CRIMP: Compact & Reliable DNN Inference on In-Memory Processing via Crossbar-Aligned Compression and Non-ideality Adaptation

SHUO HUAI, HAO KONG, XIANGZHONG LUO, and SHIQING LI, Nanyang Technological University, Singapore
RAVI SUBRAMANIAM, CHRISTIAN MAKAYA, and QIAN LIN, HP Inc., United States
WEICHEN LIU, Nanyang Technological University, Singapore

Crossbar-based In-Memory Processing (IMP) accelerators have been widely adopted to achieve high-speed and low-power computing, especially for deep neural network (DNN) models with numerous weights and high computational complexity. However, the floating-point (FP) arithmetic is not compatible with crossbar architectures. Also, redundant weights of current DNN models occupy too many crossbars, limiting the efficiency of crossbar accelerators. Meanwhile, due to the inherent non-ideal behavior of crossbar devices, like write variations, pre-trained DNN models suffer from accuracy degradation when it is deployed on a crossbar-based IMP accelerator for inference. Although some approaches are proposed to address these issues, they often fail to consider the interaction among these issues, and introduce significant hardware overhead for solving each issue. To deploy complex models on IMP accelerators, we should compact the model and mitigate the influence of device non-ideal behaviors without introducing significant overhead from each technique.

In this paper, we first propose to reuse bit-shift units in crossbars for approximately multiplying scaling factors in our quantization scheme to avoid using FP processors. Second, we propose to apply kernel-group pruning and crossbar pruning to eliminate the hardware units for data aligning. We also design a zerorize-recover training process for our pruning method to achieve higher accuracy. Third, we adopt the runtime-aware non-ideality adaptation with a self-compensation scheme to relieve the impact of non-ideality by exploiting the feature of crossbars. Finally, we integrate these three optimization procedures into one training process to form a comprehensive learning framework for co-optimization, which can achieve higher accuracy. The experimental results indicate that our comprehensive learning framework can obtain significant improvements over the original model when inferring on the crossbar-based IMP accelerator, with an average reduction of computing power and computing area by 100.02× and 17.37×, respectively. Furthermore, we can obtain totally integer-only, pruned, and reliable VGG-16 and ResNet-56 models for the Cifar-10 dataset on IMP accelerators, with accuracy drops of only 2.19% and 1.26%, respectively, without any hardware overhead.

This article appears as part of the ESWEEK-TECS special issue and was presented in the International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS), 2023.
This work is partially supported under the RIE2020 Industry Alignment Fund – Industry Collaboration Projects (IAF-ICP) Funding Initiative, as well as cash and in-kind contribution from the industry partner, HP Inc., through the HP-NTU Digital Manufacturing Corporate Lab (I1801E0028), and partially supported by Nanyang Technological University, Singapore, under its NAP (M4082282/04INS000515C130).
Authors' addresses: S. Huai, H. Kong, X. Luo, S. Li, and W. Liu (corresponding author), School of Computer Science and Engineering, Nanyang Technological University, Singapore; emails: {shuo.huai, hao.kong, xiangzho001, shiqing.li, liu}@ntu.edu.sg; R. Subramaniam, C. Makaya, and Q. Lin, HP Inc., Palo Alto, California, United States; emails: {ravi.subramaniam, christian.makaya, qian.lin}@hp.com.

1539-9087/2023/09-ART123 $15.00
https://doi.org/10.1145/3609115

CCS Concepts: • **Computing methodologies** → *Neural networks*; • **Hardware** → *Emerging technologies;*

Additional Key Words and Phrases: In-memory processing, pruning, quantization, ReRAM non-ideality

ACM Reference format:
Shuo Huai, Hao Kong, Xiangzhong Luo, Shiqing Li, Ravi Subramaniam, Christian Makaya, Qian Lin, and Weichen Liu. 2023. CRIMP: Compact & Reliable DNN Inference on In-Memory Processing via Crossbar-Aligned Compression and Non-ideality Adaptation. *ACM Trans. Embedd. Comput. Syst.* 22, 5s, Article 123 (September 2023), 25 pages.
https://doi.org/10.1145/3609115

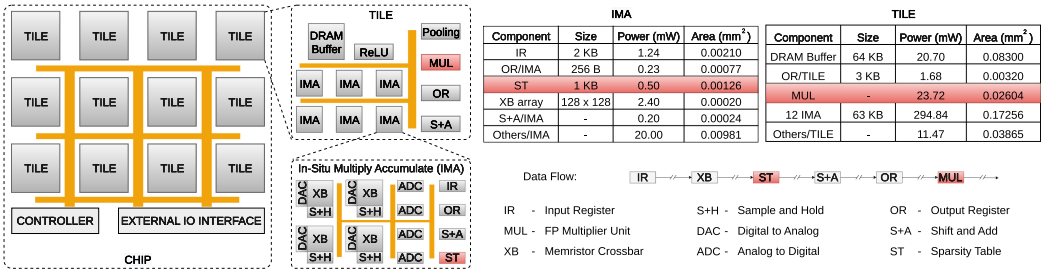
1 INTRODUCTION

Deep Neural Networks (DNNs) have advanced many fields, including image recognition and natural language processing. Meanwhile, the demand for low-power edge intelligence is rapidly increasing in everyday devices like mobile phones and wearable gadgets [23]. However, the efficient deployment of DNNs on low-power devices is hindered by their growing need for computational ability and memory resources [16]. DNN algorithms exhibit a high degree of computing parallelism but require large memory access, thus, the Resistive Random Access Memory (ReRAM)-based In-Memory Processing (IMP) crossbar architecture is an emerging and promising solution for efficiently accelerating these DNN algorithms. ReRAM crossbar architectures exploit parallel digital arithmetic units and perform computations within the memory itself, thereby maximizing parallel execution and minimizing memory access [42]. Despite the advantages of IMP crossbar architectures, current DNN models are still challenging to deploy efficiently and accurately on IMP devices due to the floating-point (FP) arithmetic [12], redundant weights [17] and ReRAM non-idealities [22].

ReRAM crossbar architectures have limited write endurance and high reconfigure latency [46]. It is crucial to minimize the number of writing operations to reduce configuration latency and extend the life of the crossbar. Therefore, during the model inference stage on ReRAM-based IMP architectures, it is impossible to dynamically reconfigure the crossbar to infer different layers or parts of a DNN model. But loading the entire model can result in using more crossbars than the IMP device owns. IMP-aware fine-grained pruning methods [8, 27, 30], which trim columns/rows of weights in each crossbar, can reduce the number of required crossbars. However, these methods necessitate expensive extra hardware (i.e., *ST* in Figure 1) to align the output/input of each crossbar [8], which severely decreases the hardware integration density and reduces the number of crossbars that can be embedded on IMP devices. They can not achieve the desired resource savings.

Meanwhile, implementing FP arithmetic units on IMP crossbars is complicated and expensive [12], thus, the crossbar architectures cannot directly perform FP multiplications. This means extra FP processors are required in the IMP device for DNN models with FP operations, which introduce a power overhead of approximately 7% and an area overhead of about 9% into the IMP device, as shown in Figure 1. To achieve low-power DNNs for IMP devices, the final DNN model should only comprise integer operations. Although traditional model quantization schemes [7] can approximate FP inputs and weights with integers, they still employ FP scaling factors to maintain accuracy. IAO [19] proposes replacing FP scaling factors with fixed-point multiplication, but this approach still requires a fixed-point multiplier, which also introduces some overhead.

Furthermore, due to the inherent features of ReRAM cells, it is subject to some non-idealities, such as Write Variation [15] and Stuck-At-Faults (SAFs) [4]. These non-idealities can significantly reduce the accuracy of model inference achieved by ReRAM-based IMP accelerators [3]. There are some strategies to address this issue. However, these techniques either necessitate more write



IMA

Component	Size	Power (mW)	Area (mm^2)
IR	2 KB	1.24	0.00210
OR/IMA	256 B	0.23	0.00077
ST	1 KB	0.50	0.00126
XB array	128 x 128	2.40	0.00020
S+A/IMA	-	0.20	0.00024
Others/IMA	-	20.00	0.00981

TILE

Component	Size	Power (mW)	Area (mm^2)
DRAM Buffer	64 KB	20.70	0.08300
OR/TILE	3 KB	1.68	0.00320
MUL	-	23.72	0.02604
12 IMA	63 KB	294.84	0.17256
Others/TILE	-	11.47	0.03865

Fig. 1. IMP architecture from ISAAC [42] with the parameter of each component. Please note that *ST* is used for crossbar-column/row IMP-aware pruning, and *MUL* is used for processing FP multiplications.

operations to the ReRAM cells [3], thereby reducing their endurance; or employ additional hardware units to compensate for non-ideality [13, 36], thereby increasing hardware overhead; or omit the impact of crossbar features (e.g., crossbar size, Analog-to-Digital converters, etc.) on the non-ideality [11, 33], thereby introducing more inaccuracies.

To address all the aforementioned problems, in this paper, we propose a novel DNN learning framework, named *CRIMP*, that can create compact and reliable DNN models with high accuracy for IMP inference. This framework integrates integer-only quantization, crossbar-aligned pruning, and runtime-aware non-ideality adaptation schemes into one training process that is executed on a GPU server, and the trained model can be directly deployed on the IMP device for inference. Our main contributions can be summarized as follows.

(1) We propose using kernel-group pruning and crossbar pruning to reduce crossbar usage without extra processing units for data aligning (i.e., crossbar-aligned pruning). For high accuracy and sparsity, we incorporate these two pruning methods to establish a multi-grained pruning. To further improve the accuracy of these two coarse-grained pruning methods, we introduce a dynamic zerorize-recover training procedure, which broadens the exploration space to discover a better pruned architecture and superior weights.
(2) We adopt a runtime-aware non-ideality adaptation scheme to learn reliable DNN models for IMP accelerators. This scheme is guided by our realistic crossbar-based runtime simulator, which accounts for various non-idealities of ReRAM cells and crossbar features. In addition, we introduce a self-compensation scheme to diminish the error from the non-ideality. It only leverages the used crossbar cells themselves to reduce the error by mutual calibration, but it also maintains the flexibility to leverage more cells to further reduce the error.
(3) We design a simple yet efficient integer-only quantization scheme tailored for the IMP crossbar architecture, achieved by reusing the bit-shift units. We also introduce a comprehensive model learning framework that combines crossbar-aligned pruning, integer-only quantization, and runtime-aware non-ideality adaptation into a single training process. This framework co-optimizes these techniques to enhance accuracy.
(4) We demonstrate the effectiveness of *CRIMP* with extensive experiments. Specifically, our quantization method does not introduce much accuracy drop compared to traditional quantization methods. And our crossbar-aligned pruning achieves a higher sparsity rate and a slighter accuracy drop without extra hardware, compared to state-of-the-art IMP-aware pruning methods. Moreover, *CRIMP* significantly reduces computing power and area, while also producing totally compact and reliable models with only a slight accuracy drop.

The rest of this paper is structured as follows. In Section 2, we provide an overview of the background and related works on IMP-aware DNN pruning, quantization, and ReRAM non-ideality

Table 1. The Symbols and their Definitions used in the Equations or Algorithms of this Paper

Symbol	Definition	Symbol	Definition
r	Ideal resistance value	W	The weights of a layer
r'	Actual resistance value	B	The biases of a layer
θ	Distribution of the error	S	The scaling factor
e	Euler's number	Q	The quantized value
ϵ	Standard deviation of the error	Z	The zero point
N	Total number or Normal distribution	$\circledast$	The convolutional operation
C	The number of channels/kernels	∂	Partial derivative
C'	The number of remaining kernels	D, d	The dimension (length) of vector
K	The size of kernels	δ	The importance factor of weights
X	The input of a layer	XB_s	The size of crossbar
γ	The weights in BN layer	s	The epoch to start zerorize-recover
lr	Learning rate	c	Ideal conductance (weight) value
XB	The abbreviation of crossbar	c'	Actual conductance (weight) value
H, h	The height of crossbar	q	The number of quantization bits
w	The width of crossbar	p	The pruning ratio
L, l	Layer index in DNN	σ	The activation function
n, a, b, m	Any number	z	Updated value in a training step
v	Voltage value, each bit of the input	t	Training step

processing. Section 3 elaborates on the details of our proposed approaches. Section 4 presents the experimental results. In Section 5, we conclude this paper.

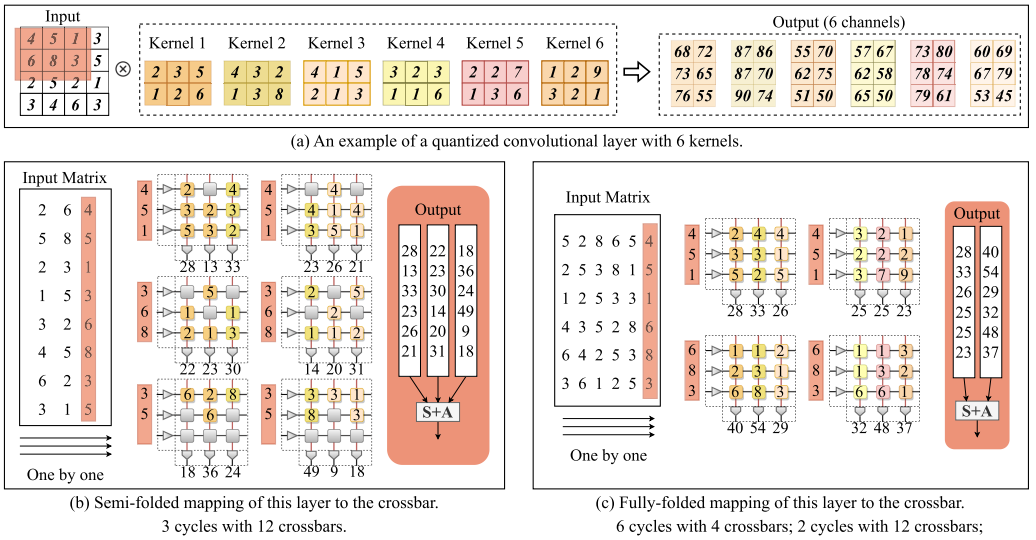
2 BACKGROUND & RELATED WORKS

In this section, we provide a brief on the fundamentals of IMP, as well as an overview of IMP-aware pruning, quantization, and non-ideality processing. Additionally, we discuss the advantages and disadvantages of some related works in this field, establishing motivation for our proposed approach. Table 1 shows the symbols and definitions used in this paper.

2.1 ReRAM Crossbar-Based IMP Architecture & DNN Mapping Scheme

The ReRAM-based IMP device is composed of numerous crossbars, with memristive cells situated between each horizontal wordline and vertical bitline in a crossbar [42]. When inference DNNs, the weights of a model are mapped to the memristive cells, and the conductance represents the associated weight values. And the digital-to-analog converter (DAC) converts the input to voltage pulses that are injected into the wordline of the crossbar. According to Kirchoff's Law, the current generated in each cell, representing the product between the voltage and the cell conductance, accumulates along the bitline, and the total current is the dot product result. This result is then converted to digital values by the analog-to-digital converter (ADC). As matrix multiplication can be performed on crossbars easily, fully-connected layers can be directly mapped to crossbars.

To demonstrate the mapping of convolutional layers on crossbars, Figure 2(a) shows an example of a quantized convolutional layer and Figure 2(b-c) illustrate different mapping methods for this layer on crossbars. In the semi-folded mapping method, as shown in Figure 2(b), several whole rows of the input (at least can be multiplied by the kernel) is first unfolded into a vector, and the kernel weights are then mapped to the corresponding crossbar memristive cells. Since the unfolded input vector needs to be multiplied by different parts of the kernel, each kernel must be duplicated several times. On the other hand, the fully-folded mapping method, as depicted in Figure 2(c), unfolds each



(a) An example of a quantized convolutional layer with 6 kernels.

(b) Semi-folded mapping of this layer to the crossbar. 3 cycles with 12 crossbars.

(c) Fully-folded mapping of this layer to the crossbar. 6 cycles with 4 crossbars; 2 cycles with 12 crossbars;

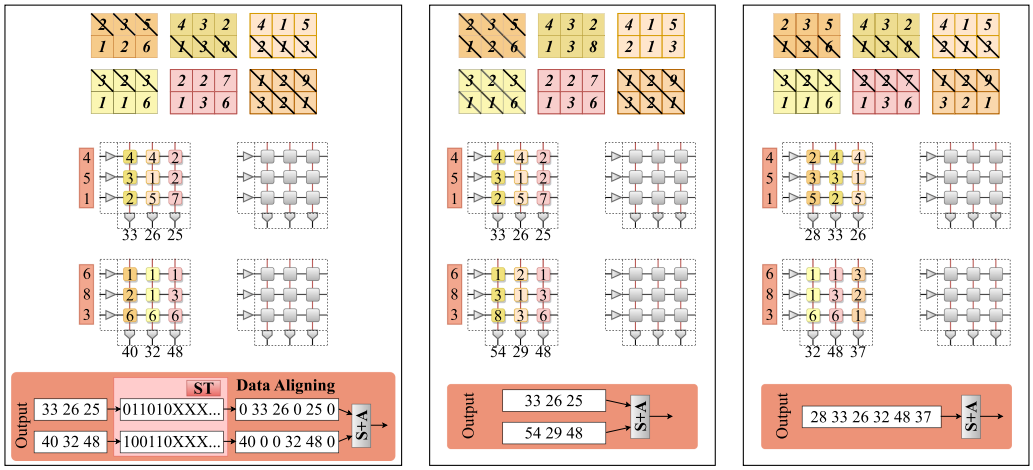
Fig. 2. (a) An example of a quantized convolutional layer; (b-c) Different mapping methods for this layer on crossbars, and their execution cycles and the number of used crossbars.

kernel to a vector and maps it to the crossbar memristive cells. The input of each layer is reshaped to a series of vectors with the same shape as the unfolded kernel vectors, and sent to the wordline of crossbars one by one. In both mapping methods, a large kernel occupies several crossbars that are connected by peripheral circuits. However, there are some non-used memristive cells in the semi-folded mapping approach. Conversely, the fully-folded mapping method can fully occupy the crossbar. Additionally, the weights in the fully-folded mapping method can also be duplicated several times to enable different input voltage injections in parallel among different replicas, which increases the parallelism of this method [6, 42] to reduce execution cycles. Thus, our proposed approach focuses on optimizing DNN models based on the fully-folded mapping scheme.

2.2 IMP-aware Fine-grained DNN Pruning

Pruning methods have been proposed to reduce model complexity and fit complex DNN models into the limited crossbars of the IMP architecture. The current IMP-aware pruning methods are fine-grained and focus on crossbar column/row pruning. For instance, XBA [27] prunes the columns of weights within a crossbar and then recombines the weights from different sparse crossbars to reduce the number of used crossbars. Similarly, SPRC [35] introduces a multi-group Lasso method that prunes a group of columns of weights in a crossbar. And this method still relies on crossbar-column pruning. PIM-Prune [8] proposes to exploit the sparsity in both row and column directions of the weight matrix and designs a new hardware data path to support their pruning method. Nonetheless, these fine-grained methods require hardware masks to align the output/input data of crossbars to function correctly, which introduces significant hardware overhead [8, 35].

To produce the necessary mask, PIM-Prune [8] proposes to use the Sparsity Table (*ST*). Take the crossbar-column pruning on the layer of Figur 2(a) with fully-folded mapping for an example, as shown in Figur 3(a), each 1 in *ST* represents picking a number from the original crossbar output, and 0 means inserting a 0 into the final output queue. Therefore, each crossbar requires a corresponding mask memory in the *ST* [8]. Meanwhile, the length of a mask in an *ST* equals the maximum pruning rate times the number of weight columns in each crossbar. For example, the



(a) Crossbar-column pruning. *ST* is needed. (b) IMP-aware kernel-group pruning. (c) Crossbar pruning

Fig. 3. The execution of a convolutional layer with three types of sparsity: crossbar-column, kernel-group, and crossbar. Crossbar-column pruning needs the data alignment hardware unit – Sparsity Table (*ST*).

ST size calculated in Figur 1 uses a maximum pruning rate of 32 and each crossbar is configured with 32 columns of weights, resulting in a *ST* of 1KB for 8 crossbars and introducing about 44% more memory and 10% more area into an In-Situ Multiply Accumulate (*IMA*) [42], and the mask may support a higher pruning rate. Also, supporting crossbar-row pruning requires another *ST* to align the input [8], further exacerbating the overhead and making it unaffordable. Moreover, when configured with DNN models, different crossbars may own different pruning rates, leading to a significant waste of resources in the *ST*, which is designed for covering the maximum pruning rate.

2.3 DNN Quantization

Meanwhile, quantization is an approach to reducing the complexity of DNNs and removing FP operations from DNNs for IMP architectures. Traditional quantization methods [7, 19] train a quantized model to determine the appropriate scaling factors that minimize the accuracy loss. PACT [7] optimizes the input clipping parameter during training to identify the appropriate quantization scale for the input, while for weight quantization, they use statistics-aware weight binning to determine the optimal scaling factor based on the statistical characteristics of the weight distribution. However, the scaling factors derived by this method are still in FP format, which necessitates the use of FP processors. IAO [19], in addition to training for optimal scaling factors, also proposes a scheme to eliminate the need for FP scaling factors by using fixed-point multiplication. However, this approach still requires fixed-point multipliers, which introduce overhead to the hardware. As shown in Figure 1, each multiplier introduces about 7% power overhead and 9% area overhead into each *Tile* [42], leading to lower energy efficiency and hardware integration density, longer pipeline cycles and worse performance. Consequently, it is crucial to eliminate all FP scaling factors and only incorporate integers in the quantized DNN model.

2.4 ReRAM Non-ideality

ReRAM-based IMP accelerators offer notable advantages in terms of area and power efficiency after pruning and quantization of the inferred DNNs. However, ReRAM suffers from various non-idealities in the analog domain, leading to weight deviations in pre-trained DNN models and further resulting in a significant decrease in inference accuracy. One prevalent non-ideality is

SAFs [4], where some device cells are consistently in either a high-resistance state (i.e., SF1) or a low-resistance state (i.e., SF0). It is a kind of error from the fabrication process and it is hard to avoid. The previous work [4] has reported that SF1 and SF0 can affect about 9.04% and 1.75% of the total device cells, respectively. And the location of cells with SAF error can be read out and recorded in advance [4]. Another non-ideality is write variation [15], which encompasses cycle-to-cycle variations (CCV) and device-to-device variations (DDV). DDV can be diminished with precise manufacturing control and fabrication process advancements, but CCV is an intrinsic property of ReRAM resistive switching behavior and is caused by the stochastic nature of the formation and rupture of a conductive filament, i.e., oxygen vacancies generation and migration process [3]. Previous works have investigated huge amounts of stochastic write variations and found that the variations follow a lognormal distribution [3, 33], indicating that the ReRAM resistance variations follow Equation (1), where r is the ideal resistance value to be programmed, r' is the actual value programmed, and $\theta \sim N(0, \epsilon^2)$, N is the normal distribution with a mean of 0 and variance of ϵ^2. We also employ this formulation to simulate the write variation in this work.

$$r' \leftarrow e^{\theta} \cdot r, \quad \theta \sim N(0, \epsilon^2) \tag{1}$$

Previous works have attempted to mitigate the impact of non-idealities in ReRAM-based IMP accelerators from both software and hardware perspectives. AIIR [3] proposed a joint algorithm-design solution that combines knowledge distillation and random sparse adaptation, but it introduces some hardware overhead (~15% more area). RVComp [13] aimed to compensate for non-idealities by utilizing extra ReRAM cells to represent the difference between ideal and actual resistance values, also resulting in hardware overhead. Not only do these methods result in less integration density but also more operational stages are introduced into the inference stage, lowering the efficiency. On the other hand, CorrectNet [11] and DRD [33] utilized non-idealities-aware training methods to reduce the impact of non-ideal ReRAM cells by software algorithms, but they neglect the execution flow of model inference on ReRAM, disregarding the influence of factors such as crossbar size, ADCs, and others. Besides, many of them do not consider the interaction with IMP-aware pruning. VACTSF [18] tries to remove weights from the model during processing the write variation, but a huge accuracy drop (6.46%) is introduced by this method. In our method, to enhance reliability while maintaining the benefits of pruning, we need to improve the accuracy of each technique and investigate the co-optimization of all proposed strategies.

3 METHODOLOGY

In this section, we elucidate our methodologies designed to optimize DNN models for IMP architectures. First, we introduce a crossbar-aligned pruning algorithm, which includes kernel-group pruning and crossbar pruning to achieve significant model sparsity without hardware overhead. Given that these two pruning strategies are more coarse-grained than existing IMP-aware crossbar column/row pruning, and coarser pruning generally implies a larger accuracy drop under equivalent sparsity levels [34], we also design a dynamic zerorize-recover learning framework to gain the better pruned architecture and weights for high accuracy. Second, we propose an integer-only quantization strategy, eliminating the need for multipliers by reusing the bit-shift unit in the IMP architecture. Third, we propose a runtime-aware non-ideality adaptation scheme, which explicitly considers the IMP inference flow. To further mitigate the error induced by non-ideality, we also present a self-compensation scheme, which only leverages the used crossbar cells themselves to reduce the error by mutual calibration, but also maintains the flexibility to leverage more cells to further reduce the error. Besides, we also establish a comprehensive DNN learning framework based on the zerorize-recover learning process to co-optimize these strategies within a single training process to further improve the overall accuracy.

3.1 Crossbar-Aligned DNN Pruning & Zerorize-Recover Training

Current IMP-aware pruning methods require extra hardware for data alignment, as shown in Figure 3(a), which reduces integration density and further lowers the number of crossbars on the IMP chip (e.g., the area of the data alignment unit is about 6 time that of a crossbar array), thus making it difficult to fully benefit from these methods in terms of hardware savings. To solve this problem, we propose a crossbar-alignment pruning algorithm that ensures the pruned DNN model is data-aligned without additional hardware units. Kernel pruning and crossbar pruning can maintain data alignment during pruning. However, as reported in prior work [34] that coarser pruning granularity leads to a more decrease in accuracy at the same sparsity level, these fine-grained IMP-aware pruning methods [8, 27, 30] result in significant accuracy drops when directly employed for data-aligned pruning directly. To solve this problem, we first introduce a multi-grained pruning method by combining kernel pruning and crossbar pruning to enlarge the sparsity rate and reduce accuracy loss under data alignment. Then we design a novel dynamic zerorize-recover procedure to finish the pruning process, widening the exploration space for higher accuracy.

3.1.1 Kernel-group Pruning. Kernel pruning is a technique that directly removes entire kernels from DNNs. If we consider a DNN model with two connected convolutional layers, denoted as l_1 and l_2, then the number of kernels in layer l_1 is equal to the number of channels in layer l_2. We represent the shape of the kernels in these two layers as $C_2 \times K_1 \times K_1 \times C_1$ and $C_3 \times K_2 \times K_2 \times C_2$, where C_n denotes the number of channels in layer l_n, C_{n+1} denotes the number of kernels in layer l_n, and K denotes the kernel size. From Figure 2(c), it can be derived that layer l_1 requires $\lceil C_2/XB_w \rceil \times \lceil (K_1 \times K_1 \times C_1)/XB_h \rceil$ crossbars, where XB_w and XB_h represent the width and height of a crossbar, respectively. Similarly, layer l_2 needs $\lceil C_3/XB_w \rceil \times \lceil (K_2 \times K_2 \times C_2)/XB_h \rceil$ crossbars. By removing kernels, the number of kernels C_{n+1} in layer l_n can be reduced. To maximize the savings of crossbars, we introduce a kernel-group pruning method, as shown in Figure 3(b), which removes a group of kernels from each layer and ensures that the number of remaining kernels in each layer is an integer multiple of XB_w. During this pruning process, we first calculate the number of kernels that need to be removed in each layer using the pruning ratio and the importance rank . Then, we round the number of remaining kernels of each layer to integer times of XB_w.

Kernel-group pruning has the potential to achieve more crossbar savings. Specifically, let C'_{n+1} represent the remaining kernels in layer l_n. Based on the above design, the number of used crossbars in layer l_1 and layer l_2 are $C'_2/XB_w \times \lceil (K_1 \times K_1 \times C_1)/XB_h \rceil$ and $C'_3/XB_w \times \lceil (K_2 \times K_2 \times C'_2)/XB_h \rceil$, respectively. If the crossbar is square (i.e., $XB_w = XB_h$) or $(K_2 \times K_2 \times XB_w) \mid XB_h$, the by-product of kernel-group pruning in layer l_1 can guarantee that the kernels in layer l_2 can fully occupy crossbars from the rowing aspect, leading to maximum crossbar savings. In other cases, the by-product of kernel-group pruning in layer l_1 can still save some crossbars of layer l_2 from the rowing aspect. However, kernel-group pruning may result in a larger accuracy drop than crossbar pruning methods, especially under a large pruning ratio [34], as it is more coarse-grained than crossbar pruning. To preserve accuracy, the amount of crossbar savings achieved through kernel-group pruning is limited, indicating the potential for further compression of the pruned model.

3.1.2 Crossbar Pruning. Crossbar pruning involves removing the crossbar-block of weights from the DNN model to save crossbars, as depicted in Figure 3(c). As an unstructured pruning method, crossbar pruning is exclusive to crossbar architecture. By removing a whole crossbar at once, data alignment units are not required in crossbar pruning. Moreover, crossbar pruning can compress both convolutional and fully-connected layers, unlike kernel pruning, which can only compact convolutional layers. Based on the mapping scheme, a mask layer is required after each convolutional or fully-connected layer for crossbar pruning, where each mask value is associated

ALGORITHM 1: Comprehensive DNN Learning Framework for IMP

Input: model, training data and settings, zerorize start epoch: s, prune_ratio: p, xb_size: XB_s, quan_bits: q, non_ideality: θ
Output: the well-trained, compact, and reliable model

```
1  Function Comprehensive_Learning(model, δ):
2      for t ← 1 to epoch_half do
3          if t ≥ s and (t%2 == 0 or t == epoch_half) then
4              scaling_factor ← quantize(model, q); # Prepare quantization
5              Force all scaling factors to 2^n; #Integer-Only quan.
6              Calculate Weight_quan according to the new scaling_factor; #Integer-Only quan.
7              forward_with_simulator(scaling_factor, weight_quan, θ, XB_s); #Training as Figure 7;
8              update_with_sparsity(model, δ); #Sparse training for pruning
9              Imp_r ← Sort kernels/Crossbars by |δ|; # Prepare pruning
10             Find δ threshold δ^l_th of each layer by Imp_r, p, XB_s;# Crossbar-aligned pruning
11             Zerorize δ^l_i if δ^l_i < δ^l_th; #Temporarily removing weights for pruning
12         else
13             forward(model); #Training
14             update_with_sparsity(model, δ); #Sparse training for pruning
15     Remove all weights from the model with zero δ^l_i;
16 Initialize the model and its weights randomly;
17 #Phase 1 – Kernel-group pruning
18 Comprehensive_Learning(model, γ);
19 #Phase 2 – Crossbar pruning
20 model_mk ← Mask(model);
21 Comprehensive_Learning(model_mk, mask);
```

with a crossbar, and zeroizing a mask value means removing all weights in this corresponding crossbar. Mask layers are differentiable to the loss function, hence optimized to reduce loss during the training stage. After training, crossbar-blocks of weights with zerorized mask values are removed from the DNN model, and other mask values are multiplied to corresponding weights before inference to eliminate the mask operation in the inference stage.

Although crossbar pruning can achieve higher sparsity than kernel-group pruning, the mask layer poses challenges to the training stage. These challenges primarily stem from two aspects. First, deeper DNNs are challenging to train, as various layers in the DNN tend to learn at different rates [39]. The mask layer increases the difficulty of model training to converge. Second, in the following zerorize-recover training process, more mask values lead to a larger variation of the compressed model architecture during training, intensifying the difficulty of convergence. Therefore, we propose to use kernel-group pruning to compress the DNN first to reduce the number of mask values, reducing the impact of mask layers in the crossbar pruning for higher accuracy. Then use crossbar pruning for higher sparsity. Thus, it is called the multi-grained pruning scheme.

3.1.3 Dynamic Zerorize-recover Framework. To further minimize the accuracy loss incurred by these coarse-grained pruning methods, besides the multi-grained pruning scheme, we propose a model learning framework that optimizes the weights and architecture during the training process via a dynamic zerorize-recover procedure. Algorithm 1 describes the proposed framework, which is divided into two phases: Kernel-group pruning (*Line 18*) and Crossbar pruning (*Line 20–21*). In the pruning process, a crucial step is to determine which parts of the model can be removed without sacrificing much accuracy. In this regard, an explicit learning medium δ is required to

distinguish the importance of the weight. For kernel-group pruning, we remove entire kernels at once, and we can use the trainable weights γ of the Batch Normalization (BN) layer to provide relevant instructions without any overhead, as γ can indicate the importance of kernels [31]. And for crossbar pruning, we design a mask layer to distinguish the importance of crossbar-size weights.

The following is a description of the core function *Comprehensive_Learning* (*Line 1–15*) in detail. Initially, the model undergoes training for several epochs, named initialize epochs ($t < s$ in *Line 3*, *Line 13–14*), to allow all the learning medium (importance factors) δ to have informative values that reflect kernel/crossbar importance, rather than being initialized randomly (*Line 16, Line 20*). Furthermore, we use sparse training [31] (*Line 8, Line 14*) throughout the training process, forcing the importance factors δ to approach zero for safely removing unimportant weights in the pruning process. The function next enters the zerorize-recover training epochs (*Line 2–14*). In the zerorize epoch (*Line 4–11*), for the crossbar-aligned pruning, the importance factors δ are first sorted according to their absolute values to obtain the global importance rank of kernels/crossbars in the entire model (*Line 9*). During the entire training process, these importance factors are jointly optimized with the network weights, and the network can automatically identify the importance of each kernel/crossbar. Next, the threshold of importance factors is calculated for each layer (*Line 10*) to ensure that the compact model fully occupies each used crossbar. Finally, the unimportant parts are temporarily shielded for the current zerorize epoch by setting δ_i^l to zero for those parts with importance factors less than the threshold (*Line 11*).

After each zerorize epoch, we employ a recover epoch (*Line 3, Line 13–14*) to improve the accuracy of the model. Similar to knowledge distillation, in the zerorize epoch, the training process aims to use a small model to cover all knowledge from the original model. There are two scenarios in this process: 1) if the loss is small, then the network retains these weights in the whole training process; 2) if the loss is huge, the optimizer significantly updates the weights. In the second scenario, the recover epoch enables these weights to reach another region where the loss is small. Therefore, the recover epoch enables the learning process to obtain better weights for high accuracy.

The recover epoch also provides an opportunity for the previously zerorized importance factors (i.e., δ_i^l) to recover and potentially play a crucial role in subsequent training and inference processes, allowing our pruning method to find a more efficient model architecture with higher accuracy. Equation (2) provides the common calculation formula for a convolutional layer with δ and an activation function (σ), where L represents the layer index, X^L represents the input of layer L, W^L represents the weight between layer $L+1$ and L, and σ^L represents the activation function at layer L. The symbol $\circledast$ denotes the convolution operation. During the recover epoch, we mainly focus on updating the importance factors δ^L that are set to zero during the previous zerorize epoch. The gradients of δ^L to the final loss function can be calculated using Equation (3). Given that some values in δ^L have been zerorized and the commonly used activation function (σ) is *ReLU*, the output of σ^L is also zero under *ReLU*. Although *ReLU* is not differentiable at zero, it is commonly assumed that its derivative is also zero. Therefore, we can easily obtain Equation (4).

$$X^{L+1} = \sigma^L(\delta^L \cdot (W^L \circledast X^L + B^L)) \tag{2}$$

$$\frac{\partial loss}{\partial \delta^L} = \frac{\partial loss}{\partial X^{L+1}} \cdot \sigma^{L'} \cdot (W^L \circledast X^L + B^L) \tag{3}$$

$$\delta^L = 0 \rightarrow \sigma^{L'} = 0 \rightarrow \frac{\partial loss}{\partial \delta^L} = 0 \tag{4}$$

Unfortunately, it stacks at a deadlock. To break the deadlock caused by the zerorized δ^L, small values can be added to δ^L. However, many training algorithms employ momentum to accelerate convergence, which accumulates gradients from previous steps to determine the direction to go. In particular, weight updates with momentum are shown in Equation (5)-(6), where lr is the learning

rate, z_t is the updated value from the previous step t, and m is the accumulation coefficient. This allows the weights W_t to be updated to W_{t+1} by combining the gradients of the current and past steps.

$$z_{t+1} = m \cdot z_t + \frac{\partial loss}{\partial W_t} \tag{5}$$

$$W_{t+1} = W_t - lr \cdot z_{t+1} \tag{6}$$

As a result, even though the current gradients of δ^L are zero, the proposed framework can update them to recover from zero by following the last updating directions, without requiring additional steps. This process is fully automated and efficient. Previously zerorized kernels/crossbars may become significant, and they are not zerorized in the future zerorize epoch, thereby altering the architecture of the final compact model. Furthermore, the training process aims to minimize loss and improve accuracy, which enables the compact architecture to become better and better. In contrast to other IMP-aware pruning methods [8, 27, 30], which solely propose pruning schemes without an optimized learning process, our proposed framework obviously improves accuracy.

3.2 Integer-Only DNN Quantization

Quantization is a technique used to represent the FP inputs (activations) and weights in DNNs using integers with n-bits. However, this technique typically involves non-integer scaling factors to adjust the range of inputs and weights, which are necessary to achieve better accuracy [7, 19]. These scaling factors require the use of multipliers, which can introduce significant power and area overhead into the IMP device, as demonstrated in Section 2. Given the increasing need for high integration density [27], the area of the multiplier, however, can be up to 11 times that of the crossbar. Therefore, eliminating multipliers from IMP architectures can significantly increase the number of crossbars, thereby enabling the support of more complex DNN models.

To remove FP from DNNs for IMP architectures, we adopt quantization for both inputs and weights. Specifically, since the voltage signals in IMP architectures can represent both positive and negative integers, we adopt a symmetric quantization scheme for inputs. However, as the conductance of the crossbar cells can only be positive, we employ an asymmetric quantization scheme for weights. For instance, in the case of a convolutional layer, the quantization process can be expressed by Equation (7), where X^L, W^L, and B^L denote the inputs, weights, and biases of the layer L, respectively, S represents the scaling factor, Q denotes the corresponding quantized value, and Z denotes the zero-point. And all values, except for Q_X, are determined during the training process and remain constant during inference. In the inference stage, the input X^{L+1} of the next layer is computed as $W^L \circledast X^L + B^L$ (with BN layers fused into the convolutional layer [19] if BN layer exists, same to mask layers), as shown in Equation (8). Meanwhile, the scaling factor $S_{X^{L+1}}$ of the next layer is also determined during training, and X^{L+1} can be represented as $X^{L+1} = S_{X^{L+1}} Q_{X^{L+1}}$. However, the crossbar in IMP architectures only supports integer arithmetic operations $(Q_X \circledast (Q_W - Z_W), Q_B - Z_B)$, making it necessary to derive the quantized value $Q_{X^{L+1}}$ in Equation (9) for performing the next layer's inference in the crossbar.

$$X^L = S_{X^L} Q_{X^L},\ W^L = S_{W^L}(Q_{W^L} - Z_{W^L}),\ B^L = S_{B^L}(Q_{B^L} - Z_{B^L}) \tag{7}$$

$$X^{L+1} = W^L \circledast X^L + B^L = S_{X^L} S_{W^L} Q_{X^L} \circledast (Q_{W^L} - Z_{W^L}) + S_{B^L}(Q_{B^L} - Z_{B^L}) \tag{8}$$

$$X^{L+1} = S_{X^{L+1}} Q_{X^{L+1}} \Rightarrow Q_{X^{L+1}} = \frac{S_{X^L} S_{W^L}}{S_{X^{L+1}}} Q_{X^L} \circledast (Q_{W^L} - Z_{W^L}) + \frac{S_{B^L}}{S_{X^{L+1}}}(Q_{B^L} - Z_{B^L}) \tag{9}$$

To minimize the power and area overhead associated with FP scaling factors in IMP devices, we propose to approximate these factors by powers of 2, as shown in Equation (10). This approach allows for the calculation of scaling factor multiplications using bit-shift units in crossbars. Unlike

the widely used representation of FP values as $a \cdot 2^b$ ($1 \leq |a| < 2$) in computers under the *IEEE Standard 754* [21], approximating $a = 1$ in our method will sacrifice some accuracy.

$$\begin{aligned} &\text{Force } S_{X^L} = 2^{a1}, S_W = 2^{a2}, S_B = 2^{a3}, S_{X^{L+1}} = 2^{b1} \\ &Q_{X^{L+1}} = 2^{a1+a2-b1} Q_{X^L} \circledast (Q_{W^L} - Z_{W^L}) + 2^{a3-b1}(Q_{B^L} - Z_{B^L}) \end{aligned} \tag{10}$$

To mitigate the loss of accuracy in our method, we integrate our integer-only quantization scheme into the learning process (i.e., Algorithm 1, *Line 4–6*) to enable its co-optimization with the crossbar-aligned pruning and the runtime-aware non-ideality adaptation. First, scaling factors are computed based on the quantization bits (q, we use 8) (*Line 4*). These scaling factors are then rounded to the nearest power of 2 for easy computation on the *S+A* units in the IMP device (*Line 5*). Finally, the quantized weights (*Weight_quan*) are computed using these scaling factors to complete the quantization process (*Line 6*). Moreover, it does not interfere with the recover process (*Line 13–14*) to reach another region that has a small loss.

3.3 Runtime-aware Non-Ideality Adaptation with Self-Compensation

The aim of training a DNN is to find the global minimum of the loss function. However, due to the flexibility of the DNN weights, there exists a region where the loss function remains small despite not being the global minimum [33]. Within such a region, the DNN's weights can vary without significantly impacting the accuracy. To create a DNN that is robust to ReRAM non-ideality, we inject the non-ideality into the training process. This ensures that the trained DNN is resilient to non-ideality, improving accuracy in non-ideal ReRAM-based IMP accelerators.

3.3.1 Crossbar Size v.s. Non-ideality Error. We first analyze the actual inference process of DNN models on the ReRAM-based crossbar and the error introduced in this process. In this analysis, each ReRAM cell can store a 2-bit weight and each weight of the DNN model is quantized into 8 bits and then mapped to 4 ReRAM cells. The input of the DNN model is also quantized into 8 bits and injected into the crossbar bit by bit (each bit is represented by v, $v \in \{0, 1\}$). The height of each crossbar is H, and the comparator in ADC can round the output of each column in the crossbar into an integer-format digital value. For a dot product operation of vector-vector multiplication (VVM) mapped onto the crossbar (like $\{4\ 5\ 1\ 6\ 8\ 3\} \cdot \{2\ 3\ 5\ 1\ 2\ 6\}$, as shown in Figure 2(c)), the weight vector is mapped onto the same column of several crossbars, and the input vector is converted to voltage and injected into the wordline of the crossbar. Assume the dimensions of these two vectors are both D, and H is the height of each crossbar, then the number of used crossbars is $\lceil D/H \rceil$. For simplicity, we assume that $\lceil D/H \rceil = D/H$, which means that the vector can fully occupy the used crossbar. Otherwise, if $\lceil D/H \rceil > D/H$, then some ReRAM cells are not used and programmed to 0, and the error should be smaller. Take write variation as an example, as previously mentioned in Section 2, it stems from the stochastic nature of the ReRAM cell, leading to potential non-ideality in each cell. Following previous works [3, 33], in this analysis and our experiments, we use c to represent the ideal value to be programmed onto the ReRAM cells ($c \in \{0, 1, 2, 3\}$, $c = 1/r$, r is the resistance), and c' is the actual value programmed (according to Equation (1), then $c' = e^{-\theta} \cdot c$; considering θ is in a normal distribution and according to [33], we also use $c' = e^{\theta} \cdot c$.).[1] Equation (11) shows the ideal calculation results for a VVM operation, where $\sum_{h=1}^{H} v_{dh} \cdot c_{dh}$ represents the output from a single crossbar and then it is subsequently aggregated by the peripheral circuitry, summing up the results from all connected crossbars [42]. For the definitions of symbols used in this and the following equations, please refer to Table 1. When taking write variation into consideration, the

[1]The ratios of resistance in this article are set to 33:50:100:33000, which correspond to the configured weight values of 3, 2, 1, and 0, respectively.

calculation results become Equation (12). The error introduced by the write variation is given by Equation (13).

$$Results = \sum_{d=1}^{D/H}\left(\sum_{h=1}^{H} v_{dh}\cdot c_{dh}\right), \qquad v_{dh}\in\{0,1\}, \quad c_{dh}\in\{0,1,2,3\} \tag{11}$$

$$Results' = \sum_{d=1}^{D/H} Round\left(\sum_{h=1}^{H} v_{dh}\cdot c'_{dh}\right) = \sum_{d=1}^{D/H} Round\left(\sum_{h=1}^{H} v_{dh}\cdot e^{\theta}\cdot c_{dh}\right) \tag{12}$$

$$\begin{aligned} Error = |Results' - Results| &= \left|\sum_{d=1}^{D/H} Round\left(\sum_{h=1}^{H} v_{dh}\cdot e^{\theta}\cdot c_{dh}\right) - \sum_{d=1}^{D/H}\left(\sum_{h=1}^{H} v_{dh}\cdot c_{dh}\right)\right| \\ &= \left|\sum_{d=1}^{D/H} Round\left(\sum_{h=1}^{H} v_{dh}\cdot (e^{\theta}-1)\cdot c_{dh}\right)\right| \end{aligned} \tag{13}$$

To quantify the magnitude of the error introduced by the write variation, we calculate the mathematical expectation of the error, represented by $E(Error)$. As the input value (v), the weight value (c), and the write variation degree of ReRAM cells (θ) [3] are independently distributed, the mathematical expectation of the error is shown in Equation (14). In the quantized DNN models, we assume that each bit of the input (v) is 0 or 1 with equal probability (inject one bit of input into the crossbar each time), and the weight (c) is in {0,1,2,3} with the same probability. Then $E(v_{dh}) = E(v) = 0.5$ and $E(c_{dh}) = E(c) = 1.5$. And the expectation of the lognormal distribution is $E(e^{\theta}) = e^{\epsilon^2/2}$. Thus, the expectation of the error is simplified to Equation (15).

$$E(Error) = \left|\sum_{d=1}^{D/H} Round\left(\sum_{h=1}^{H} E(v_{dh})\cdot (E(e^{\theta})-1)\cdot E(c_{dh})\right)\right| \tag{14}$$

$$\begin{aligned} E(Error) &= \sum_{d=1}^{D/H}\left|Round\left(\sum_{h=1}^{H} 0.5\cdot (e^{\epsilon^2/2}-1)\cdot 1.5\right)\right| \\ &= \frac{D}{H}\cdot Round(H\cdot 0.75\cdot (e^{\epsilon^2/2}-1)), \quad as\ E(e^{\theta}) = e^{\epsilon^2/2} \geq 1 \end{aligned} \tag{15}$$

Therefore, from Equation (15), we can deduce that $E(Error)$ is related to the height (H) and the error standard deviation (ϵ) of the crossbar. Figur 4(a) depicts the relationship between $E(Error)$ and H under different values of ϵ according to Equation (15). From this figure, we can more intuitively find that when $H\cdot 0.75\cdot (e^{\epsilon^2/2}-1) < 0.5$, which is equivalent to $H < \frac{2}{3\cdot(e^{\epsilon^2/2}-1)}$, then $E(Error) = 0$. This means that when H is small, the error accumulated in each crossbar is insignificant, and it can be eliminated through the rounding function in the ADC. On the other hand, when $n \leq H\cdot 0.75\cdot (e^{\epsilon^2/2}-1) < n+0.5$, where $n = 0, 1, 2, \ldots, E(Error) = \frac{nD}{H}$. This function decreases as H increases. Therefore, H should be increased to the upper bound, which is $H = \lfloor\frac{2+4n}{3\cdot(e^{\epsilon^2/2}-1)}\rfloor, \epsilon \neq 0$. Also, the function graph of $H = \lfloor\frac{2}{3\cdot(e^{\epsilon^2/2}-1)}\rfloor, \epsilon \neq 0$ is shown in Figure 4(b), and the region below the blue line indicates the valid values of H that guarantee $E(Error) = 0$. When H is small, the rounding function by the comparator in ADC can automatically correct the error, as illustrated in Figure 5. Meanwhile, due to factors such as nonlinearity and coupling effects between the crossbar cells, activating many rows concurrently in a crossbar may cause interference that consequently affects the precision of computations. As such, there are limitations on the number of rows that can be activated simultaneously. When the number of rows that can be simultaneously activated is less than the height of the crossbar, it is equivalent that the unactivated cells are programmed to 0, and the error should be smaller.

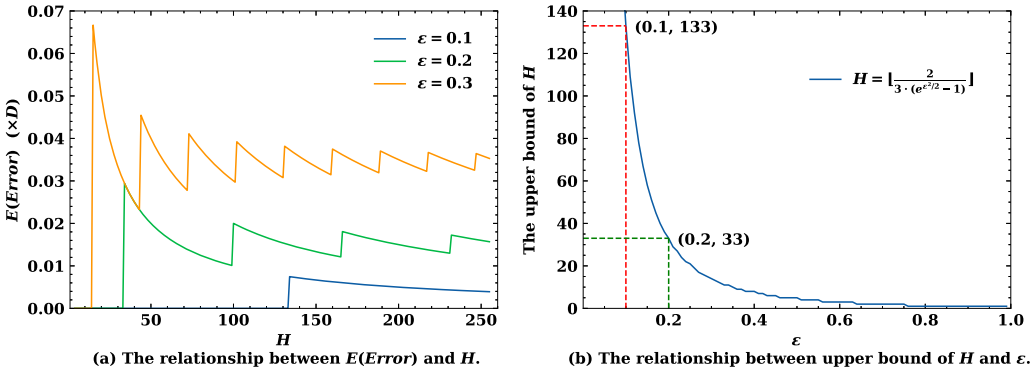


Fig. 4. Relationship between error expectation ($E(Error)$), crossbar height (H), and write variation degree (ϵ).

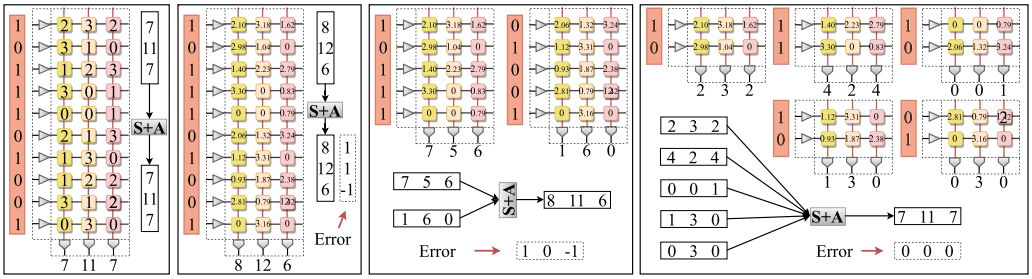


Fig. 5. Diagrammatic representation of the effect of crossbar height on the error of write variation.

Until now, we establish a relationship among the height of each crossbar, the write variation degree of ReRAM cells, and the expectation of the error introduced by write variation. This demonstrates the necessity of perceiving the runtime inference process during the non-ideality adaptation, which diminishes the error only through the comparator in the ADC, without any hardware overhead. In addition, this finding serves as a useful reference not only for supporting the algorithm in this paper but also as a guideline for the further design of ReRAM-based IMP devices. When designing IMP devices, it is also important to consider the impact of crossbar height on the device's tolerance to errors. This consideration should be balanced with other crucial factors such as power and area, for optimizing the overall performance of ReRAM-based IMP devices.

Regarding the SAF, as outlined in Section 2, it stems from the fabrication process. Consequently, the locations of SF1 and SF0 for a specific device remain constant and can be recorded in advance [4]. Consistent with previous works [3, 4, 20, 26, 29, 45], our method trains the model for the device with the known locations of SF1 and SF0. In our experiments, the proportion of SF1 and SF0 cells to total cells is 9.04% and 1.75% respectively, which aligns with the assumptions in prior works [3, 4].

3.3.2 Self-compensation Scheme. From the preceding analysis and insights in Figure 4(b), it is apparent that as ϵ increases, the upper bound of H for ensuring $E(Error) = 0$ decreases approximately exponentially. In practical IMP design, factors such as power and area necessitate that a crossbar cannot be too small, with a commonly used size being 128×128. Thus, only when $\epsilon = 0.1$

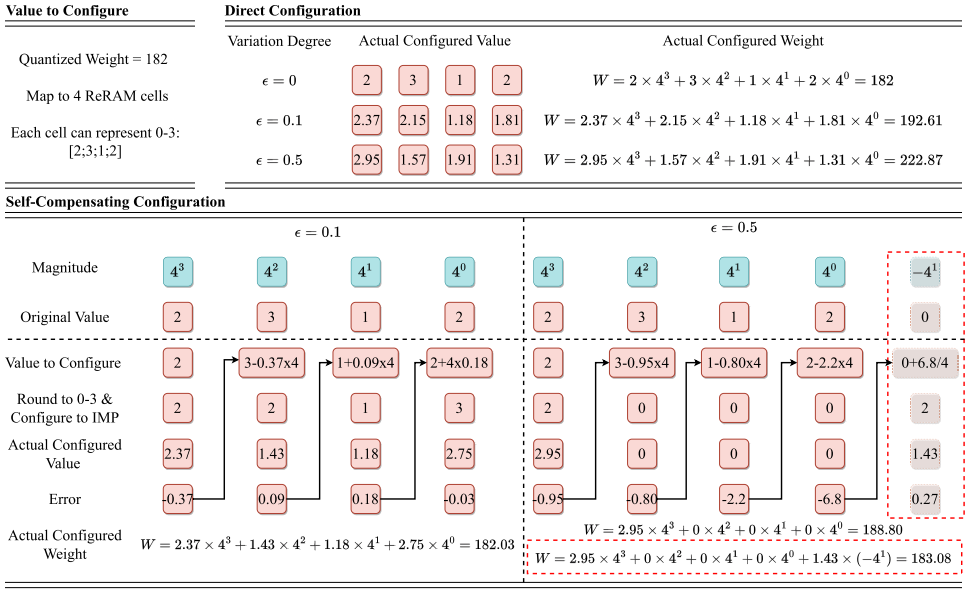


Fig. 6. Schematic illustration of the self-compensation scheme. The area highlighted in red dashed boxes indicates the use of an additional cell to further diminish the error.

can the error induced by the write variation be recuperated by the rounding function. However, in actual application scenarios, the write variation degree ϵ might exceed 0.1. Therefore, as depicted in Figure 6, we also propose a self-compensation scheme to further mitigate the error caused by write variation. Our compensation scheme is used during the configuration phase of the model to the IMP device, not during the model training process. Nevertheless, we incorporate it into our training process to better fit our trained model with this compensation scheme. Furthermore, without loss of generality, this scheme can also be applied to mitigate errors from SAF.

As illustrated in Figure 6, we take configuring a weight value of 182 as an example. This weight is quantized into 8 bits and mapped to 4 ReRAM cells. Each cell can represent 2 bits, with a range of 0−3. For configuring 182, the 4 ReRAM cells are assigned values 2; 3; 1; 2, respectively. When directly configuring 182 into the ReRAM cell, the actual configured value is 182 under no write variation error (i,e, $\epsilon = 0$). However, when ϵ is 0.1 or 0.5, the actual value configured will deviate much from 182 due to the presence of write variation. Therefore, we also propose a self-compensation scheme to reduce this error. It can rely on the used ReRAM cells only and incurs no hardware overhead, yet retains the flexibility to further minimize errors with additional cells. The magnitude of each value on the ReRAM cell is represented in the blue box. And in our self-compensation scheme, the ReRAM non-ideality still influences all cells we used.

During the model configuration process, we start from the cell that contains the most significant bit and proceed to the cell with the lowest significant bit. After configuring the first cell, we read out the actual configured value and calculate the error. For configuring the second cell, we multiply the error by 4 (i.e., the ratio between their magnitudes) and add it to the value to be configured. This new value is then rounded to the nearest integer within the range of $0 - 3$. Then configure this rounded value to the second ReRAM cell. The configuration process follows this procedure until all cells are configured, as illustrated in Figure 6. With our self-compensation scheme, when $\epsilon = 0.1$, the actual configuration value is 182.03, significantly reducing the error. When $\epsilon = 0.5$,

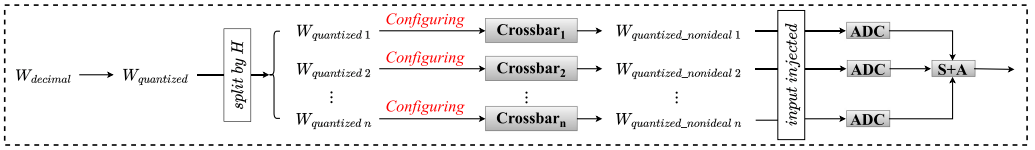
ALGORITHM 2: Self-Compensation Configuration Scheme

Input: the trained model, IMP device, additional compensation cells: n
Output: IMP device preloaded with the model

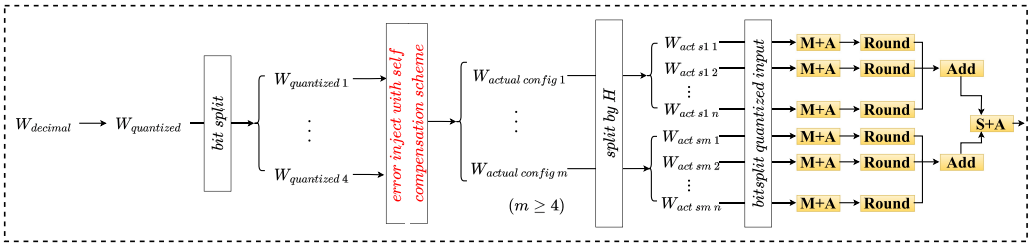
1 Split the model weights according to the crossbar height;
2 Distribute each weight to 4 cells; ***# Most significant bit first format***
3 Allocate crossbars for all weights;
4 **for** *all used crossbars* ***in parallel*** **do**
5 **for** $column_i$ *in current crossbar* ***and*** $column_i$ *is used* **do**
6 $col_i \leftarrow column_i\%(4+n)$;
7 **if** $col_i == 0$ **then**
8 $errors \leftarrow zero\ array$;
9 **for** row_i *in current crossbar* **do**
10 **if** $col_i < 4$ **then**
11 $cur\_need\_config \leftarrow weight[col_i][row_i] + 4 \times errors[row_i]$;
12 $cur\_need\_config\_03 \leftarrow Round\_03(cur\_need\_config)$;
13 $Configure\ cur\_need\_config\_03\ to\ crossbar\ cell[col_i][row_i]$;
14 $Read\ cell[col_i][row_i] to\ cur\_act\_config$;
15 $errors[row_i] \leftarrow cur\_act\_config - cur\_need\_config$;
16 **else if** $col_i < 4+n$ **then**
17 **if** $row_i == 0$ **then**
18 $magnitude\_cur\_col \leftarrow Round\_Pow4(average(errors))$ ***# From errors in col$_i$.***
19 $cur\_need\_config \leftarrow errors[row_i]/magnitude\_cur\_col$;
20 $cur\_need\_config\_03 \leftarrow Round\_03(cur\_need\_config)$;
21 $Configure\ cur\_need\_config\_03\ to\ crossbar\ cell[col_i][row_i]$;
22 $Read\ cell[col_i][row_i] to\ cur\_act\_config$;
23 $errors[row_i] \leftarrow cur\_act\_config - cur\_need\_config$;

the actual configured value is 188.80, also reflecting a significant error reduction. This process only introduces a read operation for each cell during configuration, not influencing its endurance and no other overhead. To further minimize the error, we can represent a weight using more cells, as depicted by the red dotted box in $\epsilon = 0.5$. The magnitude of this additional cell is determined as the nearest power of 4 to the average of all values that need to be configured on this particular crossbar column, as introduced in Algorithm 2. Then the actual configured value is 183.08 under $\epsilon = 0.5$.

The principle behind this method is that, given the random nature of write variations, the cells for the one weight may not trend towards larger or smaller values concurrently. As such, the errors from different cells of this weight can calibrate each other. Algorithm 2 elucidates our self-compensating model configuration process, executed on the host machine that configures the ReRAM IMP device. Initially, each model weight is allocated to the crossbar (*Line 1–3*). The weights are then configured simultaneously for all used crossbars (*Line 4*). The configuration is performed column by column for all used columns within the crossbar (*Line 5*). For the cells within the same column of the crossbar, the configuration is implemented row by row (*Line 9*). For each new weight column that needs configuration, the initial step is to set its error to an array with 0 (*Line 6–8*). When the current column corresponds to one of the columns that are used for weights not for compensation (*Line 10*), the weight is configured adhering to the compensating process described previously (*Line 11–15*). When the current column forms part of the additional columns for minimizing error (*Line 16*), the magnitude of the column is computed during the configuration of the first row by rounding the average of all errors in this column to the nearest power of 4 (*Line 17–18*). Every subsequent row within this column applies the same magnitude (*Line 19*),



(a) Configure & Execution flow of model inference on a crossbar-based IMP device;
Configuring means with our self-compensation scheme.



(b) Execution flow in our simulation for non-ideality adaptation training.
It is used in the training forward propagation stage. **M+A** means multiplication & accumulation.

Fig. 7. Execution flow of DNN on ReRAM IMP and its corresponding simulated flow used in the forward propagation stage of our training process.

owing to the inherent limitation of a crossbar column only supporting one magnitude. Upon finalizing all configurations, a ready-to-run IMP device is obtained. This self-compensation scheme can be used independently, but its effectiveness can be further enhanced when integrated into our training process. By considering the self-compensation scheme during training, the model can learn to adapt to the non-idealities with this compensation, resulting in improved performance and robustness.

3.3.3 Runtime-aware Non-ideality Adaptation. So far, it has been established that the inherent properties of the crossbar and optimized configuration methods can effectively reduce errors of non-ideality. Moreover, model training based on the runtime error can adapt more effectively to these errors [33], thereby enhancing accuracy. Therefore, we need to build a realistic runtime ReRAM simulator.

Since the height of the crossbar in the IMP device has a great influence on the error, directly multiplying DNN weights by e^{θ} as used in other methods [33] is not applicable for simulating the error of write variation. Figure 7(a) illustrates the inference process of the DNN model on a ReRAM-based IMP device. First, the 32-bit decimal weights ($W_{decimal}$) are quantized into 8-bit integers ($W_{quantized}$), and then these quantized weights are mapped and configured into the ReRAM crossbar, according to Figure 2(c). Our self-compensation scheme can be used in the configuration procedure to reduce the error from non-ideality. If the height of the weight matrix is greater than the height of the crossbar, the weight matrix is split into n smaller matrices, with each matrix mapped to one crossbar. During this configuration process, the non-ideality of ReRAM leads to the programmed values ($W_{quantized\_nonideal}$) differing from the required values. The multiplication of the non-ideal weights and inputs is calculated on the crossbar, and the output of each crossbar is converted into integer digital values by the ADC. Finally, the digital output of each crossbar is shifted and added by the *S+A* unit to obtain the final result.

To exploit the crossbar characteristics described above for higher accuracy under device non-ideality, it is crucial to accurately simulate the execution flow of the DNN on the ReRAM crossbar. Figure 7(b) depicts the simulation process for DNN runtime inference on the crossbar (named runtime simulator), which can be also used in the forward propagation stage of the training process

for perceiving the non-ideality. It begins by quantizing the decimal weight matrix ($W_{decimal}$) into 8-bits integers ($W_{quantized}$). To account for the non-ideality introduced, each quantized weight of 8 bits is divided into 4 parts, each corresponding to the ideal mapped value on a ReRAM cell of the crossbar. Then we use our self-compensation scheme algorithm to obtain the actual weight that will be configured on the ReRAM cell under this compensation and also inject the error introduced by ReRAM non-ideality. Next, we split the actually-configured weights according to the height of the crossbar (H) to simulate the calculation process on each crossbar. The multiplication and accumulation between the weights and the bit-split quantized input are then computed, and the result is rounded to integer format. By combining all outputs from all splits, we obtain the final results, which is equivalent to the results of running on the crossbar with device non-ideality. This process can calculate the errors introduced by non-ideality for different split parts in parallel while accurately simulating the crossbar operation for model inference, greatly improving efficiency.

We also integrate this runtime-aware non-ideality adaptation scheme into the learning process (i.e., Algorithm 1, *Line 7*) to enable its co-optimization with the crossbar-aligned pruning and the integer-only quantization. The forward propagation stage of our training process is performed with our runtime simulator to incorporate the non-ideality in this training process for adaptation. Since our pruning, quantization, and adaptation methods are all tied to the loss function, they can be jointly optimized to improve accuracy during the training process. This is why this algorithm is named comprehensive DNN learning. Finally, we eliminate all weights corresponding to zerorized δ_i^l (Algorithm 1, *Line 15*), resulting in a highly efficient, integer-only, and reliable model that is well-suited to the IMP architecture while still achieving high accuracy.

4 EXPERIMENTAL EVALUATION

In this section, we first provide a detailed overview of the hardware platforms, benchmark models, and datasets utilized in our experiments. Next, we evaluate the performance of our proposed integer-only quantization method. Subsequently, we conduct a comprehensive comparative evaluation of our crossbar-aligned pruning and integer-only quantization approach, which are co-optimized within the zerorized-recover training process. This evaluation is in comparison to state-of-the-art IMP-aware pruning methods in terms of sparsity rate, accuracy, area, and power. Finally, we evaluate the effectiveness of our overall comprehensive learning framework in achieving efficient, compact, and reliable DNN models for IMP devices.

4.1 Experiment Setup

The IMP accelerator used in this paper is based on the ISAAC architecture [42], which is a widely-adopted ReRAM-based IMP platform. The crossbar size is set to 128×128, and each memristor cell in this architecture can store 2 bits. We quantize the model to 8 bits, map each weight to 4 memristor cells and use the *S+A* unit in the device to combine the results from different cells. For components that ISAAC did not provide, such as *ST*, we use CACTI [38] at 32nm to model their power and area. The design configurations are simulated using the modified NeuroSim [5] simulator and under the 32nm CMOS library. The proposed training framework is implemented using the PyTorch framework [40]. We evaluate the proposed method on representative DNNs: LeNet [25], VGG [44], ResNet [14] and IMU [37] on four datasets: Mnist [10], Cifar-10 [24], Human Activity Recognition (HAR) [2] and ImageNet [41]. In line with previous works [1, 3, 9, 32, 43], the total training epochs we used are 200, 160, 160, and 90 for Mnist, Cifar-10, HAR, and ImageNet datasets, respectively. As introduced in Section 2, write variation stems from the stochastic characteristics of the ReRAM cell, which can render each cell potentially non-ideal. And the degree of error in a specific cell may vary across different configurations, thus, we have followed previous works [3, 11, 33] in employing $r' = e^{\theta} \cdot r$, and $\theta \sim N(0, \epsilon^2)$, where N represents a normal distribution with a mean

Table 2. Accuracy Comparison of Different Quantization Methods

Dataset	Network	Baseline Acc. (%)	Method	Quantized Acc. (%)	Acc. Drop (%)
Mnist	LeNet-5	99.21	IAO [19]	**99.10**	**0.11**
			CRIMP-Quan	99.09	0.12
Cifar-10	VGG-16	93.28	IAO [19]	93.14	0.14
			CRIMP-Quan	**93.39**	**−0.11**
	Resnet-56	93.34	IAO [19]	**93.44**	**−0.10**
			CRIMP-Quan	93.37	−0.03
HAR	IMU	97.22	IAO [19]	**97.01**	**0.21**
			CRIMP-Quan	96.95	0.27
ImageNet	Resnet-18	69.79	IAO [19]	**69.54**	**0.25**
			CRIMP-Quan	69.48	0.31

of 0 and variance of ϵ^2 to illustrate this error. This implies that θ can be selected randomly from $N(0, \epsilon^2)$. And we employ two commonly used $\epsilon = 0.1$ or $\epsilon = 0.5$ in previous works [3, 11, 33] for comparison. It is noteworthy that, as shown in Figure 4(b) when $\epsilon = 0.1$ and $H = 128$, the error introduced by write variation can be significantly mitigated by the rounding function in ADC. On the other hand, the SAF error emanates from the fabrication process, thus, the locations of SAF for a specific device are constant and can be recorded in advance [4]. Consistent with previous works [3, 20, 26, 29, 45], we train a model for the device using the known SAF locations. To simulate this process, in our experiment, the proportion of SF1 and SF0 cells to total cells is 9.04% and 1.75% respectively [3, 4]. And they keep a constant location during the training and testing.

4.2 Evaluation of Model Compact w/o Non-ideality

4.2.1 Accuracy Evaluation of Quantized Models. The performance evaluation of the proposed integer-only quantization approach (*CRIMP*-Quan) is demonstrated in Table 2, where we can find that our approach does not lead to a significant decrease in accuracy compared to the baseline (i.e., full precision), and even outperforms full precision in some models. Moreover, under the same training settings, the accuracy of our technique is similar to that of the IAO quantization method [19], which does not enforce scaling factors to be a power of 2. This highlights the effectiveness of our quantization scheme, which avoids FP multiplication operations while maintaining accuracy.

4.2.2 Accuracy Evaluation of Final Compact Models. Table 3 presents a comparison of our proposed crossbar-aligned pruning method (with integer-only quantization and completed by the zerorized-recover training scheme, represented by *CRIMP*-PQ) to state-of-the-art IMP-aware pruning methods, including LSRR [28], SPRC [35], XBA [27] and PIM-P [8]. We refer to the results reported in LSRR [28], SPRC [35], XBA [27], and re-implement PIM-P [8] for a comprehensive comparison. And in these methods, the effect of non-ideality is ignored. It should be noted that other methods still use FP scaling factors in their quantization schemes. To demonstrate the effectiveness of pruning in reducing the utilization of crossbars, we also provide the results of crossbar savings achieved by different pruning methods. It is important to note that the extent of crossbar savings is influenced by the size of the crossbar utilized. Therefore, we include the specific crossbar sizes employed in each of the evaluated works to provide a comprehensive understanding. The highest weight pruned or crossbar saved, highest accuracy, and lowest accuracy reductions are marked in **boldface**. From this table, we can find that for small datasets such as Mnist and Cifar-10, all methods can compress the model to reduce many weights with only a little accuracy

Table 3. Accuracy Comparison of Final Compact Models Without Non-ideality

Dateset	Network	Method	Baseline Acc. (%)	Crossbar Size	Weight Pruned (%)	Crossbar Saved (%)	Final Acc. (%)	Acc. Drop (%)
Mnist	LeNet-5	LSRR [28]	99.23	32 × 32	92.00	–	**99.15**	**0.08**
		CRIMP-PQ	99.21	128 × 128	**94.89**	**89.47**	98.90	0.31
Cifar-10	VGG-16	LSRR [28]	93.64	32 × 32	92.5	–	**93.72**	−0.08
		PIM-P [8]	93.28	128 × 128	87.30	83.61	93.22	0.06
		CRIMP-PQ	93.28	128 × 128	**88.25**	**88.52**	93.71	**−0.43**
	ResNet-56	SPRC [35]	–	144 × 32	–	33.40	92.80	–
		CRIMP-PQ	93.34	128 × 128	**51.36**	**48.22**	**93.18**	**0.16**
HAR	IMU	*CRIMP*-PQ	97.22	128 × 128	**87.20**	**86.59**	**96.40**	**0.82**
ImageNet	ResNet-18	XBA [27]	69.31	–	–	50.59	66.07	3.24
		SPRC [35]	69.76	144 × 32	22.33	–	67.50	2.26
		PIM-P [8]	69.79	128 × 128	32.41	30.93	68.67	1.12
		CRIMP-PQ	69.79	128 × 128	**50.50**	**51.32**	**68.82**	**0.97**

drop. Our method tries to remove more redundant weights from LeNet-5 on the Mnist dataset, which causes a larger accuracy drop than LSRR [28]. However, our method can save more crossbars on the VGG-16 model with even accuracy improvement, indicating that VGG-16 overfits the Cifar-10 dataset. For Resnet-56, we can still maximize the final accuracy.[2] In the case of the HAR dataset, which has a similar complexity as the Cifar-10 dataset, our method can still remove a substantial number of weights, with only a mild accuracy drop. This indicates that our approach has broad applicability and can effectively reduce the model size. For a large dataset like ImageNet, the sparsity rate achieved by all methods is relatively lower, but our method achieves a larger sparsity rate and higher accuracy. Overall, our proposed framework achieves comparable or better performance compared to state-of-the-art IMP-aware pruning methods, demonstrating its effectiveness in achieving compact and accurate DNNs for IMP devices without any hardware overhead.

4.2.3 Power & Area Evaluation of Final Compact Models. We employ the modified NeoroSim [5] simulator to measure the power and area consumption of models of various methods based on configuring to ISAAC architecture [42]. It should be noted that crossbar-column and crossbar-row pruning methods require the Sparsity Table unit for data alignment, and methods using FP scaling factors require FP processors. LSRR [28] and SPRC [35] use 4-bit quantization.[3] We normalize the power and area consumption to the total consumption of the original model. And we calculate the computing power and static power separately, as well as the computing area (crossbars, multipliers, and STs) and other areas (interconnect and memory) separately. From Figure 8, we can see that our method achieves the highest power and area efficiency improvement, especially for the computing parts. Compared to the original model, our method achieves an average total power reduction of 4.09× and computing power reduction of 100.02×, as well as a total area reduction of 2.88× and computing area reduction of 17.37×.

4.3 Evaluation of Whole Comprehensive Learning Framework

4.3.1 Evaluation of the Impact from Pruning & Non-ideality Adaptation. To investigate the impact of pruning and non-ideality adaptation (NIA) on the accuracy of DNN models when deployed on non-ideal ReRAM devices, we conducted an analysis of the changing trend of LeNet-5 accuracy

[2]SPRC does not report the baseline accuracy of Resnet-56 on Cifar-10.
[3]We use 4-bits quantization to model their power and area.

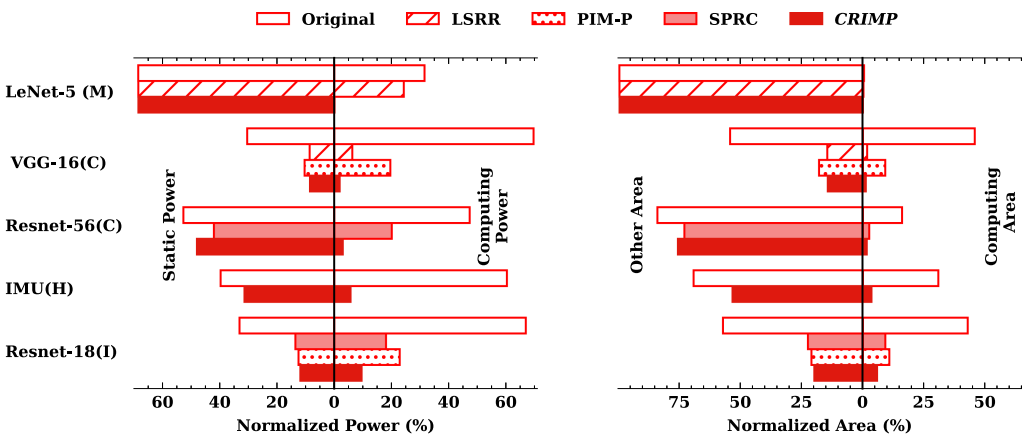


Fig. 8. Evaluation of the power and area consumption of different models from different methods.

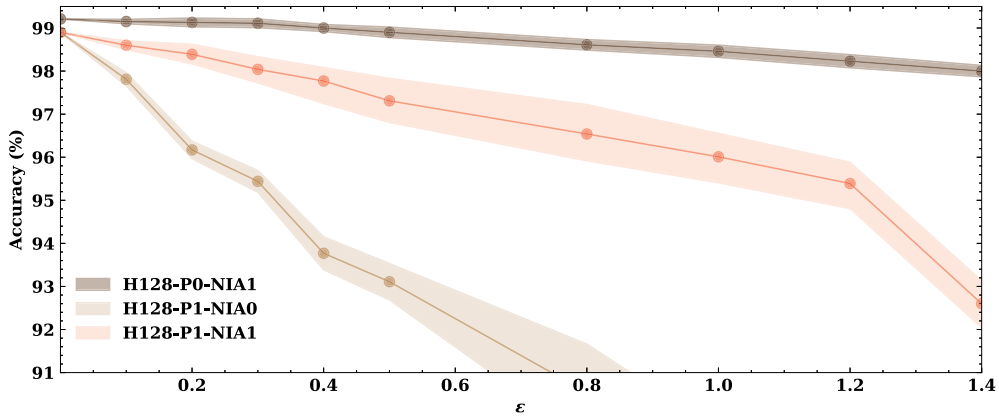


Fig. 9. The relationship between accuracy and ϵ under different conditions, including with or without pruning (P1 or P0), and with or without non-ideality adaptation (NIA1 or NIA0).

on the Mnist dataset under different degrees of non-ideality (by changing the ϵ in Equation (1)) without our self-compensation scheme, as shown in Figure 9. The pruning rate is the same as that presented in Table 3, where 94.89% of the weights are removed. As the non-ideality of ReRAM cells is random, we performed 20 inferences for each trained model, and the line in Figure 9 represents the average of 20 inferences, while the shaded part denotes the range of the accuracy. From this figure, we can find that without pruning, the accuracy of the network does not decrease significantly with the device's non-ideality increasing under NIA. Since Mnist is a small dataset and LeNet-5 has a high redundancy in the Mnist dataset, NIA training only needs to train most of the weights close to zero. Therefore, the unpruned redundant model is more tolerant of the device's non-ideality. In contrast, under network pruning, we found that NIA can increase the accuracy of the network on non-ideal devices. The accuracy without NIA dropped very quickly as the ϵ increases, and the average accuracy is only 90.54% with $\epsilon = 0.8$. However, the accuracy of NIA is 96.54%. This proves that NIA can improve the reliability of models.

4.3.2 Evaluation of Final Compact & Reliable Models for IMP. Table 4 presents a comparison of our proposed method with the state-of-the-art non-ideality processing methods, including

Table 4. Evaluation of Final Compact & Reliable Models for IMP; A. Means *Area*. W. means *Weight*

Dateset	Network	Method	Quan. Level	SAF	Varia. (ϵ)	Crossbar Saved (%)	Hardware Overhead	Runtime Overhead	Baseline Acc. (%)	ReRAM Acc. (%)	Acc. ↓ (%)
Mnist	LeNet-5	R-V-W [3]	16	✓	0.1	0	✗	Multi-Write	99.51	**99.24**	**0.27**
		AIIR [3]	16	✓	0.1	0	~15% A. ↑	Remapping	99.51	99.13	0.38
		KD [3]	16	✓	0.1	0	✗	✗	99.51	97.43	2.08
		Co.Net [11]	NA	✗	0.5	0	✗	~5% W. ↑	98.79	97.47	1.32
		CTSF [18]	8	✗	0.5	0	~10% A. ↑	✗	98.91	**98.67**	**0.24**
		CRIMP	8	✓	0.1	**89.47**	✗	✗	99.21	98.60	0.61
		CRIMP	8	✗	0.1	**89.47**	✗	✗	99.21	**99.02**	**0.19**
		CRIMP	8	✗	0.5	**82.61**	✗	✗	99.21	98.34	0.87
Cifar-10	VGG-16	R-V-W [3]	16	✓	0.1	0	✗	Multi-Write	93.35	88.43	4.92
		AIIR [3]	16	✓	0.1	0	~15% A. ↑	Remapping	93.35	**91.86**	**1.49**
		KD [3]	16	✓	0.1	0	✗	✗	93.35	87.13	6.22
		Co.Net [11]	NA	✗	0.5	0	✗	~0.58% W. ↑	93.20	91.29	**1.91**
		CTSF [18]	8	✗	0.5	0	~10% A. ↑	✗	93.21	91.06	2.15
		VACTSF [18]	8	✗	0.5	~9.03	~10% A. ↑	✗	93.21	86.75	6.46
		DRD [33]	8	✗	0.1	0	✗	✗	93.28	92.76	0.52
		CRIMP	8	✓	0.1	**88.52**	✗	✗	93.28	91.09	2.19
		CRIMP	8	✗	0.1	**88.52**	✗	✗	93.28	**93.20**	**0.08**
		CRIMP	8	✗	0.5	**83.64**	✗	✗	93.28	**91.35**	1.93
	ResNet-56	*CRIMP*	8	✓	0.1	**48.22**	✗	✗	93.34	**92.08**	**1.26**
		CRIMP	8	✗	0.1	**48.22**	✗	✗	93.34	**93.18**	**0.16**
		CRIMP	8	✗	0.5	**24.34**	✗	✗	93.34	**91.02**	**2.32**
HAR	IMU	*CRIMP*	8	✓	0.1	**86.59**	✗	✗	97.22	**93.35**	**3.87**
		CRIMP	8	✗	0.1	**86.59**	✗	✗	97.22	**93.60**	**3.62**
		CRIMP	8	✗	0.5	**79.71**	✗	✗	97.22	**92.53**	**4.69**

R-V-W [3], AIIR [3], KD [3], CorrectNet (Co.Net) [11], CTSF [18], VACTSF [18], and DRD [33]. We follow these methods to demonstrate the performance of our method on Mnist and Cifar-10 datasets, and the optimal values for each metric are indicated in **boldface**. To demonstrate the wide applicability of our method, we also conducted experiments on the HAR dataset. To ensure a fair comparison of accuracy, we report the quantization bit number (Quan. level), whether the methods consider SAF, the degree of variation (Varia. (ϵ)), and whether the models are compressed. Additionally, we report the overhead introduced by each method. To compare with each method, we evaluate our method under two degrees of device variation (0.1 and 0.5) and with and without SAF. Based on Figure 4(b), the non-ideality error for $\epsilon = 0.1$ can be mitigated through the rounding function in the ADC. Therefore, for $\epsilon = 0.1$, we do not employ the self-compensation scheme, as it introduces some read operations during the configuration process. However, for $\epsilon = 0.5$, we incorporated the self-compensation scheme, which also uses two additional cells for each weight to further diminish the error. It is important to note that the "crossbar saved" reported in Table 4 already includes the consideration of these two additional cells. For clear presentation, we do not include them in the "runtime overhead", as our pruning method reduces the number of weights, thereby offsetting the impact of these additional cells. Furthermore, for SFA non-ideality, we trained a model for a specific device using the known SAF locations, and show the evaluation results on this specific device.

From Table 4, we can conclude that, on the Mnist dataset, *CRIMP* only introduces a slight accuracy drop compared to the other methods, although *CRIMP* is with fewer quantization bits and model pruning, and without incurring overhead. Under the VGG-16 model in the Cifar-10 dataset, compared to R-V-W [3], AIIR [3], and KD [3], *CRIMP* achieves similar accuracy even with model compression, smaller quantization bit, and no introducing overhead. Compared to DRD [33], *CRIMP* considers the runtime characteristics of the crossbar to improve the accuracy significantly. Compared to VACTSF [18], which is CTSF [18] with pruning, *CRIMP* achieves a much larger

compression ratio and higher precision without introducing overhead. Furthermore, to demonstrate the applicability of *CRIMP* to different model architectures, we also show its results on the ResNet-56 and IMU models. And *CRIMP* can still achieve good performance. For SAF error, *CRIMP* is trained for a specific device with known SFA locations. Finally, *CRIMP* achieves compact and reliable models with a slight loss of accuracy (i.e., 2.19%, 1.26%, and 3.78% for VGG-16, ResNet-56, and IMU models under $\epsilon = 0.1$ and with SAF, respectively), which indicates it is a promising approach for deploying DNN on ReRAM IMP accelerators.

5 CONCLUSION

This work presented a comprehensive learning approach, named *CRIMP*, for achieving compact and reliable IMP inference acceleration. First, a multi-grained crossbar-aligned pruning method was introduced, which included kernel-group pruning and crossbar pruning to save crossbars without additional hardware. It is also further optimized by our dynamic zerorize-recover procedure for better architecture and weights, achieving higher accuracy. Second, a simple yet efficient integer-only quantization scheme was proposed to avoid using FP multipliers in IMP devices. Third, a runtime-aware non-ideality adaptation scheme was presented, using a realistic crossbar-based runtime simulator to learn reliable DNN models for IMP devices without any overhead. In addition, we designed a self-compensation scheme to further mitigate the errors caused by non-idealities. Finally, a novel learning framework was designed to complete these three schemes and co-optimize them, enhancing accuracy. The evaluation results showed that, compared to the original model, *CRIMP* achieved a 100.02× reduction in computing power and a 17.37× savings in computing area on average. Moreover, *CRIMP* obtained totally integer-only, pruned, and reliable VGG-16 and ResNet-56 models for the Cifar-10 dataset on ReRAM-based IMP devices with only 2.19% and 1.26% accuracy drops, respectively, without any additional hardware overhead.

REFERENCES

[1] Saud Aljaloud. 2023. Performance refinement of convolutional neural network architectures for solving big data problems. *Tikrit Journal of Pure Science* 28, 1 (2023), 89–95.

[2] Davide Anguita, Alessandro Ghio, Luca Oneto, Xavier Parra, Jorge Luis Reyes-Ortiz, et al. 2013. A public domain dataset for human activity recognition using smartphones.. In *Esann*, Vol. 3. 3.

[3] Gouranga Charan, Abinash Mohanty, Xiaocong Du, Gokul Krishnan, Rajiv V. Joshi, and Yu Cao. 2020. Accurate inference with inaccurate rram devices: A joint algorithm-design solution. *IEEE Journal on Exploratory Solid-State Computational Devices and Circuits* 6, 1 (2020), 27–35.

[4] Ching-Yi Chen, Hsiu-Chuan Shih, Cheng-Wen Wu, Chih-He Lin, Pi-Feng Chiu, Shyh-Shyuan Sheu, and Frederick T. Chen. 2014. RRAM defect modeling and failure analysis based on march test and a novel squeeze-search scheme. *IEEE Trans. Comput.* 64, 1 (2014), 180–190.

[5] Pai-Yu Chen, Xiaochen Peng, and Shimeng Yu. 2018. NeuroSim: A circuit-level macro model for benchmarking neuro-inspired architectures in online learning. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 37, 12 (2018), 3067–3080.

[6] Ping Chi, Shuangchen Li, Cong Xu, Tao Zhang, Jishen Zhao, Yongpan Liu, Yu Wang, and Yuan Xie. 2016. Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory. *ACM SIGARCH Computer Architecture News* 44, 3 (2016), 27–39.

[7] Jungwook Choi, Swagath Venkataramani, Vijayalakshmi Viji Srinivasan, Kailash Gopalakrishnan, Zhuo Wang, and Pierce Chuang. 2019. Accurate and efficient 2-bit quantized neural networks. *Proceedings of Machine Learning and Systems* 1 (2019), 348–359.

[8] Chaoqun Chu, Yanzhi Wang, Yilong Zhao, Xiaolong Ma, Shaokai Ye, Yunyan Hong, Xiaoyao Liang, Yinhe Han, and Li Jiang. 2020. PIM-prune: Fine-grain DCNN pruning for crossbar-based process-in-memory architecture. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

[9] PyTorch Examples Contributors. 2022. PyTorch Examples. https://github.com/pytorch/examples/tree/main/imagenet [Online; accessed 28-May-2023].

[10] Li Deng. 2012. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine* 29, 6 (2012), 141–142.

[11] Amro Eldebiky, Grace Li Zhang, Georg Boecherer, Bing Li, and Ulf Schlichtmann. 2022. CorrectNet: Robustness enhancement of analog in-memory computing for neural networks by error suppression and compensation. *arXiv e-prints* (2022), arXiv–2211.

[12] Sina Sayyah Ensan and Swaroop Ghosh. 2019. FPCAS: In-memory floating point computations for autonomous systems. In *2019 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 1–8.

[13] Jingyu He, Yucong Huang, Miguel Lastras, Terry Tao Ye, Chi-Ying Tsui, and Kwang-Ting Cheng. 2023. RVComp: Analog variation compensation for RRAM-based in-memory computing. In *Proceedings of the 28th Asia and South Pacific Design Automation Conference*. 246–251.

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 770–778.

[15] K. C. Hsu, Feng-Min Lee, Y. Y. Lin, E. K. Lai, J. Y. Wu, D. Y. Lee, Min-Hee Lee, H. L. Lung, K. Y. Hsieh, and C. Y. Lu. 2015. A study of array resistance distribution and a novel operation algorithm for WOx ReRAM memory. In *SSDM*.

[16] Shuo Huai, Di Liu, Hao Kong, Weichen Liu, Ravi Subramaniam, Christian Makaya, and Qian Lin. 2023. Latency-constrained DNN architecture learning for edge systems using zerorized batch normalization. *Future Generation Computer Systems* 142 (2023), 314–327.

[17] Shuo Huai, Di Liu, Xiangzhong Luo, Hui Chen, Weichen Liu, and Ravi Subramaniam. 2023. Crossbar-aligned & integer-only neural network compression for efficient in-memory acceleration. In *Proceedings of the 28th Asia and South Pacific Design Automation Conference*. 234–239.

[18] Chenglong Huang, Nuo Xu, Junwei Zeng, Wenqing Wang, Yihong Hu, Liang Fang, Desheng Ma, and Yanting Chen. 2022. Rescuing ReRAM-based neural computing systems from device variation. *ACM Transactions on Design Automation of Electronic Systems* 28, 1 (2022), 1–17.

[19] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. 2018. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2704–2713.

[20] Giju Jung, Mohammed Fouda, Sugil Lee, Jongeun Lee, Ahmed Eltawil, and Fadi Kurdahi. 2021. Cost-and dataset-free stuck-at fault mitigation for ReRAM-based deep learning accelerators. In *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1733–1738.

[21] William Kahan. 1996. IEEE standard 754 for binary floating-point arithmetic. *Lecture Notes on the Status of IEEE* 94720-1776 (1996), 11.

[22] Michael Klachko, Mohammad Reza Mahmoodi, and Dmitri Strukov. 2019. Improving noise tolerance of mixed-signal neural networks. In *2019 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 1–8.

[23] Hao Kong, Di Liu, Shuo Huai, Xiangzhong Luo, Weichen Liu, Ravi Subramaniam, Christian Makaya, and Qian Lin. 2022. Smart scissor: Coupling spatial redundancy reduction and CNN compression for embedded hardware. In *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*. 1–9.

[24] Alex Krizhevsky, Geoffrey Hinton, et al. 2009. Learning multiple layers of features from tiny images. (2009).

[25] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.

[26] Wen Li, Ying Wang, Huawei Li, and Xiaowei Li. 2019. RRAMedy: Protecting ReRAM-based neural network from permanent and soft faults during its lifetime. In *2019 IEEE 37th International Conference on Computer Design (ICCD)*. IEEE, 91–99.

[27] Ling Liang, Lei Deng, Yueling Zeng, Xing Hu, Yu Ji, Xin Ma, Guoqi Li, and Yuan Xie. 2018. Crossbar-aware neural network pruning. *IEEE Access* 6 (2018), 58324–58337.

[28] Jilan Lin, Zhenhua Zhu, Yu Wang, and Yuan Xie. 2019. Learning the sparsity for ReRAM: Mapping and pruning sparse neural network for ReRAM based accelerator. In *Proceedings of the 24th Asia and South Pacific Design Automation Conference*. 639–644.

[29] Chenchen Liu, Miao Hu, John Paul Strachan, and Hai Li. 2017. Rescuing memristor-based neuromorphic design with high defects. In *Proceedings of the 54th Annual Design Automation Conference 2017*. 1–6.

[30] Chenchen Liu, Fuxun Yu, Zhuwei Qin, and Xiang Chen. 2020. Enabling efficient ReRAM-based neural network computing via crossbar structure adaptive optimization. In *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design*. 133–138.

[31] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. 2017. Learning efficient convolutional networks through network slimming. In *Proceedings of the IEEE International Conference on Computer Vision*. 2736–2744.

[32] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. 2017. Learning efficient convolutional networks through network slimming. In *Proceedings of the IEEE International Conference on Computer Vision*. 2736–2744.

[33] Yun Long, Xueyuan She, and Saibal Mukhopadhyay. 2019. Design of reliable DNN accelerator with un-reliable ReRAM. In *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1769–1774.
[34] Huizi Mao, Song Han, Jeff Pool, Wenshuo Li, Xingyu Liu, Yu Wang, and William J. Dally. 2017. Exploring the granularity of sparsity in convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 13–20.
[35] Jian Meng, Li Yang, Xiaochen Peng, Shimeng Yu, Deliang Fan, and Jae-Sun Seo. 2021. Structured pruning of RRAM crossbars for efficient in-memory computing acceleration of deep neural networks. *IEEE Transactions on Circuits and Systems II: Express Briefs* 68, 5 (2021), 1576–1580.
[36] Ziqi Meng, Weikanu Oian, Yilonz Zhao, Yanan Sun, Rui Yang, and Li Jiang. 2021. Digital offset for rram-based neuromorphic computing: A novel solution to conquer cycle-to-cycle variation. In *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1078–1083.
[37] Fernando Moya Rueda, René Grzeszick, Gernot A. Fink, Sascha Feldhorst, and Michael Ten Hompel. 2018. Convolutional neural networks for human activity recognition using body-worn sensors. In *Informatics*, Vol. 5. MDPI, 26.
[38] Naveen Muralimanohar, Rajeev Balasubramonian, and Norman P. Jouppi. 2009. CACTI 6.0: A tool to model large caches. *HP Laboratories* 27 (2009), 28.
[39] Michael A. Nielsen. 2015. *Neural Networks and Deep Learning*. Vol. 25. Determination press San Francisco, CA, USA.
[40] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An imperative style, high-performance deep learning library. In *NIPS*. 8024–8035.
[41] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. 2015. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision* 115 (2015), 211–252.
[42] Ali Shafiee, Anirban Nag, Naveen Muralimanohar, Rajeev Balasubramonian, John Paul Strachan, Miao Hu, R. Stanley Williams, and Vivek Srikumar. 2016. ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars. *ACM SIGARCH Computer Architecture News* 44, 3 (2016), 14–26.
[43] Zhouxing Shi, Yihan Wang, Huan Zhang, Jinfeng Yi, and Cho-Jui Hsieh. 2021. Fast certified robust training with short warmup. *Advances in Neural Information Processing Systems* 34 (2021), 18335–18349.
[44] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).
[45] Lixue Xia, Mengyun Liu, Xuefei Ning, Krishnendu Chakrabarty, and Yu Wang. 2017. Fault-tolerant training with online fault detection for RRAM-based neural computing systems. In *Proceedings of the 54th Annual Design Automation Conference 2017*. 1–6.
[46] Shimeng Yu and Pai-Yu Chen. 2016. Emerging memory technologies: Recent trends and prospects. *IEEE Solid-State Circuits Magazine* 8, 2 (2016), 43–56.

Received 23 March 2023; revised 2 June 2023; accepted 13 July 2023