 2016*, pp. 27–39, IEEE Computer Society, 2016.

[2] A. Shafiee *et al.*, "ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," in *43rd ACM/IEEE Annual International Symposium on Computer Architecture, ISCA 2016*, pp. 14–26, IEEE Computer Society, 2016.

[3] 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, HPCA 2017*, pp. 541–552, IEEE Computer Society, 2017.

[4] M. Hu, Y. Chen, J. J. Yang, Y. Wang, and H. H. Li, "A compact memristor-based dynamic synapse for spiking neural networks," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 36, no. 8, pp. 1353–1366, 2016.

[5] X. Yang, B. Taylor, A. Wu, Y. Chen, and L. O. Chua, "Research progress on memristor: From synapses to computing systems," *IEEE Transactions on Circuits and Systems I: Regular Papers*, 2022.

[6] W. Wen *et al.*, "Terngrad: Ternary gradients to reduce communication in distributed deep learning," in *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017*, pp. 1509–1519, 2017.

[7] Y. Lin, S. Han, H. Mao, Y. Wang, and B. Dally, "Deep gradient compression: Reducing the communication bandwidth for distributed training," in *6th International Conference on Learning Representations, ICLR 2018*, OpenReview.net, 2018.

[8] Y. Cai *et al.*, "Long live TIME: improving lifetime for training-in-memory engines by structured gradient sparsification," in *Proceedings of the 55th Annual Design Automation Conference, DAC 2018*, pp. 107:1–107:6, ACM, 2018.

[9] Y. Long, E. M. Jung, J. Kung, and S. Mukhopadhyay, "ReRAM crossbar based recurrent neural network for human activity detection," in *2016 International Joint Conference on Neural Networks, IJCNN 2016, Vancouver, BC, Canada*, pp. 939–946, IEEE, 2016.

[10] F. Chen, L. Song, and Y. Chen, "ReGAN: A pipelined ReRAM-based accelerator for generative adversarial networks," in *23rd Asia and South Pacific Design Automation Conference, ASP-DAC 2018*, pp. 178–183, IEEE, 2018.

[11] X. Yang, B. Yan, H. Li, and Y. Chen, "ReTransformer: ReRAM-based processing-in-memory architecture for transformer acceleration," in *IEEE/ACM International Conference On Computer Aided Design, ICCAD 2020*, pp. 92:1–92:9, IEEE, 2020.

[12] X. Yang, C. Wu, M. Li, and Y. Chen, "Tolerating noise effects in processing-in-memory systems for neural networks: A hardware–software codesign perspective," *Advanced Intelligent Systems*, p. 2200029, 2022.

[13] B. Liu *et al.*, "Reduction and ir-drop compensations techniques for reliable neuromorphic computing systems," in *The IEEE/ACM International Conference on Computer-Aided Design, ICCAD 2014*, pp. 63–70, IEEE, 2014.

[14] 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 2019*, pp. 1769–1774, IEEE, 2019.

[15] X. Yang *et al.*, "Multi-objective optimization of ReRAM crossbars for robust DNN inferencing under stochastic noise," in *IEEE/ACM International Conference On Computer Aided Design, ICCAD 2021*, pp. 1–9, IEEE, 2021.

[16] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," 2009.

[17] M.-J. Lee *et al.*, "A fast, high-endurance and scalable non-volatile memory device made from asymmetric Ta2O5-x/TaO2-x bilayer structures," *Nature materials*, vol. 10, no. 8, pp. 625–630, 2011.

[18] C.-W. Hsu *et al.*, "Self-rectifying bipolar TaOx/TiO2 RRAM with superior endurance over 10^{12} cycles for 3d high-density storage-class memory," in *2013 Symposium on VLSI Technology*, pp. T166–T167, IEEE, 2013.

[19] W. Chen *et al.*, "Switching characteristics of W/Zr/HfO2/TiN ReRAM devices for multi-level cell non-volatile memory applications," *Semiconductor Science and Technology*, vol. 30, no. 7, p. 075002, 2015.

[20] Q. Wu, W. Banerjee, J. Cao, Z. Ji, L. Li, and M. Liu, "Improvement of durability and switching speed by incorporating nanocrystals in the HfOx based resistive random access memory devices," *Applied Physics Letters*, vol. 113, no. 2, p. 023105, 2018.

[21] W. Wen, Y. Zhang, and J. Yang, "RENEW: Enhancing lifetime for reram crossbar based neural network accelerators," in *2019 IEEE 37th International Conference on Computer Design (ICCD)*, pp. 487–496, IEEE, 2019.

[22] S. Zhang, G. L. Zhang, B. Li, H. H. Li, and U. Schlichtmann, "Aging-aware lifetime enhancement for memristor-based neuromorphic computing," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1751–1756, IEEE, 2019.

[23] M. Giordano *et al.*, "CHIMERA: A 0.92 TOPS, 2.2 TOPS/W edge AI accelerator with 2 MByte on-chip foundry resistive RAM for efficient training and inference," in *2021 Symposium on VLSI Circuits*, pp. 1–2, IEEE, 2021.

[24] M. Hu, H. Li, Q. Wu, and G. S. Rose, "Hardware realization of BSB recall function using memristor crossbar arrays," in *The 49th Annual Design Automation Conference 2012, DAC '12*, pp. 498–503, ACM, 2012.

[25] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proceedings of COMPSTAT'2010*, pp. 177–186, Springer, 2010.

[26] N. Qian, "On the momentum term in gradient descent learning algorithms," *Neural networks*, vol. 12, no. 1, pp. 145–151, 1999.

[27] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," in *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015*, vol. 37 of *JMLR Workshop and Conference Proceedings*, pp. 1737–1746, JMLR.org, 2015.

[28] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016*, pp. 770–778, IEEE Computer Society, 2016.

[29] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *3rd International Conference on Learning Representations, ICLR 2015*, 2015.

[30] K. Baek, S. Park, J. Park, Y.-M. Kim, H. Hwang, and S. H. Oh, "In situ tem observation on the interface-type resistive switching by electrochemical redox reactions at a tin/pcmo interface," *Nanoscale*, vol. 9, no. 2, pp. 582–593, 2017.

[31] Z. Wang *et al.*, "Resistive switching materials for information processing," *Nature Reviews Materials*, vol. 5, no. 3, pp. 173–195, 2020.

[32] T. Van Nguyen, J. An, S. Oh, S. N. Truong, and K.-S. Min, "Quantization, training, parasitic resistance correction, and programming techniques of memristor-crossbar neural networks for edge intelligence," *Neuromorphic Computing and Engineering*, vol. 2, no. 3, p. 032001, 2022.

[33] W. Choi, M. Kwak, S. Heo, K. Lee, S. Lee, and H. Hwang, "Hardware neural network using hybrid synapses via transfer learning: WOx nanoresistors and TiOx RRAM synapse for energy-efficient edge-ai sensor," in *2021 IEEE International Electron Devices Meeting (IEDM)*, pp. 23–1, IEEE, 2021.

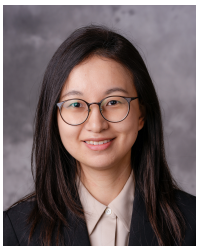
[34] H. Aziza *et al.*, "Multi-level control of resistive RAM (RRAM) using a write termination to achieve 4 bits/cell in high resistance state," *Electronics*, vol. 10, no. 18, p. 2222, 2021.

[35] G. S. Dhillon *et al.*, "Stochastic activation pruning for robust adversarial defense," *arXiv preprint arXiv:1803.01442*, 2018.

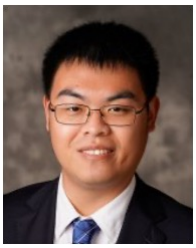
[36] X. Ye *et al.*, "Accelerating cnn training by pruning activation gradients," in *European Conference on Computer Vision*, pp. 322–338, Springer, 2020.

[37] 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.

[38] H. Ji, L. Song, L. Jiang, H. Li, and Y. Chen, "ReCom: An efficient resistive accelerator for compressed deep neural networks," in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 237–240, IEEE, 2018.



Xiaoxuan Yang (Student Member, IEEE) is a Ph.D. Candidate in Electrical and Computer Engineering Department at Duke University, under the supervision of Dr. Hai (Helen) Li and Dr. Yiran Chen in the CEI Lab. She received the B.S. degree in electrical engineering from Tsinghua University and the M.S. degree in electrical engineering from the University of California, Los Angeles. Her research interests include emerging nonvolatile memory technologies, robustness and reliability enhancement in processing-in-memory designs, and hardware accelerators for deep learning applications. She serves as the reviewer for multiple IEEE and ACM journals. Xiaoxuan won Best Research Award at ACM SIGDA Ph.D. Forum at Design Automation Conference (DAC), 2022. She is selected as a Rising Star in EECS by UT Austin in 2022.



Huanrui Yang (Student Member, IEEE) is Postdoctoral Scholar in the EECS department of UC Berkeley, supervised by Prof. Kurt Keutzer. Before joining Berkeley he earned PhD from Duke University in 2022, supervised by Prof. Hai Li and Yiran Chen, and earned Bachelor's degree from Tsinghua University in 2017. His main research interest lies in compressing neural network models with methods like sparsity and quantization, and to evaluate and enhance the robustness of deep learning models. He published multiple papers at conferences such as NeurIPS, ICLR, CVPR, KDD, DAC, ICCAD, etc, and served as reviewer for multiple journals and conferences.

Janardhan Rao Doppa (Senior Member, IEEE) is the Huie-Rogers Endowed Chair Associate Professor at Washington State University. He received his PhD in computer science from Oregon State University. His research interests are at the intersection of machine learning and computing systems design. He won NSF CAREER award, Outstanding Paper Award from AAAI conference (2013), Best Paper Award from ACM Transactions on Design Automation of Electronic Systems (2021), IJCAI Early Career Award (2021), Best Paper Award from Embedded Systems Week Conference (2022), Outstanding Junior Faculty in Research Award (2020) and Reid-Miller Teaching Excellence Award (2018) from the College of Engineering, Washington State University.

Partha Pratim Pande (Fellow, IEEE) is a professor and holder of the Boeing Centennial Chair in computer engineering at the school of Electrical Engineering and Computer Science, Washington State University, Pullman, USA. He is currently the director of the school. His current research interests are novel interconnect architectures for manycore chips, on-chip wireless communication networks, heterogeneous architectures, and ML for EDA. Dr. Pande currently serves as the Editor-in-Chief (EIC) of IEEE Design and Test (D&T). He is on the editorial boards of IEEE Transactions on VLSI (TVLSI) and ACM Journal of Emerging Technologies in Computing Systems (JETC) and IEEE Embedded Systems letters. He was/is the technical program committee chair of IEEE/ACM Network-on-Chip Symposium 2015 and CASES (2019-2020). He also serves on the program committees of many reputed international conferences. He has won the NSF CAREER award in 2009. He is the winner of the Anjan Bose outstanding researcher award from the college of engineering, Washington State University in 2013.

Krishnendu Chakrabarty (Fellow, IEEE) received the B. Tech. degree from the Indian Institute of Technology, Kharagpur, in 1990, and the M.S.E. and Ph.D. degrees from the University of Michigan, Ann Arbor, in 1992 and 1995, respectively. He is now the John Cocke Distinguished Professor of Electrical and Computer Engineering at Duke University.

Prof. Chakrabarty is a recipient of the National Science Foundation CAREER award, the Office of Naval Research Young Investigator award, the Humboldt Research Award from the Alexander von Humboldt Foundation, Germany, the IEEE Transactions on CAD Donald O. Pederson Best Paper Award (2015), the IEEE Transactions on VLSI Systems Prize Paper Award (2021), the ACM Transactions on Design Automation of Electronic Systems Best Paper Award (2017), multiple IBM Faculty Awards and HP Labs Open Innovation Research Awards, and over a dozen best paper awards at major conferences. He is also a recipient of the IEEE Computer Society Technical Achievement Award (2015), the IEEE Circuits and Systems Society Charles A. Desoer Technical Achievement Award (2017), the IEEE Circuits and Systems Society Vitold Belevitch Award (2021), the IEEEHKN Asad M. Madni Outstanding Technical Achievement and Excellence Award (2021), the Semiconductor Research Corporation Technical Excellence Award (2018), and the IEEE Test Technology Technical Council Bob Madge Innovation Award (2018). He is a 2018 recipient of the Japan Society for the Promotion of Science (JSPS) Invitational Fellowship in the "Short Term S: Nobel Prize Level" category.

Prof. Chakrabarty's current research projects include: design-for-testability of 3D integrated circuits a d system-on-chip; AI accelerators and neuromorphic computing; microfluidic biochips; hardware security; AI for healthcare. He is a Fellow of ACM, IEEE, and AAAS, and a Golden Core Member of the IEEE Computer Society.

Hai (Helen) Li (Fellow, IEEE) is Clare Boothe Luce Professor and Chair of the Electrical and Computer Engineering Department at Duke University. She received the B.S. and M.S. degrees from Tsinghua University, Beijing, China, and the Ph.D. degree from the Department of Electrical and Computer Engineering, Purdue University, West Lafayette, IN, USA, in 2004. Prior to joining Duke University, she has been working with Qualcomm Inc., Intel Corporation, Seagate Technology, the Polytechnic Institute of New York University, and the University of Pittsburgh.

Prof. Li serves as Associate Editor-in-Chief of IEEE Transactions on Circuits and Systems I (TCAS-I), served as Senior Editorial Board member of IEEE Journal on Emerging and Selected Topics in Circuits and Systems (JETCAS), as well as Associate Editors of multiple IEEE and ACM journals. She has served as general chair and technical program chair of multiple IEEE conferences, including DAC, ISLPED, ISVLSI, SoCC, and ISQED, and the Technical Program Committee members of over 30 international conference series. She has been on the steering committee of ISVLSI and iNIS since 2016. Dr. Li serves on the IEEE Fellow committee.

Prof. Li's research interests include neuromorphic computing systems, machine learning and deep neural networks, memory design and architecture, and cross-layer optimization for low power and high performance. She has authored or co-authored more than 300 technical papers in peer-reviewed journals and conferences and a book entitled Nonvolatile Memory Design: Magnetic, Resistive, and Phase Changing (CRC Press, 2011). She received 9 best paper awards and an additional 9 best paper nominations from international conferences.

Prof. Li is a Distinguished Lecturer of the IEEE CAS society (2018-2019) and a distinguished speaker of ACM (2017-2020). Prof. Li is a recipient of the NSF Career Award, DARPA Young Faculty Award (YFA), TUM-IAS Hans Fischer Fellowship from Germany, and ELATE Fellowship (2022). She is a fellow of ACM and IEEE.