Optimizing for In-Memory Deep Learning With Emerging Memory Technology

Zhehui Wang, Tao Luo, Rick Siow Mong Goh, Wei Zhang, and Weng-Fai Wong, Senior Member, IEEE

Abstract—In-memory deep learning executes neural network models where they are stored, thus avoiding long-distance communication between memory and computation units, resulting in considerable savings in energy and time. In-memory deep learning has already demonstrated orders of magnitude higher performance density and energy efficiency. The use of emerging memory technology (EMT) promises to increase density, energy, and performance even further. However, EMT is intrinsically unstable, resulting in random data read fluctuations. This can translate to nonnegligible accuracy loss, potentially nullifying the gains. In this article, we propose three optimization techniques that can mathematically overcome the instability problem of EMT. They can improve the accuracy of the in-memory deep learning model while maximizing its energy efficiency. Experiments show that our solution can fully recover most models’ state-of-the-art (SOTA) accuracy and achieves at least an order of magnitude higher energy efficiency than the SOTA.

Index Terms—Deep learning, emerging memory technology (EMT), in-memory computing, optimization.

I. INTRODUCTION

DEEP learning neural networks (DNNs) are widely used today in many applications, such as image classification and object detection. High throughput and high energy efficiency are two of the pressing demands on DNNs these days. However, current computation mechanisms are not the best choices for DNNs in terms of efficiency. As DNN models become increasingly complex, the computation process takes a lot of time and energy [1]. In traditional von Neumann architectures, storage memory is separated from computation operations. To compute an output feature map, we need to read the input feature map and weights from the memory units and send them to the computation module to compute before writing the results back to the memory units. During the whole process, the system spends a large portion of energy and time in data movement [2]. This is made worse with advances in process technology, making the relative distances involved even longer and thus more costly.

Emerging memory technologies (EMTs), including phase change random-access memory (PCRAM), spin-transfer torque magnetic RAM (STT-RAM), and ferroelectric RAM (FeRAM) [3], promise better density and energy efficiency, especially since many are nonvolatile and well suited to the read-mostly applications. In-memory deep learning using EMT cells, especially in analog mode, has already demonstrated an order of magnitude better performance density and two orders of magnitude better energy efficiency than the traditional deep learning on the MNIST dataset [4], [5], [6]. Due to its computational efficiency, in-memory computing has drawn a lot of attention from the community. The impact of in-memory computing on machine learning applications has been explored theoretically and experimentally in the literature [7], [8]. In recent years, with inspiration from new research interest in neural science, many artificial intelligence (AI) applications and platforms utilizing in-memory computing have been proposed [9], [10], [11], [12].

As in-memory deep learning integrates computation with the memory operations [13], the computation results can be directly read from the memory modules using a single instruction. This is different from traditional deep learning, where the memory operations are executed separately from the computation operations. Computing where data are stored reduces the need to move large amounts of data around frequently. Especially when technology scales and on-chip distances become longer, we can expect substantial savings of time and energy if the EMT is used in the in-memory deep learning paradigm. Today, the majority of research papers for in-memory deep learning are based on the analog mode [14], which uses high-precision memory cells. In this analog mode, we use only one EMT cell to store one parameter in the model. This mode is also pursued by the industry via analog flash devices [15]. Overall, the analog mode is more energy and area efficient compared with other working modes.

However, the analog mode usually aggravates a major problem of in-memory computing systems, that is, the instability of EMT cells [16], [17]. Unlike traditional memory technology, where the data are stored in stable memory cells, data stored in analog mode EMT cells may fluctuate, and different values can be outputted. For instance, suppose that we store the weight $w$ in an EMT cell. When we read it from the EMT cell, the output may become $w + \Delta w$ instead of $w$. Here, $\Delta w$ is a fluctuating amplitude of that memory cell. Because of this instability, in-memory deep learning using EMT, especially in
an analog manner, may make incorrect classification [18]. This instability of EMT cells can severely limit its application in the real world.

Table I lists the state-of-the-art (SOTA) methods and our solutions that can alleviate the weight fluctuation problem of in-memory computing. Existing solutions can be classified into two categories. The first category is device optimization, including binarized encoding [19], weight scaling [20], and fluctuation compensation [21]. These solutions can alleviate the fluctuation of devices by tuning the configuration parameters or modifying the device’s configurations. The second category is model optimization, including model compression [22] and knowledge distillation [23]. They cannot alleviate the weight fluctuation directly. However, with a reduced number of operations and increased energy budget for each cell operation, the weight fluctuation can be physically diminished [20].

Existing solutions only optimize a single dimension of the in-memory deep learning systems: the device or the model. The limitation on searching space may lead to suboptimal results. This article proposes a new design approach that can simultaneously optimize all system dimensions. We call it device, dataset, and model optimization, a new design paradigm for in-memory computing deep learning. In addition to the device and the model, we can also optimize the dataset for better performance. Because of the increased optimizing space, our new design paradigm is more likely to find optimal solutions than single-dimension solutions. Experiments show that the performance of our solution is better than the previous results. This article proposes a new design approach that can alleviate weight fluctuation. We prove the effectiveness of all these techniques both theoretically and experimentally. They can substantially improve models’ accuracy and energy efficiency for in-memory deep learning. We organize the rest of this article as follows. Section II gives related work. Section III presents the background knowledge. Section IV introduces our optimization method. Section V shows the experiment results. Finally, Section VI concludes this article.

### II. RELATED WORK

Binarized encoding is the first solution from the device optimization class that can alleviate weight fluctuation. The information in each memory cell is digitized into 1 bit instead of being stored as a full-precision number. In other words, the data stored are either 1 or 0 [24]. Theoretically, the 1-bit data are more robust than a high-precision number at the same level of fluctuation. Several previous works used the 1-bit design to compute the binarized neural networks. Sun et al. [25] used the single-bit cell to execute the XNOR operations in XNOR-Net. Chen et al. [26] used the single-bit cell to execute the basic operations in binary DNN. Tang et al. [27] proposed a customized binary network for the single-bit cell. However, either the XNOR-Net or the binary neural network has a large accuracy drop compared with the full-precision model [28]. Recently, new progress in this research direction is to store a high-precision weight using a group of single-bit memory cells. For example, Zhu et al. [19] used single-bit memory cells \( N \times N \) to store a \( N \)-bit weight. Such a method can increase the model accuracy because it can increase the effective precision of weights. Compared with the traditional design, it uses more memory cells.

Weight scaling is the second solution from the device optimization class that can alleviate weight fluctuation. Theoretically, we can reduce the amplitude of weight fluctuation by scaling up the weight values stored in the memory cell [29]. After computation, we scale the result down using the same scaling factor. In the literature, many research works have found other physical ways to reduce the amplitude of weight fluctuation. For example, He et al. [30] found that we can reduce the fluctuation amplitude by lowering the operation frequency. Chai et al. [31] found one material, which has a lower fluctuation amplitude than the other types of material. However, these methods demand strict physical conditions. Compared with them, weight scaling is a more general method that can reduce the fluctuation amplitude of memory cells in most conditions [32]. However, Choi et al. [33] found that although the memory cell using scaled weights showed smaller fluctuation amplitude, it also consumed higher energy consumption. Ielmini et al. [20] modeled the relationship between the scaling factor, the fluctuation amplitude, and the energy consumption, which could help us to find the optimal scaling factor for in-memory computing.

Fluctuation compensation is the third solution from the device optimization class that can alleviate weight fluctuation. To alleviate the instability, they first read the memory cell by multiple times and then record the statistical results such as mean and standard deviation [34]. Afterward, they either calibrate the model parameter or the model output directly.

| TABLE I |
| --- |
| **SOTA SOLUTIONS AND OURS** |
| **SOTA: Device Optimization** | Binarized Encoding |
| **SOTA: Model Optimization** | Weight Scaling |
| **Ours: Device, Dataset and Model Optimization** | Fluctuation Compensation |
| **SOTA: Model Optimization** | Model Compression |
| **Ours: Device, Dataset and Model Optimization** | Knowledge Distillation |
| **SOTA: Model Optimization** | Energy Regularization |
| **Ours: Device, Dataset and Model Optimization** | Low-fluctuation Decomposition |
based on that statistical results [35]. This method is also widely used during the memory cell programming stage. For example, Joshi et al. [36] compensated the programming fluctuation by tuning the batch normalization layer parameters. Alternatively, Zhang et al. [37] compensated the programming fluctuation by offsetting the weight values. These methods are effective in a static device environment. If we face a dynamic environment, a more general way is needed. One popular approach is to have many equivalent models running in parallel. Then, we calculate the mean of the results. Joksas et al. [38] did this by applying the committee machine theory into the in-memory computing devices. Wan et al. [21] optimized this process by running a single model on the same device and reading the memory cells multiple times. This method can average out the fluctuation and get a more stable result.

Model compression [39], [40] and knowledge distillation [41] are two solutions from the model optimization class that can alleviate weight fluctuation. Since they reduce the number of operations, we can increase the energy budget for each read operation without exceeding the total budget, thus physically diminishing the cell’s instability [20]. One model compression technique is pruning, where we remove some parameters or channels from the model for a smaller model size. Some SOTA model compression (pruning) works are [22], [42], [43], [44], [45], and [46]. Another model compression technique is quantization, where we lower the precision of weights and activations. Quantization is a default technique for EMT cells because the data need to be quantized before we store it in the cell [14]. In knowledge distillation, we transfer the knowledge from a large teacher model to a smaller student model so that the student can have a close accuracy as the teacher, with fewer parameters. Some SOTA knowledge distillation works are [23], [47], [48], [49], [50], and [51].

III. PRELIMINARIES

EMT-based in-memory deep learning can be very efficient [4] because of its analog operation. Fig. 1(a) shows the difference between the traditional and EMT memory cells. When we read a weight \( w \) from the traditional memory cell, the input to the corresponding memory cell is 1, meaning that the read request to that memory cell is enabled. Afterward, the memory cell returns \( w \) as an output. An EMT cell for in-memory deep learning is quite different. When we read the weight \( w \), the input to the memory cell is a variable \( x \) instead of the fixed data 1. The memory cell then returns \( x \cdot w \) directly, the product of the input signals \( x \) and the stored weight \( w \). In other words, the EMT cell integrates the multiplication operation into the read operation.

Analog EMT is more efficient than traditional memory not only in multiplication operations but also in addition operations [32]. To better explain this, we show how traditional cells and EMT cells execute the multiply–accumulate (MAC) operation in Fig. 1(b) and (c), respectively. For traditional memory cells, we first read weight \( w_i \) from the corresponding memory cells. Afterward, the output \( w_i \) is multiplied by the activation \( x_i \) using a multiplier. Finally, we sum all products \( x_i \cdot w_i \) from each of the multipliers together either by a single adder sequentially or by a tree of adders to perform the sum in parallel. To achieve the same computation using EMT-based in-memory computing, we just need to connect the output of each memory cell to the same port. Physically, the sum of all the memory cell outputs \( \sum x_i \cdot w_i \) can be obtained from that port directly. This is also referred to as a current sum.

A. Challenges of Analog In-Memory Deep Learning

The energy consumption of an EMT cell is much less than that of traditional memory cells when they execute the MAC operation. There are important differences. In traditional memory cells, energy consumption is not related to the weight value that is stored. In analog EMT, it is proportional to the weight value [53], as shown in Fig. 2(a). We use a parameter \( \rho \) to denote the ratio between the energy consumption and the weight value. We call this ratio as the energy coefficient. This parameter is tunable, and we can use it to optimize the energy consumption of EMT cells. Theoretically, a small coefficient \( \rho \) can help us improve the energy efficiency of models.

A big challenge in using the emerging memory technology is that regardless of the actual technology, their memory cells do not output stable results. Several works in the literature have modeled such fluctuation [16], [54]. Physically, each memory cell has multiple states, and it changes its state with time randomly. Whenever we read the memory cell, it can be in any of the states. We use the function \( \bar{w} = r_l(w, \rho) \) to generalize the relationship between the actual weight \( \bar{w} \) we read from the EMT cell with unit input data and the prestored weight value \( w \) when the cell is staying at state \( l \). As this relationship is also related to the energy coefficient \( \rho \), we include \( \rho \) as the second input of the function. Given an input \( x \), the output data become \( x \cdot r_l(w, \rho) \) instead of \( x \cdot w \). Different types of EMT cells have
Fig. 2. (a) Relationship between the energy consumption of the EMT cell and the value of the weight. (b) Probability distribution of the weight fluctuation, with different energy coefficients $\rho$. The traditional cell is marked in gray color for reference.

different functions of $r_l(w, \rho)$. Fig. 2(b) shows one model [20] for $r_l(w, \rho)$, as expressed in the following equation:

$$r_l(w, \rho) = w \cdot (1 + c_l/\rho)$$

(1)

where inputs $w$ and $\rho$ are the prestored weight and the energy coefficient, respectively. $c_l (c_l > 0)$ is the intrinsic coefficient related to the specific model of the EMT cell at the $l$th state. In Fig. 2(b), we show a simple case where the cell has only two states, with $c_0 = 0$ and $c_1 = 0.02$ pJ. When we read the weight from the EMT cell, it has a 50% probability of staying at state 0 and another 50% probability of staying at state 1. If the store weight is $w$ and the energy coefficient is $\rho$, the output result can be either $x \cdot r_0(w, \rho)$ or $x \cdot r_1(w, \rho)$ depending on its state, where $x$ is the input activation. The bottom three subgraphs in Fig. 2(b) correspond to memory cells with three different values of energy coefficient $\rho$.

The fluctuation shown in Fig. 2(b) is a simplified example. The practical EMT cells have complicated physical characteristics. A model can be obtained during the postfabrication stage by measuring its complex nonideal properties and then stage by measuring its complex nonideal properties and then fit the model into the data by finding the exact number of fluctuation states and the probability of each state [55].

B. Incompatibility of Traditional Training Method

Fig. 3 shows the standard training process in deep learning. The loss function takes the dataset and the model and generates a measure of the distance between the current parameter values and their optimal values. The optimizer uses gradient descent to reduce this distance by updating the parameters. This process would iterate for many epochs until the optimizer can find an optimal set of parameters. We define $X \in \mathcal{X}$ as the image data and $Y \in \mathcal{Y}$ as the label data. Here, $\mathcal{X}$ and $\mathcal{Y}$ denote the spaces for images $X$ and labels $Y$. For simplicity, we express a one-layer neural network model in the following equation:

$$Y = f(X) = WX + B$$

(2)

where the weight $W$ and the bias $B$ are both trainable parameters. Let us define the function class $F \subset \mathcal{Y}^X$ as the search space of function $f$. Let $Z = \mathcal{X} \times \mathcal{Y}$ be the combination of $\mathcal{X}$ and $\mathcal{Y}$ and $Z = (X, Y)$ be the combination of $X$ and $Y$. Define $\mathcal{D}$ as the unknown distribution of data $Z$ on space $\mathcal{Z}$. Given the above definitions, the loss function can be expressed as $L : F \times Z \rightarrow \mathbb{R}$. It is the mapping from the combination of $\mathcal{F}$ and $Z$ to the real number space $\mathbb{R}$. Theoretically, the training process of the neural network is to find the optimal function $f$ from the function class $\mathcal{F}$ that can minimize the risk, that is, the expectation of the loss function. We express it in the following equation:

$$R[f] = \mathbb{E}_{z \sim \mathcal{D}}[L(f, z)].$$

(3)

The difficulty in solving this problem is that the distribution $\mathcal{D}$ is unknown. What we have is just the dataset $\{Z_1, \ldots, Z_N\}$, which are independent and identically distributed (i.i.d.) samples from the distribution $\mathcal{D}$. Alternatively, the traditional training process of the neural network is to find the optimal model $f$ from the function class $\mathcal{F}$ that can minimize the empirical risk, that is, the average of loss functions on the sampled dataset. We express it in the following equation:

$$R_s[f] = \frac{1}{N} \sum_{i=1}^{N} L(f, z_i).$$

(4)
The distance between the risk and the empirical risk is called the generalization error $\epsilon$, expressed in (5). The crucial problem of the training process is to make sure that the generalization error $\epsilon$ can be bounded by which we decompose the computation involved into several time steps. This decomposition achieves high model accuracy and energy efficiency. We shall now give the mathematical basis of these three techniques.

IV. OPTIMIZING FOR IN-MEMORY DEEP LEARNING

Our training method for in-memory computing can effectively improve the model accuracy, with improved energy efficiency. Fig. 4 shows the overview picture. Essentially, we propose three optimization techniques and integrate them into the training process. The first technique is called device-enhanced datasets. This technique integrates device information as additional data into the dataset. The second technique is called energy regularization, which we add a new regularization term into the loss function that makes the optimizer reduce the energy consumption automatically. The third technique is called low-fluctuation decomposition by which we decompose the computation involved into several time steps. This decomposition achieves high model accuracy and energy efficiency. We shall now give the mathematical basis of these three techniques.

A. Device-Enhanced Dataset

Our first optimization technique is to enhance the dataset with device information. In addition to the regular image data $X$ and the label data $Y$, our enhanced dataset has another source of data, the fluctuation $S$, which reflects the random behavior of memory cells. Fig. 5(a)–(d) shows an visualized example of $X$, $Y$, and $S$, with four training data. They can be classified as either letter A or letter B. Images in the same class can have different variants. For instance, it can be in any font, either normal [Fig. 5(a)] or italic [Fig. 5(c)]. No matter what the variant is, its pixels must follow $D_A$, the distribution for class A. After training, we can accurately classify any image that belongs to class A as long as its pixels follow the distribution $D_A$ [Fig. 5(e)]. This truth also holds for images of letter B, whose pixels follow the distribution $D_B$. The fluctuation data $S$ reflect the random states of memory cells. In our visualized example, the locations of pixels indicate the states of the memory cells. The patterns for fluctuation $S$ also follow a certain distribution $R$, which can be learned during training. Using this enhanced dataset, the model can make correct predictions for in-memory deep learning because now it becomes aware of the fluctuations [Fig. 5(e) and (f)].

As we integrate the fluctuation device data into the dataset, the model will not overfit during training. In Fig. 6, we visualized an example of training using only the dataset $X$ and do not consider the device information. All pixels in the data...
S are in the center of the matrix, indicating the absence of device information [Fig. 6(a) and (b)]. During training, the model will overfit this static data S because it does not have any variant. As we can see from Fig. 6(c), the distribution learned by the model is only a straight line (orange), which is different from the real fluctuation of memory cells (purple). Therefore, the model will misclassify the images. On the other hand, if we include the fluctuation data S, the overfit can be avoided. As we can see from Fig. 6(e) and (f), the fluctuation of memory cells (purple) will follow the learned distribution (orange) so that the model can accurately classify the images using the EMT cells.

We use Figs. 5 and 6 to illustrate why we need to merge the fluctuation information into the data. As the EMT cell output fluctuating weights, the neural networks cannot successfully predict the image unless it has learned both the data distribution and the fluctuation distribution. To achieve this, we can integrate the fluctuation information into the data so that the model can learn the fluctuation distribution during the regular training process.

We developed a method to integrate the fluctuation data S into the data. To simplify this problem, we first decompose the computation of neural network model [see (2)] into several subtasks. For example, each element yij in the output matrix Y can be computed independently using (7). Here, vector w is the ith row in weight matrix W, vector xj is the jth column in the input matrix X, and bj is the element in the bias matrix B at the ith row and the jth column

\[ y_{ij} = w_i x_j + b_j. \]  

As shown in Fig. 2, the weight we read from the EMT cell is unpredictable. Physically, the memory cell changes its status randomly, and the exact output value depends on the state of the memory cell when we are reading it. We denote w_ik be the kth element in vector w, \( \tilde{w}_{ik}(j) \) is the sampled data when we read w_ik from the memory cell and multiply it with x_kj, the element in the input vector x_j. Mathematically, we can express \( \tilde{w}_{ik}(j) \) as a polynomial, which is shown in the following equation:

\[ \tilde{w}_{ik}(j) = \sum_{l=1}^{m} (r_l(w_{ik}, \rho) \cdot s_{ijkl}). \]  

We use \( r_l(w_{ik}, \rho) \) to denote the weight retrieved when the memory cell is at the lth state. It can be considered as a function of the prestored weight w_ik and energy coefficient \( \rho \). At any moment, each memory cell can only be in one state. Hence, we use a one-hot encoded vector \( [s_{ijkl}]_{1 \leq l \leq m} \) to indicate the state of the memory cell when w_ik is sampled. The value of \( s_{ijkl} \) is shown in (9). For given indexes i, j, and k, if the corresponding memory cell is at the l0th state, only \( s_{ijkl0} \) equals 1 and all the other coefficients equal 0

\[ s_{ijkl} = \begin{cases} 1, & \text{if } l = l_0 \\ 0, & \text{if } l \neq l_0. \end{cases} \]  

As we can see from (8), the sampled weight \( \tilde{w}_{ik} \) from the memory cell consists of two parts. We summarize them as follows.

1) The deterministic parameter \( r_l(w_{ik}, \rho) \) is a function indicating the returned value from the memory cell storing weight w_ik for a memory cell that is in the lth state. We denote the matrix \( [r_l(w_{ik}), \rho], \) as \( r(w_i, \rho) \), shown in (10). \( r(w_i, \rho) \) can be considered a function of the weight vector \( w_i \) and the energy coefficient \( \rho \).

2) The stochastic parameter \( s_{ijkl} \) is a random coefficient indicating whether the memory cell storing weight w_ik is at the lth state when it is sampled and multiplied with the input vector \( x_j \). We denote matrix \( [s_{ijkl}] \) as \( S_{ij} \), shown in (11). \( S_{ij} \) can be considered as a part of fluctuation data S

\[ r(w_i, \rho) = \begin{bmatrix} r_1(w_{i1}, \rho) & r_1(w_{i2}, \rho) & \cdots & r_1(w_{in}, \rho) \\ r_2(w_{i1}, \rho) & r_2(w_{i2}, \rho) & \cdots & r_2(w_{in}, \rho) \\ \vdots & \vdots & \ddots & \vdots \\ r_m(w_{i1}, \rho) & r_m(w_{i2}, \rho) & \cdots & r_m(w_{in}, \rho) \end{bmatrix} \]  

\[ S_{ij} = \begin{bmatrix} s_{ij11} & s_{ij21} & \cdots & s_{ijn1} \\ s_{ij12} & s_{ij22} & \cdots & s_{ijn2} \\ \vdots & \vdots & \ddots & \vdots \\ s_{ij1m} & s_{ij2m} & \cdots & s_{ijnm} \end{bmatrix}. \]  

We can now integrate the fluctuation data S into the data for in-memory deep learning. Each element y_{ij} in the output matrix Y can be calculated using (12). y_{ij} is a function of both the deterministic and stochastic parameters. Here, \( \tilde{w}_i \) refers to the sampled weight vector read from the memory cell. For simplicity, we assume that the bias b_j is a deterministic parameter. In some cases, the bias is also fluctuating. We can use the same method to separate deterministic and stochastic parameters for b_j

\[ y_{ij} = \tilde{w}_i x_j + b_j = 1(r(w_i, \rho) \circ S_{ij}) x_j + b_j \]

\[ = \text{tr}(r(w_i, \rho) \text{diag}(x_j) S_{ij}) + b_j \]

\[ = \text{tr}(r(w_i, \rho) \tilde{X}_j) + b_j. \]  

The \( \circ \) operator between \( r(w_i, \rho) \) and \( S_{ij} \) is the Hadamard product, that is, elementwise product. The unit vector 1 is expressed as \( 1 = [1 1 \cdots 1] \), which has m elements. We use it to sum up the entire column of the target matrix. The operator tr is the trace of the matrix, which is defined as the sum of elements on the matrix’s main diagonal. The operator diag turns vector into a diagonal matrix.

Based on (13), we can now merge the device influence S into the data X and obtain the device-enhanced data \( \tilde{X} \). Note that the notation \( \tilde{X}_j, S_{ij}, \) and \( x_j \) is a part of the matrix \( X, S, \) and \( X, \) respectively

\[ \tilde{X}_j = \text{diag}(x_j) S_{ij}^T. \]  

B. Energy Regularization

Our second optimization technique adds an energy regularization term into the loss functions during training. From (12), we can infer that the loss function of the model \( L_0 \) is a function of weights w and energy coefficient \( \rho \). The target of our optimization technique is to find the optimal energy coefficient \( \rho \) that can improve both the model accuracy and the energy efficiency. However, it is not an easy task. We prefer a smaller \( \rho \) for higher energy efficiency. However, as we see in Fig. 2, the lower \( \rho \) causes a larger fluctuation amplitude of
the weights, which results in the accuracy loss. On the other hand, if we choose a larger energy coefficient \( \rho \), the model accuracy would be less affected by the weight fluctuation, but the energy consumption becomes larger.

Our new loss function \( L \) is expressed in (14). The first term \( L_0 \) is the original loss function of the model, and the second term represents the energy consumption of the model. \( \lambda \) is a hyperparameter indicating the significance of the energy regularization term. \( a_t \) is a constant indicating the number of reading operations from the memory cell storing weight \( w_t \).

The overall loss function \( L \) can be considered as a function of \( w \) (\( w_t \) is the \( r \)th element of \( w \)) and \( \rho \), which are both trainable parameters. We can use any popular optimizer (such as SGD optimizer [56] or Adam optimizer [57]) to search for the optimal weight \( w \) and energy coefficient \( \rho \)

\[
L(w, \rho) = L_0(w, \rho) + \lambda \sum_t a_t |w_t|.
\]

During training, gradient descent will minimize the loss function \( L \). After optimization, both \( w \) and \( \rho \) will become smaller. Physically, the energy consumption of each EMT cell is proportional to the value of weight \( w \) stored in that cell (shown in Fig. 2). Hence, having small weights can help us reduce the energy consumption of the related EMT cells. We show this process in Fig. 7. With the help of the energy regularization term, we can improve both the model accuracy and energy efficiency simultaneously.

**C. Low-Fluctuation Decomposition**

The third optimization technique is to decompose the computation process into multiple time steps. We can visualize the computation involved in Fig. 8. The input activation \( x \) and the weight \( w \) equal the length of the horizontal bar and the vertical bar, respectively. The computation result got from the memory cell equals the area of the square, whose two edges have the same length as the horizontal bar and the vertical bar. In the example of original computing [Fig. 8(a)], the lengths of the horizontal bar and vertical bar are seven (\( x = 7 \)) and one (\( w = 1 \)), respectively. The area of the output square is thus seven (\( x \cdot w = 7 \)).

Theoretically, we can express any input \( x \) as a polynomial, as shown in (15). Here, the fraction bit \( \delta_p \) is binary data, which equals either 0 or 1, and \( 2^p \) is the scaling factor for that term. For example, if the input equals 7, we can decompose it into three parts: \( 1 \cdot 2^0 \), \( 1 \cdot 2^1 \), and \( 1 \cdot 2^2 \)

\[
x = \sum \delta_p \cdot 2^p.
\]

In our low-fluctuation decomposed computation mechanism [Fig. 8(b)], we read each memory cell in multiple time steps instead of once. As shown in (16), at each time step, we only input the fraction bit \( \delta_p \) to the memory cell, obtain the value of \( \delta_p \cdot w \), and then scale the output from the memory cell by the factor \( 2^p \). Finally, we sum up all the results from each time step. In this example, we use three steps to process input seven (\( x = 7 \)). The scaling factor of each time step is \( 2^0 \), \( 2^1 \), and \( 2^2 \). As the weight is one (\( w = 1 \)), the final accumulated result is seven (\( x \cdot w = 7 \)), the same result as the original computing mechanism

\[
x \cdot w = \sum (\delta_p \cdot w \cdot 2^p).
\]

As the name indicates, our low-fluctuation decomposition can alleviate the fluctuations of the memory cell effectively. We can explain this using Fig. 8, where we show the fluctuation amplitude in the yellow blocks. The block and hollow block denote positive and negative fluctuation amplitudes, respectively. As we can see from Fig. 8(b), using the decomposed computation mechanism, the negative fluctuation amplitude (hollow block) at the third time step can partially average out the positive fluctuation amplitude at the second time step (solid block). Statistically, the accumulated fluctuations from the decomposed computation mechanism have a lower standard deviation than that of the original computation mechanism.

We can mathematically compare their standard deviations. Equation (17) shows the standard deviation of the original computation mechanism, where \( O_{ori} \) is the original output and \( \sigma(\bar{w}) \) is the standard deviation of weight when we read \( \bar{w} \) from the memory cell. Equation (18) shows the standard deviation of our low-fluctuation decomposed computation mechanism, where \( O_{new} \) is the new output and \( \bar{w}(\rho) \) is the weight sampled from the memory cell at the \( p \)th time step. Since reading memory cells can be considered as independent events, we have
\[ \sigma(\tilde{w}(p)) = \sigma(\tilde{w}) \]

\[ \sigma(O_{\text{ori}}) = \sigma(x \cdot \tilde{w}) = \sum 2^p \delta_p \cdot \sigma(\tilde{w}) \]

\[ \sigma(O_{\text{new}}) = \sigma\left(\sum 2^p \delta_p \tilde{w}(p)\right) = \sqrt{\sum 2^p \delta_p^2 \cdot \sigma^2(\tilde{w}(p))} \]

\[ < \sqrt{\left(\sum 2^p \delta_p^2\right)^2 \cdot \sigma^2(\tilde{w})}. \]

We can infer from (17) and (18) that our decomposed computation result has a lower standard deviation than the original computation result, leading to high model accuracy

\[ \sigma(O_{\text{new}}) < \sigma(O_{\text{ori}}). \]

Our low-fluctuation decomposition can also improve energy efficiency since it can save a large amount of energy from the EMT cells. To prove this, we first model the energy consumption. An EMT cell can be modeled as a resistor connecting the driver’s output and the ground. Physically, we can use (20) to calculate the power consumption of the cell [58], [59]

\[ P = V_s \cdot V_d / R_T. \]

Here, \( V_s \) is the supply voltage of the driver, \( R_T \) is the resistance of the cell, and \( V_d \) is the driving voltage to the EMT cell. For in-memory computing systems, \( V_d \) is proportional to \( x \), and the input of the EMT cell, that is, \( V_d = \alpha \cdot x \). Hence, the energy consumption to drive a single EMT cell within a period \( \Delta T \) can be expressed in the following equation:

\[ E = P \cdot \Delta T = V_s \cdot V_d \cdot \Delta T / R_T = (\alpha \cdot V_s \cdot \Delta T / R_T) \cdot x. \]

Assuming \( V_s \) and \( R_T \) to be fixed and \( \beta \) equals \( \alpha \cdot V_s / \Delta T / R_T \), the relationship between the energy consumption \( E \) and the input data \( x \) can be expressed in (22). This equation can also be proved experimentally using actual hardware implementation. For example, it can be verified in various Texas Instruments’ devices [58]

\[ E = \beta \cdot x. \]

We can derive (23) based on (22). \( E(O_{\text{ori}}) \) stands for the energy consumption of the original computation mechanism, where we read once from the EMT cell with input data \( x \). \( E(O_{\text{new}}) \) represents the energy consumption of the new computation mechanism with low-fluctuation decomposition method, where we need to read the EMT cell multiple times. Each time, the input delta is \( \delta_p \), the fraction bit of \( x \) [see (15)]. Based on (23), we can mathematically prove that \( E(O_{\text{new}}) \) is smaller than \( E(O_{\text{ori}}) \) [see (24)]

\[ E(O_{\text{ori}}) = \beta \cdot x \quad E(O_{\text{new}}) = \beta \sum \delta_p \]

\[ E(O_{\text{new}}) < E(O_{\text{ori}}). \]

Besides, the low-fluctuation decomposition method can alleviate the fluctuations of EMT cells and thus improve the model accuracy. Due to this feature, we can decrease \( \rho \), the energy coefficient of EMT cells by a certain percentage, causing larger weight fluctuation [Fig. 2(b)], but still avoid the accuracy loss caused by the increased fluctuation. In this process, lowering the energy coefficient \( \rho \) can help us to save energy further [Fig. 2(a)].

This technique can be implemented in many existing neuromorphic computing devices, such as [60]. Its computation process is similar to the spiking neural network (SNN) [61]. We first read \( \delta_p \) sequentially, where \( \delta_p \) is a binary bit equalling 0 or 1. We then accumulate the partial sum over multiple time steps and shift the accumulated data by 1 bit (equivalent to multiplying by a decay factor). All the processes can be easily implemented in SNN-compatible hardware. Note that shifting the accumulated data by 1 bit for every time step is equivalent to multiplying the partial sum by \( 2^p \) [62].

### D. Convergence of the Training Method

The training images and labels can be considered as samples of all the real-world images. We do not know the distribution of them before training, but the neural networks can learn the distribution. Similarly, we do not know the distribution of weight fluctuation before training. However, the neural network can learn the distribution using gradient descent algorithms once we have the sample data: fluctuation \( S \).

We shall now mathematically prove the convergence of our training method. Equation (25) shows the basic relationship between image data \( X \) and label data \( Y \) in the traditional deep learning network. The output matrix \( Y \in \mathcal{Y} \) is a function \( f \) of the input matrix \( X \in \mathcal{X} \). Here, \( \mathcal{X} \) and \( \mathcal{Y} \), respectively, denote the space of \( X \) and \( Y \)

\[ Y = f(X). \]

For in-memory deep learning applications, the computation becomes unpredictable because of weight fluctuation. As can be seen from (12), output \( y_{ij} \) is a function of both the input \( x_i \) and the fluctuation data \( S_{ij} \). To generalize this, the output matrix \( Y \in \mathcal{Y} \) can be defined as a function \( \tilde{f} \) of the input matrix \( X \in \mathcal{X} \) and the fluctuation data \( S \in \mathcal{S} \), shown in (26). Here, \( \mathcal{S} \) denotes the space of \( S \)

\[ Y = \tilde{f}(X, S). \]

We define a new data \( \tilde{Z} \in \tilde{\mathcal{Z}} \) as the combination of images \( X \), labels \( Y \), and fluctuations \( S \). Here, the space \( \tilde{\mathcal{Z}} \) is defined as the combination of spaces \( \mathcal{X} \), \( \mathcal{Y} \), and \( \mathcal{S} \). Since \( \mathcal{Z} \) is the combination of \( \mathcal{X} \) and \( \mathcal{Y} \), we can express the space \( \tilde{\mathcal{Z}} \) as the combination of the spaces \( \mathcal{Z} \) and \( \mathcal{S} \). We show their relationships in the following equations:

\[ \tilde{Z} = (X, Y, S) \]

\[ \tilde{\mathcal{Z}} = \mathcal{X} \times \mathcal{Y} \times \mathcal{S} = \mathcal{Z} \times \mathcal{S}. \]

Given the fact that \( \mathcal{Z} \) follows the distribution \( \mathcal{D} \), while \( \mathcal{S} \) follows the distribution \( \mathcal{R} \), we can infer that the space \( \tilde{\mathcal{Z}} \) (the combination of \( \mathcal{Z} \) and \( \mathcal{S} \)) must follow a distribution \( \tilde{\mathcal{D}} \) (the combination of \( \mathcal{D} \) and \( \mathcal{R} \)). We show this in the following equation:

\[ \mathcal{Z} \sim \mathcal{D}, \quad \mathcal{S} \sim \mathcal{R} \Rightarrow \tilde{\mathcal{Z}} \sim \tilde{\mathcal{D}}. \]

Finally, our proposed training process for in-memory deep learning models can be concluded as follows: given a function \( \tilde{f} \) in the space \( \tilde{\mathcal{F}} \) and a loss function \( \mathcal{L} : \tilde{\mathcal{F}} \times \tilde{\mathcal{Z}} \rightarrow \mathbb{R}, \)
we would like to find $\tilde{f} \in \tilde{F}$ that can minimize the risk, that is, the expectation of the loss function
\[
R[\tilde{f}] = E_{z \sim \tilde{D}}[L(\tilde{f}, z)].
\] (30)

Thus, we convert the convergence problem of the training process for in-memory computing into the convergence problem for regular neural networks (see Section III-B). Finding the optimal function is well studied, and we can use various existing optimizers, such as SGD [56] or Adam [57], to find the optimal function $\tilde{f}$.

V. EXPERIMENTS

The hardware implementation of the proposed techniques is verified on a hardware-based emulator [17] and software simulator [63]. Both the emulator and the simulator can imitate the system behavior and the characteristics of EMT cells well, including the fluctuations. These platforms can sufficiently prove the feasibility and effectiveness of the optimization methods on practical in-memory computing devices. We synthesize each module and obtain metrics such as energy per operation from the report. These parameters are integrated into the simulation platform. We simulate the behavior of the system and evaluate each method’s effectiveness using the simulator [53], [63]. We trained the models on the Pytorch platform. To accelerate the training process, we start each experiment from a well-trained model with full-precision weights [64] and then fine-tune the model by applying our proposed optimizations. During fine-tuning, we quantize both the activations and weights [14].

To form the device-enhanced dataset, we fetch the images and labels from the regular datasets (such as CIFAR-10 and ImageNet) as data $X$ and data $Y$, respectively. We do not design the fluctuation but sample the fluctuation $S$ from the EMT cells using exact device models [16], [20], [63], [65], [66], which reflect the hardware mechanism’s real characteristics, including device nonideality and noises/errors. We use a workstation with an Nvidia 2080TI graphic card to train the model. For the CIFAR-10/ImageNet dataset, each experiment can finish in about 1 h/an entire day.

We proposed three techniques, denoted as “A,” “B,” and “C.” As shown in Fig. 4; the notations A, B, and C stand for the device-enhanced dataset, energy regularization, and

### TABLE II

|                | 0% accuracy drop | 1% accuracy drop | 2% accuracy drop |
|----------------|------------------|------------------|------------------|
|                | Energy ($\mu J$) | #Cells | Delay ($\mu S$) | Energy ($\mu J$) | #Cells | Delay ($\mu S$) | Energy ($\mu J$) | #Cells | Delay ($\mu S$) |
| VGG-16 (93.6% Acc.) |                |        |               |                |        |               |                |        |               |
| Binarized Encoding | 378 | 74M | 2.8 | 135 | 74M | 2.8 | 94 | 74M | 2.8 |
| Weight Scaling | 444 | 15M | 2.8 | 78 | 15M | 2.8 | 49 | 15M | 2.8 |
| Fluctuation Compensation | 1091 | 15M | 14 | 157 | 15M | 14 | 82 | 15M | 14 |
| Ours (A+B) | 36 | 15M | 2.8 | 16 | 15M | 2.8 | 11 | 15M | 2.8 |
| Ours (A+B+C) | 4.1 | 15M | 14 | 1.0 | 15M | 14 | 0.5 | 15M | 14 |

### TABLE III

|                | 0% accuracy drop | 1% accuracy drop | 2% accuracy drop |
|----------------|------------------|------------------|------------------|
|                | Energy ($\mu J$) | #Cells | Delay ($\mu S$) | Energy ($\mu J$) | #Cells | Delay ($\mu S$) | Energy ($\mu J$) | #Cells | Delay ($\mu S$) |
| ResNet-18 (95.2% Acc.) |                |        |               |                |        |               |                |        |               |
| Binarized Encoding | 876 | 56M | 6.8 | 389 | 56M | 6.8 | 286 | 56M | 6.8 |
| Weight Scaling | 1127 | 11M | 6.8 | 209 | 11M | 6.8 | 158 | 11M | 6.8 |
| Fluctuation Compensation | 2217 | 11M | 34 | 474 | 11M | 34 | 347 | 11M | 34 |
| Ours (A+B) | 83 | 11M | 6.8 | 22 | 11M | 6.8 | 10 | 11M | 6.8 |
| Ours (A+B+C) | 6.9 | 11M | 34 | 1.1 | 11M | 34 | 0.7 | 11M | 34 |

Authorized licensed use limited to the terms of the applicable license agreement with IEEE. Restrictions apply.
This observation can prove the near-orthogonality of our three techniques. We can draw a similar conclusion for solution $A + B$.

Models optimized by our proposed methods have much higher accuracy than the model trained by the traditional optimizer. In Fig. 10, we show the accuracy achieved by solution $A$, solution $A + B$, and solution $A + B + C$ under different energy budgets. As a reference, we also give the model accuracy trained by the traditional optimizer. As we can see from the figure, at $16 \mu J$ energy budget, the accuracy of solution $A + B + C$ is very close to the SOTA accuracy that the respective model can achieve (shown as dashed lines in the top subfigures). On the other hand, the traditional optimizer exhibits relatively low accuracy due to its unawareness of memory fluctuation.

The energy budget limits the maximum energy the computing system can use to process an image. According to Fig. 2(a), if we tighten the energy budget, we must tune the EMT cell physically with a smaller energy coefficient $\rho$. Referring to Fig. 2(b), a smaller energy coefficient $\rho$ will cause a more significant weight fluctuation, resulting in a larger accuracy loss. From Fig. 2, we can see that when the energy budget is decreased, models trained using the traditional optimizer show a dramatic decrease in accuracy. On the contrary, our solution $A + B + C$ can achieve high model accuracy even if we reduce the energy budget. Even just using Solutions $A$ and $A + B$ is enough to maintain a relatively high accuracy, only to be outperformed by Solution $A + B + C$. This observation further proves the effectiveness of our proposed three techniques on in-memory computing.

We can also see that under $16 \mu J$ energy consumption, the ResNet-18 trained by the traditional optimizer shows much lower accuracy than the VGG-16. By using solution $A + B + C$, ResNet-18 can fully recover the accuracy and thus outperforms VGG-16. This experiment also shows that MobileNet is not suitable for in-memory deep learning. Under the same energy budget, MobileNet shows lower accuracy than VGG-16 and ResNet-18. We attribute this to its depthwise layer. When we compute a regular convolution layer, the system reads hundreds of EMT cells at once. However, to process the depthwise layer, it only read nine EMT cells at once. Therefore, a large portion of the energy is consumed in the peripheral circuits, causing a significant amount of energy overhead.

### A. Ablation Study of Proposed Techniques

In Fig. 9, we compare the energy reduction of the three techniques under the same accuracy loss (0%). Fig. 9(a) lists the normalized energy consumption, and Fig. 9(b) shows the respective reduction ratio. Compared with the baseline settings, the three techniques can reduce energy consumption by around $7.1 \times$, $2.7 \times$, and $10.9 \times$, respectively. Technique $C$ can make the most significant contribution but may also cause longer latency. We leave the trade-off between energy consumption and latency to hardware developers and propose two solutions: $A + B$ and $A + B + C$. In Fig. 9(c), we plot solution $A + B$’s and $A + B + C$’s energy reduction ratio on the log scale axis. We also directly accumulate the reduction ratio from every technique in the first row as a reference. Fig. 9(c) shows that the reduction ratio of solution $A + B + C$ is very close to the product of A’s, B’s, and C’s ratios (shown as the three stacking blocks). This observation can prove the

---

**Fig. 9.** (a) Normalized energy consumption to apply only one technique. (b) Respective energy reduction ratio. (c) Energy reduction ratio of $A + B$ and $A + B + C$, in log scale. All the methods are applied to the baseline settings in [20]. We test the VGG-16 network on the CIFAR-10 dataset.

---

low-fluctuation decomposition, respectively. Solution $A$ uses only the first technique, solution $A + B$ applies the first two, and solution $A + B + C$ combines all three. We evaluate popular models, including VGG-16, ResNet-18/34, and MobileNet. VGG-16 is a regular deep neural network with only $3 \times 3$ kernels. ResNet-18/34 are popular models that achieve competitive accuracy by adding residual links between layers. MobileNet is a small-size model that achieves high efficiency due to its special depthwise layer. We compare our work with SOTA solutions including binarized encoding [19], weight scaling [20], fluctuation compensation [21], model compression (pruning) [22], and knowledge distillation [23] as described in detail in Section II. We implement our techniques and SOTA solutions on the same platform [53], [63] with the same conditions.

### B. Robustness to Different Devices

Today, academia and industry have developed various types of EMT cells, which have different fluctuation patterns with different levels of fluctuation intensity. Hence, it is necessary to prove the robustness of our solutions under different fluctuations. In Fig. 11, we test our solutions and the SOTA under three fluctuation patterns with three typical levels of fluctuation intensities [67]: weak, normal, and strong. The experiment is conducted on the ImageNet dataset using two ResNet models. All solutions, including the SOTA, are free to tune the energy coefficient $\rho$. We compare the energy consumption when the model achieves its maximum accuracy. Note that on the ImageNet dataset, our solutions can reach the SOTA accuracy of the respective model, where the SOTA solutions cannot.
Fig. 10. Comparison between our proposed optimization solutions and the traditional optimizer. We test models on the CIFAR-10 dataset. The first row of subfigures shows the accuracies in a zoomed range, and the second row of subfigures shows accuracies in the full range.

Fig. 11. Energy comparison between our proposed solutions and the SOTA, under three levels of fluctuation intensity. We test models on the ImageNet dataset. Both our solutions and the SOTA are free to tune the energy coefficient $\rho$.

Fig. 12. Accuracy comparison between our proposed solutions and the SOTA. We test models on the ImageNet dataset. The solid bar and hollow bar denote the top-1 accuracy and top-5 accuracy, respectively. The dashed line denotes the SOTA accuracy of the respective model.

The results show the robustness of our solutions. At any level of fluctuation intensity, our solution has almost the same performance on energy reduction. When fluctuation intensity is increased, both ours and the SOTA solutions prefer a higher energy coefficient $\rho$ to maximize the model accuracy, resulting in higher energy consumption. However, our solutions still
Fig. 13: Robustness of our optimization algorithm with method: (a) A + B (device-enhanced dataset + energy regularization) and (b) A + B + C (A + B + low-fluctuation decomposition). The fluctuation model covers different types of nonlinear characteristics of the emerging memory device that makes the computation results unstable, including device nonideality [65] and noises/errors (RTN, thermal noise, shot noise, programming error, and so on) [66]. We fix the strength of RTN noise and scale up the standard deviation of the other forms of noise up to 25% of the signal strength.

C. Verification of the Optimization Solutions

In Fig. 12, we verified our optimization solution by testing two ResNet models on the ImageNet dataset. The dashed line in Fig. 12 shows the SOTA accuracy. We defined it as the maximum accuracy that the respective model can achieve on graphics processing unit (GPU). We also list the accuracy of the SOTA for comparison. Among all, our solution A + B又有 the highest top-1 and top-5 accuracies, which are the same as the SOTA accuracies. The accuracy of solution A + B also shows higher accuracy than previous methods. We can observe a small accuracy loss under a smaller energy budget. In contrast, models optimized by the SOTA have significant accuracy losses. For example, we can see at least 0.9% and 0.8% top-1 accuracy losses of the ResNet-18 and ResNet-34, respectively. We can also see that ResNet-18 on the ImageNet consumes more energy than CIFAR-10, mainly because of the larger image size.

D. Robustness of the Optimization Algorithms

The results from validating the robustness of our optimization method are shown in Fig. 13. The fluctuation model covers different types of nonlinear characteristics of the emerging memory device that negatively impact the computation’s results, including device nonideality [65] and noises/errors (random telegraph noise (RTN), thermal noise, shot noise, programming error, and so on) [66].

In the ideal situation, emerging memory devices can execute the multiplication operations based on Ohm’s law. However, the device’s nonideality will turn the input activation x into $\sinh(\gamma \cdot x)/\gamma$ [65], making the result inaccurate, where $\gamma$ is a parameter related to the supply voltages of the memory device. To explore the robustness of our algorithm to different strengths of nonideality, we evaluate three values of $\gamma$: 0.5, 1, and 2, which correspond to three typical supply voltages: 0.3, 0.6, and 1.2 V, respectively, giving a fitting voltage $v_0$ of 0.6 V [68]. A larger value of $\gamma$ corresponds to greater nonideality. We also compare the ideal case, assuming that a specific input circuit module can compensate for the data-dependent nonideality.

In addition to device nonideality, noise and error will also cause fluctuation in the output signals, thus leading to model inaccuracy. This experiment includes four main sources of noise and error: RTN, thermal noise, shot noise, and programming error [66]. The latter three are modeled as random variables with a mean value of zero, so we can evaluate their strength by the standard deviation. In most cases, RTN noise dominates. The other types of noise (thermal noise, shot noise, and programming error) in total account for less than 5% of the signal strength [66]. Previous experiments have proved the effectiveness of our algorithm in this common scenario. To validate the robustness of our algorithms under a stronger nonlinear noise effect, we fix the strength of RTN noises and scale up the standard deviation of the other types of noise to up to 25% of the signal strength.

From Fig. 13, we can see that our method can recover the model’s accuracy well, even under large nonideality and noise. The key reason is that our method can help the neural network better learn the nonlinear characteristics of the devices and thus compensate for the computation error. Note that technique C can avoid data-dependent nonideality. Using technique C, the data input is either 1 or 0. Since the input amplitude is...
fixed, we can precisely compensate for the data-dependent nonideality using a fixed compensation factor.

E. Holistic Comparison With the SOTA

Our proposed solutions have better performance not only on energy reduction but also on cost and latencies. In Tables II and III, we give a holistic comparison of our solutions with the SOTA on accuracy and energy efficiency. The indices (x) of knowledge distillation and model compression (pruning) are given as follows: (a) IE [22], (b) FPGM [42], (c) DMC [43], (d) NISP [44], (e) PFEC [45], (f) SFP [46], (g) HSADK [23], (h) ICXD [47], (i) WSL [48], (j) CRCD [49], (k) LSHFM [50], and (l) DKD [51].

![Fig. 14. Comparison between our proposed solutions and the SOTA in terms of accuracy and energy efficiency. The larger, the better. We test the ResNet-34 model on the ImageNet dataset.](image)

VI. CONCLUSION

In-memory deep learning has a promising future in the AI industry because of its high energy efficiency over traditional deep learning. This is even more so if the potential of EMT, especially in analog computing mode, is used. In the analog mode, we use only one cell to store one parameter in the model, pursuing high energy and area efficiency. Unfortunately, one of the major limitations of EMT is the intrinsic instability of EMT cells, which can cause a significant loss in accuracy. On the other hand, falling back on a digital mode of operation will erode the potential gains. In this work, we propose three optimization techniques that can fully recover the accuracy of EMT-based analog in-memory deep learning models while minimizing their energy consumption. They include the device-enhanced dataset, energy regularization, and low-fluctuation decomposition. Based on the experiment results, we offer two solutions. Developers can either apply the first two optimization techniques or apply all three to the target model. Both solutions can achieve higher accuracy than their SOTA counterparts. The first solution shows at least one order of magnitude improvement in energy efficiency, with the least hardware cost and latency. The second solution further improves energy efficiency by another order of magnitude at the cost of higher latency.

REFERENCES

[1] (Apr. 2018). GPU-Based Deep Learning Inference: A Performance and Power Analysis. [Online]. Available: https://www.nvidia.com
[2] T. Yang, Y. Chen, and V. Sze, “Designing energy-efficient convolutional neural networks using energy-aware pruning,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), Jul. 2017, pp. 6071–6079.

[3] J. Meena, S. Sze, U. Chand, and T.-Y. Tseng, “Overview of emerging non-volatile memory technologies,” Nanos. Res. Lett., vol. 9, pp. 1–33, Sep. 2014.

[4] P. Yao et al., “Fully hardware-implemented memristor convolutional neural network,” Nature, vol. 577, no. 7792, pp. 641–646, Jan. 2020.

[5] F. Wu et al., “Fully integrated non-programmable memristor-CMOS system for efficient multiply-accumulate operations,” Nature Electron., vol. 2, no. 7, pp. 290–299, Jul. 2019.

[6] T. Luo et al., “NC-Net: Efficient neuromorphic computing using aggregated subnets on a crossbar-based architecture with nonvolatile memory,” IEEE Trans. Comput.-Aided Design Integrate. Circuits Syst., vol. 41, no. 9, pp. 2957–2969, Sep. 2022.

[7] A. Sebastian, M. Le Gallo, R. Khaddam-Aljameh, and E. Eleftheriou, “Experimental study of artificial neural networks using a digital memristor simulator,” IEEE Trans. Neural Netw. Learn. Syst., vol. 29, no. 10, pp. 5098–5110, Oct. 2018.

[8] S. Yang et al., “Scalable digital neuromorphic architecture for large-scale biophysically meaningful neural network with multi-compartment neurons,” IEEE Trans. Neural Netw. Learn. Syst., vol. 31, no. 1, pp. 148–162, Jan. 2020.

[9] O. Krestinskaya, A. P. James, and L. O. Chua, “Neuromemristive circuits for edge computing: A review,” IEEE Trans. Neural Netw. Learn. Syst., vol. 31, no. 1, pp. 4–23, Jan. 2020.

[10] F. Merrikh-Bayat, X. Guo, M. Klachko, M. Prezioso, K. K. Likharev, and D. B. Strukov, “High-performance mixed-signal neuromcomputing with nanoscale floating-gate memory cell arrays,” IEEE Trans. Neural Netw. Learn. Syst., vol. 29, no. 10, pp. 4782–4790, Oct. 2018.

[11] M. T. Li, Y. Shen, Q. Wu, G. L. Z. Lerman, and B. Yang, “Memristor crossbar-based neuromorphic computing system: A case study,” IEEE Trans. Neural Netw. Learn. Syst., vol. 25, no. 10, pp. 1864–1878, Oct. 2014.

[12] A. K. Maan, D. A. Jayadevi, and A. P. James, “A survey of memristive threshold logic circuits,” IEEE Trans. Neural Netw. Learn. Syst., vol. 28, no. 8, pp. 1734–1746, Aug. 2017.

[13] A. R. K. G. L. Alajameh, and E. Eleftheriou, “Memory devices and applications for in-memory computing,” Nature Nanotechnol., vol. 15, no. 7, pp. 529–544, Jul. 2020.

[14] S. Stathopoulos et al., “Multitbit memory operation of metal-oxide bi-layer memristors,” Sci. Rep., vol. 7, no. 1, p. 1–7, Dec. 2017.

[15] M. Demler, “Mythic multiplies in a flash,” Microprocess. Rep., Aug. 2018. [Online]. Available: https://www.mythic-ai.com/wp-content/uploads/2018/08/Mythic-Multiplies-In-A-Flash.pdf

[16] N. Raghavan et al., “RTN insight to filamentary instability and disturb immunity in ultra-low power switching HfO2 content,” Nature, vol. 577, no. 8, pp. 1734–1746, Aug. 2017.

[17] H. Zhou et al., “Rethinking soft labels for knowledge distillation: A bias–variance tradeoff perspective,” in Proc. Int. Conf. Learn. Represent., (ICLR) 2021, pp. 4335–4344.

[18] Z. Wang, T. Luo, M. Li, J. T. Zhou, R. S. M. Goh, and L. Zhen, “Evolutionary multi-objective model compression for deep neural networks,” IEEE Comput. Intell. Mag., vol. 16, no. 3, pp. 10–21, Aug. 2021.

[19] J. Gou, B. Yu, S. J. Maybank, and D. Tao, “Knowledge distillation: A survey,” Int. J. Comput. Vis., vol. 129, no. 6, pp. 1789–1819, Jun. 2021.

[20] Y. He, P. Liu, Z. Wang, Z. Hu, and Y. Yang, “Filter pruning via geometric median for deep convolutional neural networks acceleration,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR), Jun. 2019, pp. 4335–4344.

[21] W. Chen et al., “A 65nm 1Mb nonvolatile computing-in-memory ReRAM macro with sub-16ns multiply-and-accumulate for binary DNN AI edge processors,” in Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers, Feb. 2018, pp. 494–496.

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

[23] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, “XNOR-Net: Imagenet classification using binary convolutional neural networks,” in Proc. Eur. Conf. Comput. Vis. Cham, Switzerland: Springer, 2016, pp. 525–542.

[24] X. Peng, R. Liu, and S. Yu, “Optimizing weight mapping and data flow for convolutional neural networks on ReRAM based Processing-In-Memory architecture,” in Proc. IEEE Int. Symp. Circuits Syst. (ISCAS), May 2019, pp. 1–5.

[25] Z. He, J. Lin, R. Ewertz, J. Yuan, and D. Fan, “Noise injection adaption: End-to-end ReRAM crossbar non-ideal effect adaption for neural network mapping,” in Proc. 56th ACM/IEEE Design Autom. Conf. (DAC), Jun. 2019, pp. 1–6.

[26] S. Han, H. Mao, and W. J. Dally, “Deep compression: Compressing deep neural network with pruning, trained quantization and Huffman coding,” in Proc. 4th Int. Conf. Learn. Represent., (ICLR) 2016, pp. 1–5.

[27] G. L. Zhang et al., “Reliable and robust ReRAM-based neuromorphic computing,” in Proc. Great Lakes Symp. (VLIS), Sep. 2020, pp. 33–38.

[28] D. Jokas et al., “Committee machines—A universal method to deal with non-idealities in memristor-based neural networks,” Nature Commun., vol. 11, no. 1, pp. 1–10, Aug. 2020.

[29] S. Han, H. Mao, and W. J. Dally, “Deep compression: Compressing deep neural network with pruning, trained quantization and Huffman coding,” in Proc. 4th Int. Conf. Learn. Represent., (ICLR) San Juan, Puerto Rico, May 2016.

[30] Z. Wang, T. Luo, M. Li, J. T. Zhou, R. S. M. Goh, and L. Zhen, “Evolutionary multi-objective model compression for deep neural networks,” IEEE Comput. Intell. Mag., vol. 16, no. 3, pp. 10–21, Aug. 2021.

[31] J. Gou, B. Yu, S. J. Maybank, and D. Tao, “Knowledge distillation: A survey,” Int. J. Comput. Vis., vol. 129, no. 6, pp. 1789–1819, Jun. 2021.

[32] Y. He, P. Liu, Z. Wang, Z. Hu, and Y. Yang, “Filter pruning via geometric median for deep convolutional neural networks acceleration,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR), Jun. 2019, pp. 4335–4344.

[33] S. Gao, F. Huang, J. Pei, and H. Huang, “Discrete model compression with resource constraint for deep neural networks,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR), Jun. 2020, pp. 1899–1908.

[34] Y. He et al., “NISP: Pruning networks using neuron importance score propagation,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., Jun. 2018, pp. 9194–9203.

[35] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, “Pruning filters for efficient ConvNets,” in Proc. Int. Conf. Learn. Represent., 2017.

[36] Y. He, G. Kang, X. Dong, Y. Fu, and Y. Yang, “Soft filter pruning for accelerating deep convolutional neural networks,” in Proc. 27th Int. Joint Conf. Artif. Intell., Jul. 2018, pp. 2234–2240.

[37] H. Li, Y. He, P. Liu, Z. Wang, Z. Hu, and Y. Yang, “Filter pruning via geometric median for deep convolutional neural networks acceleration,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR), Jun. 2019, pp. 4335–4344.

[38] J. Zhu et al., “Complementary relation contrastive distillation,” in Proc. ICLR, 2020, pp. 899–908.

[39] X. Peng, R. Liu, and S. Yu, “Optimizing weight mapping and data flow for convolutional neural networks on ReRAM based Processing-In-Memory architecture,” in Proc. IEEE Int. Symp. Circuits Syst. (ISCAS), May 2019, pp. 1–5.

[40] Z. He, J. Lin, R. Ewertz, J. Yuan, and D. Fan, “Noise injection adaption: End-to-end ReRAM crossbar non-ideal effect adaption for neural network mapping,” in Proc. 56th ACM/IEEE Design Autom. Conf. (DAC), Jun. 2019, pp. 1–6.
Zehui Wang received the B.S. degree in electrical engineering from Fudan University, Shanghai, China, in 2010, and the Ph.D. degree in electronic and computer engineering from The Hong Kong University of Science and Technology, Hong Kong, in 2016. He is currently a Research Scientist with the Institute of High Performance Computing, Agency for Science, Technology and Research, Singapore. His research interests include efficient artificial intelligence (AI) deployment, AI on emerging technologies, hardware–software coexploration, and high-performance computing.

Tao Luo received the bachelor’s degree from the Harbin Institute of Technology, Harbin, China, in 2010, the master’s degree from the University of Electronic Science and Technology of China, Chengdu, China, in 2013, and the Ph.D. degree from the School of Computer Science and Engineering, Nanyang Technological University, Singapore, in 2018. He is currently a Research Scientist with the Institute of High Performance Computing (IHPC), Agency for Science, Technology and Research (A*STAR), Singapore. His current research interests include high-performance computing, green artificial intelligence (AI), quantum computing, software–hardware coexploration, and efficient AI and its application.

Rick Siow Mong Goh received the Ph.D. degree in electrical and computer engineering from the National University of Singapore, Singapore, in 2006. He is currently the Director of the Computing and Intelligence (CI) Department, Institute of High Performance Computing, Agency for Science, Technology and Research, Singapore, where he leads a team of over 80 scientists in performing world-leading scientific research, developing technology to commercialization, and engaging and collaborating with industry. His current research interests include artificial intelligence, high-performance computing, blockchain, and federated learning.

Wei Zhang was an Assistant Professor with the School of Computer Engineering, Nanyang Technological University, Singapore, from 2010 to 2013. She is currently a Full Professor with The Hong Kong University of Science and Technology, Hong Kong, where she established the Reconfigurable System Laboratory. She has authored and coauthored more than 150 papers in peer-reviewed journals and international conferences. Her current research interests include reconfigurable system and field-programmable gate array (FPGA)-based design, low-power high-performance multicore systems, and embedded systems.

Weng-Fai Wong (Senior Member, IEEE) received the B.Sc. degree from the National University of Singapore, Singapore, in 1988, and the Dr.Eng.Sc. degree from the University of Tsukuba, Tsukuba, Japan, in 1993. He is currently an Associate Professor at the Department of Computer Science, National University of Singapore, Singapore. His research interests include computer architecture, compilers, and high-performance computing.