DDC-PIM: Efficient Algorithm/Architecture Co-design for Doubling Data Capacity of SRAM-based Processing-In-Memory

Cenlin Duan, Jianlei Yang, *Senior Member, IEEE,* Xiaolin He, Yingjie Qi, Yikun Wang, Yiou Wang, Ziyan He,
Bonan Yan, *Member, IEEE,* Xueyan Wang, *Member, IEEE,* Xiaotao Jia, *Member, IEEE,*
Weitao Pan, *Member, IEEE,* and Weisheng Zhao, *Fellow, IEEE*

Abstract—Processing-in-memory (PIM), as a novel computing paradigm, provides significant performance benefits from the aspect of effective data movement reduction. SRAM-based PIM has been demonstrated as one of the most promising candidates due to its endurance and compatibility. However, the integration density of SRAM-based PIM is much lower than other non-volatile memory-based ones, due to its inherent 6T structure for storing a single bit. Within comparable area constraints, SRAM-based PIM exhibits notably lower capacity. Thus, aiming to unleash its capacity potential, we propose DDC-PIM, an efficient algorithm/architecture co-design methodology that effectively doubles the equivalent data capacity. At the algorithmic level, we propose a filter-wise complementary correlation (FCC) algorithm to obtain a bitwise complementary pair. At the architecture level, we exploit the intrinsic cross-coupled structure of 6T SRAM to store the bitwise complementary pair in their complementary states $(Q/\overline{Q})$, thereby maximizing the data capacity of each SRAM cell. The dual-broadcast input structure and reconfigurable unit support both depthwise and pointwise convolution, adhering to the requirements of various neural networks. Evaluation results show that DDC-PIM yields about $2.84\times$ speedup on MobileNetV2 and $2.69\times$ on EfficientNet-B0 with negligible accuracy loss compared with PIM baseline implementation. Compared with state-of-the-art SRAM-based PIM macros, DDC-PIM achieves up to $8.41\times$ and $2.75\times$ improvement in weight density and area efficiency, respectively.

Index Terms—Processing-In-Memory, Algorithm/Architecture Co-design, Doubling Data Capacity, SRAM-PIM

I. INTRODUCTION

DEEP Neural Networks (DNNs) are ubiquitous in a variety of applications, such as image recognition [1–3], speech recognition [4, 5], and object detection [6–8]. To achieve higher accuracy, an intuitive approach is to design deeper and more intricate network models. However, the complexity

C. Duan, X. Wang, X. Jia and W. Zhao are with BDBC, Fert Beijing Research Institute, School of Integrated Circuit Science and Engineering, Beihang University, Beijing, 100191, China. E-mail: weisheng.zhao@buaa.edu.cn

J. Yang, X. He, Y. Qi, Yikun Wang and Yiou Wang are with BDBC, Fert Beijing Research Institute, School of Computer Science and Engineering, Beihang University, Beijing, 100191, China. E-mail: jianlei@buaa.edu.cn

Z. He and W. Pan are with School of Telecommunications Engineering, Xidian University, Xi'an, 710071, China.

B. Yan is with Institute of Artificial Intelligence, Peking University, Beijing, 100871, China.
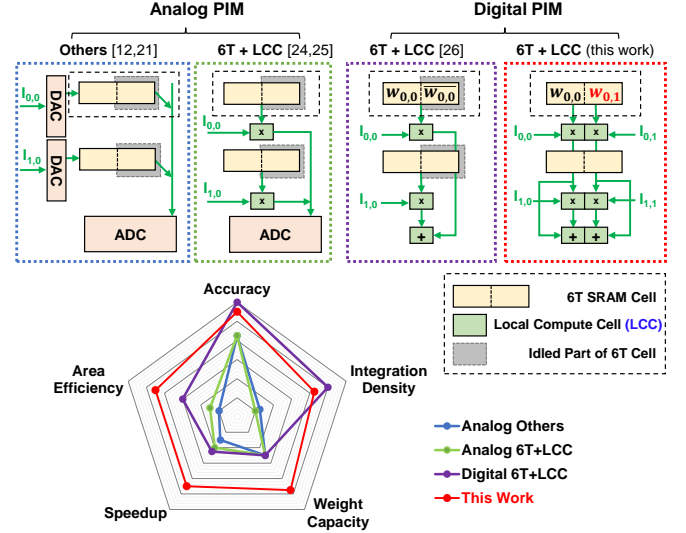
Fig. 1. Typical principles demonstration and comparison among several state-of-the-art 6T SRAM-based PIM studies.

of these models makes them difficult to deploy on edge devices, such as wearable devices, medical equipment, and mobile devices. To tailor for mobile and resource-constrained environments, there is a growing interest in building small and efficient neural networks (NNs), such as MobileNet [9] and EfficientNet-B0 [10]. These compact NNs usually decompose standard convolution into separable convolution, significantly reducing the number of parameters and computations with negligible accuracy loss.

While compact NNs offer the potential for enhanced efficiency, the inherent data-intensive nature of such networks continues to present challenges for edge devices. This is particularly evident in edge devices built on conventional Von Neumann architectures, as the deployment on such devices will bring in large energy consumption, due to frequent data movements between memory and computing units. With the saturation of Moore's law, reducing energy consumption becomes progressively more difficult. As a new architectural paradigm, Processing-in-memory (PIM) performs Multiply-Accumulate (MAC) operations in memory to eliminate the *Memory Wall* bottleneck, making it a promising candidate architecture for energy-efficient edge devices. As illustrated in Fig. 1, PIM macros are usually categorized into analog PIM and digital PIM [11–14]. Analog PIM usually performs

MAC operations in the voltage or current domain [11, 12]. However, the extra ADC/DAC [13] and process variation often lead to low area efficiency, low computation parallelism, and non-negligible accuracy loss.

To tackle the above limitations, digital PIM is further proposed by performing bitwise operations [14]. Generally, digital PIM integrates logic units into a single cell array, simultaneously activating all rows to improve computing parallelism. Previous studies have shown that various technologies, such as RRAM [15, 16], SRAM and MRAM [17–20], are available candidates for digital PIM. Among these technologies, SRAM is widely used in academia and industry due to its faster write speed, lower write energy and compatibility with proven process technologies. However, SRAM suffers from a low integration density compared to non-volatile memory. The conventional structure of SRAM cells requires six transistors to store one bit in two cross-coupled inverters, whereas non-volatile memory is capable of storing multiple bits with fewer transistors. Thus, inhibited by the integration density, the capacity has become a crucial limitation for SRAM-based PIM.

From the essential observations of adopted cross-coupled inverters in SRAM cells, a pair of complementary states (denoted as Q and $\overline{Q}$) is used for storing a single bit. In practice, only one state within the same pair will participate in the computation regarding the bit it represents. Based on this observation, we seek to promote the representational power of complementary pairs by treating each state as an independent bit of information. To realize this vision, the data organization must be adjusted correspondingly at the algorithmic level. Hence, storing the weights of compact NNs in this structure should embody the above complementary characteristic. Based on this cooperation between algorithm and architecture, the *weight density* (i.e., weight capacity per area) will achieve a twofold increase. As shown in Fig. 2, this approach is equivalent to using three transistors for storing one bit, which offers us an opportunity to boost the capacity of SRAM-based PIM. As the increase in overall area for supporting the above approach is negligible, the weight density achieves remarkable improvement compared with prior works. Moreover, compared to [14] with a PIM macro similar to ours, the area-efficiency of DDC-PIM also approximately doubles due to improved computation parallelism.

In this work, we propose DDC-PIM, an efficient algorithm/architecture co-design framework that comprises a novel DDC-PIM architecture and a Filter-wise Complementary Correlation (FCC) algorithm. Essentially, DDC-PIM doubles the weight density without modifying the SRAM structure, leading to higher equivalent data capacity and area efficiency. Our main contributions can be summarized as follows:

- We propose a novel DDC-PIM architecture, the first in-memory architecture for doubling the equivalent data capacity of SRAM-based PIM. By fully exploiting the intrinsic cross-coupled structure, DDC-PIM overcomes the limitation of low integration density of SRAM.
- We propose a FCC algorithm to obtain bitwise complementary filters. This method can reduce the memory footprint/computation latency and improve the equivalent
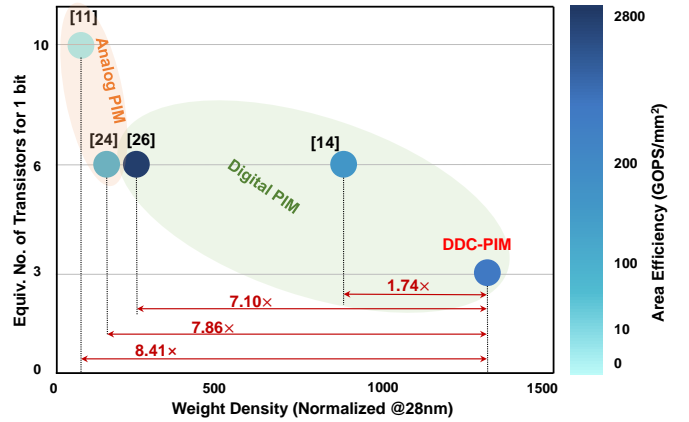

Fig. 2. Normalized weight density improvement and area efficiency comparison with prior SRAM-based PIM solutions.

data transfer bandwidth of NNs with negligible accuracy loss.
- A dual-broadcast input structure and a reconfigurable unit, together with a flexible mapping strategy are co-optimized to achieve a high PIM array utilization rate.

The rest of this paper is organized as follows. Section II provides background and motivations. Section III demonstrates the details of the proposed methodology. Section IV illustrates the experimental results. Related discussions are illustrated in Section V and conclusion remarks are given in Section VI.

II. BACKGROUND AND MOTIVATIONS

A. Requirements from Compact NN Models

Modern state-of-the-art NN models usually necessitate lots of memory and computational resources that are easily beyond the capabilities of typical edge devices. Hence, there is a growing interest in developing compact and efficient NN models. One of the mainstream approaches is to obtain smaller models in decomposed convolution manner, minimizing the scale of parameters and computations. Among them, MobileNet and EfficientNet are two typical compact NN models, which inherently possess significantly lower levels of redundancy compared to regular NNs. MobileNet decomposes *standard convolution* (std-conv) into *pointwise convolution* (pw-conv) and *depthwise convolution* (dw-conv), which are usually collectively referred to as *separable convolution* (sp-conv) [9]. For std-conv, convolution filters are considered three-dimensional, including a channel dimension and two spatial dimensions. For sp-conv, each input channel is assigned with a single filter in dw-conv for spatial correlation, the results of which are then combined through 1×1 convolutions in pw-conv for channel correlation. Thus, the amount of computation in sp-conv is only about $1/8$ of std-conv. Following the idea of sp-conv in MobileNet, EfficientNet uniformly scales all dimensions of *depth*, *width* and *resolution*, using highly effective compound coefficients to maximize the model efficiency [10]. EfficientNet can usually obtain a faster inference speed and better accuracy with a smaller model size than the most popular convolutional NN models.

B. Capacity Issues of SRAM-based PIM

For earlier SRAM-based PIM macros [12, 21], as illustrated in Fig. 1 (*denoted by* **Analog Others**), input features are first converted into analog signals through digital to analog converters (DACs), and then taking part in analog MAC operations with the pre-stored weights in SRAM cells. The results are then converted back to the digital domain by analog to digital converter (ADCs). In general, multiple cell rows in PIM array would activate simultaneously to increase computational parallelism. However, this approach will introduce *read disturbance* due to the expanded voltage swing of the bitline. One possible solution to this issue is to increase the bitline voltage, sacrificing the signal margin of read operation. Another approach is to use non-6T structure SRAM, such as 8T, 10T, or 12T, which also brings great area overhead [11, 22, 23]. Some researchers have proposed LCC-based (Local Compute Cell-based) PIM macros as shown in Fig. 1 (*denoted by* **Analog 6T+LCC**) to resolve the above limitations [24, 25], where LCC means the newly integrated *local computing cells*. The performance of these PIM architectures is often limited by kinds of analog-specific non-idealities, including process variations and tremendous area/power overhead due to essential data conversions. Moreover, handicapped by the accuracy of ADCs, only a limited number of cell rows can be activated per cycle, leading to low PIM array utilization and high computation latency.

In contrast to analog PIM solutions which perform computations in the voltage or current domain, digital PIM incorporates logic units within a single-cell array to execute digital logic operations. This approach enhances the accuracy, area efficiency, and energy efficiency of the system, as demonstrated by previous studies [26–31]. Yu et al. [26] propose an all-digital SRAM-based full-precision PIM macro. Yue et al. [28] propose a floating-point CIM processor for NN inference/training applications. Yan et al. [14] propose an ADC-less SRAM-based PIM with reconfigurable bitwise operations as shown in Fig. 1 (*denoted by* **Digital 6T+LCC**). However, only half of the 6T bitcells within these PIM macros contribute to the MAC computations, due to that a single bit is represented by two complementary states in the cross-coupled structure. As shown in the upper right of Fig. 1 (*denoted by* **this work**), our intuitive approach is to treat each state as an independent bit of information to fully utilize these bitcells for *doubling the equivalent data capacities*. When a pair of computation units are connected with the cross-coupled structure, two independent AND operations can also be performed simultaneously, providing opportunities for *doubling computation parallelism*. As a result illustrated in the radar plot from Fig. 1, our work achieves remarkable enhancements in area efficiency, weight density and speed up, with minor trade-offs in integrated density and accuracy.

III. METHODOLOGIES

This section presents the proposed DDC-PIM in detail, including the FCC algorithm, DDC-PIM architecture, and data mapping method, which seeks to harness the capacity potential and improve the area efficiency of digital 6T SRAM PIM.
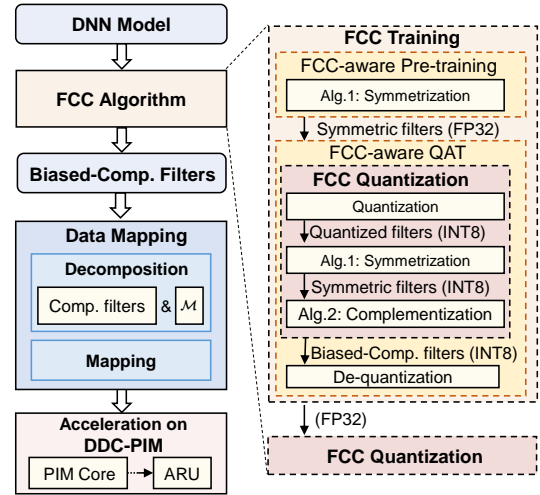

Fig. 3. Overview of proposed DDC-PIM framework.

A. Overall Framework

Fig. 3 illustrates the overview of the DDC-PIM framework, consisting of three major components: FCC algorithm, DDC-PIM architecture, and data mapping method. For compact NNs, DDC-PIM first transforms the weights into a biased bitwise complementary format by the two-stage FCC algorithm, comprising FCC training and FCC quantization. During data mapping, the results of FCC algorithm are decomposed into bitwise complementary filters and corresponding mean values layer-by-layer. The per-layer configuration signals are also generated offline during this procedure. Due to the complementary nature in these filters, only half of the bitwise complementary filters are required for storage and transmission. DDC-PIM then performs MAC operations with these filters on the PIM core. The convolution results are recovered from the MAC outputs and the mean value using an accumulate and recover unit (ARU). This framework can double data capacity and reduce computation latency by exploiting the bitwise complementary nature of weights. Meanwhile, only half of the complementary filters are required during data transmission, equivalently increasing the data transfer bandwidth. Subsequently, we will detail the three main components of DDC-PIM.

B. FCC Algorithm

For simplicity, we first introduce the notations that will facilitate the subsequent exposition, as illustrated in Tab. I.

A convolutional (Conv) layer applies N 3-dimensional (3D, $K \times K \times C$) filters ($\mathcal{F}$) on 3D ($H \times W \times C$) input feature maps ($\mathcal{I}$) to extract embedded characteristics and generate the output feature maps ($\mathcal{O}$). H and W denote the height/width of $\mathcal{I}$. C and N denote the number of channels of $\mathcal{I}$ and $\mathcal{O}$, respectively. K denotes the kernel size. As illustrated in Fig. 4, we use two adjacent filters (f_0 and f_1) with the size of $2 \times 2 \times 1$ as an example, where $\mathcal{M}_0$ denotes the mean value of f_0 and f_1. The weights located in the same position of these two filters are denoted as *twin-weights*, i.e. $w_{i,j}$ and $w_{i,j+1}$, where $i \in [0, 1, 2, \ldots, K \times K \times C - 1]$ and $j \in [0, 2, 4, \ldots, N-2]$.
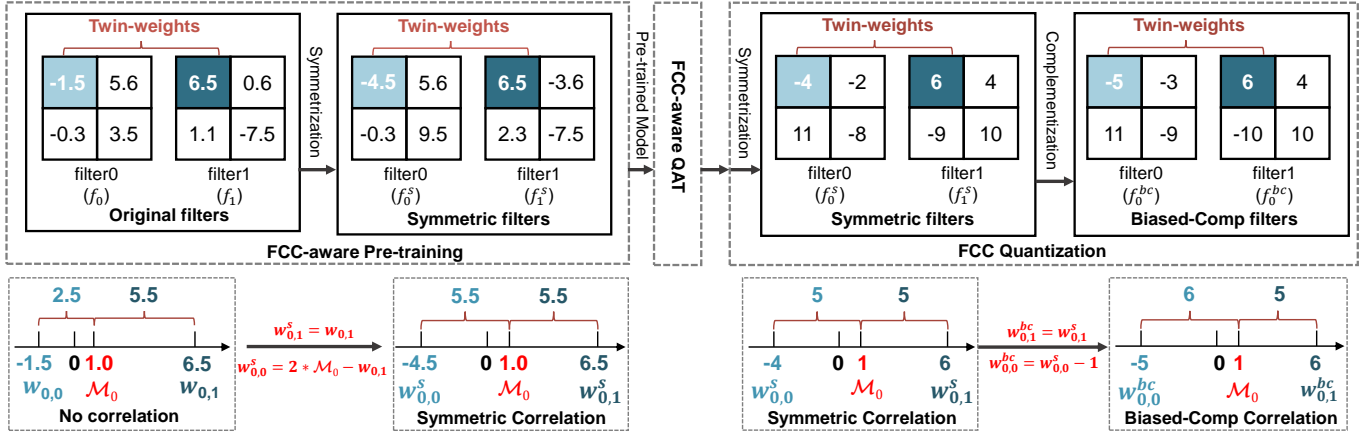
Fig. 4. Demonstration of FCC algorithm.

TABLE I
NOTATIONS OF THE INVOLVED MATHEMATICAL SYMBOLS AND
CORRESPONDING DESCRIPTIONS.

Term	Description
$\mathcal{I}$	input feature map
$\mathcal{O}$	output feature map
$\mathcal{F}$	original filters
$\mathcal{F}^s$	symmetric filters
$\mathcal{F}^{bc}$	biased-comp filters
$f_j^{(s)}$	j^{th} filter with symmetric correlation
$f_j^{(c)}$	j^{th} filter with complementary correlation
$f_j^{(bc)}$	j^{th} filter with bias-complementary correlation
$w_{i,j}$	i^{th} weight in j^{th} filter
$\mathcal{M}$	mean value of a pair of adjacent filters
$*$	convolution operation function
$\sim$	bitwise complement operation

And a filter pair comprising twin-weights with symmetric correlation is denoted as *symmetric filters*. More precisely, twin-weights in *symmetric filters* satisfy:

$$w_{i,j}^s - \mathcal{M} = -(w_{i,j+1}^s - \mathcal{M}). \tag{1}$$

A filter pair composed of bitwise complementary twin-weights is denoted as *complementary filters (Comp filters)*, which satisfies:

$$w_{i,j}^c = \sim w_{i,j+1}^c. \tag{2}$$

A pair of complementary filters with a fixed bias (equals to $\mathcal{M}$) is denoted as *biased-complementary filters (Biased-Comp filters)*, which satisfies:

$$w_{i,j}^{bc} - \mathcal{M} = \sim (w_{i,j+1}^{bc} - \mathcal{M}). \tag{3}$$

By storing the complementary bits ($w_{i,j}^c[k]$ and $w_{i,j+1}^c[k]$, $k \in [0, 1, 2, \ldots, 7]$) in the cross-coupled structure of 6T SRAM, we only need to transfer half of the filters and a group of $\mathcal{M}$. Thus, the data transfer bandwidth can be equivalently improved by about $2\times$ with little extra overhead.

FCC algorithm includes FCC training and FCC quantization, as shown in Fig. 3. FCC training procedure mainly

---

Algorithm 1: Symmetrization

Input: Original or quantized filters
$\mathcal{F} \doteq [f_0, \ldots, f_{N-1}]$, where C/N are the numbers of input/output channels, K is kernel size.

Output: Symmetric filters $\mathcal{F}^s \doteq [f_0^s, \ldots, f_{N-1}^s]$.

1   $\mathcal{L} \leftarrow K \times K \times C$
2   **for** j in $[0, 2, 4, \ldots, N-1]$ **do**
3     $sum_j \leftarrow (\sum f_j + \sum f_{j+1})$
4     $\mathcal{M}_j \leftarrow sum_j/(2 \times \mathcal{L})$
5   **if** $|f_j(\cdot) - \mathcal{M}_j| \geq |f_{j+1}(\cdot) - \mathcal{M}_j|$ **then**
6       $f_j^s(\cdot) \leftarrow f_j(\cdot)$
7       $f_{j+1}^s(\cdot) \leftarrow (2 \times \mathcal{M}_j - f_j(\cdot))$
8   **else**
9       $f_j^s(\cdot) \leftarrow (2 \times \mathcal{M}_j - f_{j+1}(\cdot))$
10      $f_{j+1}^s(\cdot) \leftarrow f_{j+1}(\cdot)$
11   **end**
12 **end**

---

consists of two steps, FCC-aware Pre-training and FCC-aware QAT, as following.

1) FCC-aware Pre-training: We pre-train the NN models to acquire *symmetric filters* through *Symmetrization* during pre-training, as shown in Alg. 1. Considering bitwise operations are usually incompatible with floating point numbers, FCC uses negations to approximate the bitwise complementary operations during pre-training. As shown in Eq. 4, the opposite value of an integral number is similar to its bitwise complement, making it a viable solution for approximation before quantization.

$$-w_{i,j} = \sim w_{i,j} + 1. \tag{4}$$

However, the constraint of symmetrizing all filter pairs to zero is too rigid for NN models to extract enough information during training. Hence, it will introduce drastic accuracy loss. For instance, such a restriction resulted in MobileNetV2 accuracy plummeted from 97.05% to 71.65%. To alleviate the impact of such restriction, we propose to symmetrize filter pairs to their mean value $\mathcal{M}$ for more flexibility. Fig. 4

Algorithm 2: Complementization

Input: Symmetric filters $\mathcal{F}^s \doteq \left[f_0^s, \ldots, f_{N-1}^s\right]$, where N is the number of output channels.

Output: Biased-Comp filters $\mathcal{F}^{bc} \doteq \left[f_0^{bc}, \ldots, f_{N-1}^{bc}\right]$.

1 **for** j in $[0, 2, 4, \ldots, N-2]$ **do**
2   **if** $f_j^s(\cdot) \geq f_{j+1}^s(\cdot)$ **then**
3 $f_{j+1}^{bc}(\cdot) \leftarrow \left(f_{j+1}^s(\cdot) - 1\right)$
4 $f_j^{bc}(\cdot) \leftarrow f_j^s(\cdot)$
5   **else**
6 $f_j^{bc}(\cdot) \leftarrow \left(f_j^s(\cdot) - 1\right)$
7 $f_{j+1}^{bc}(\cdot) \leftarrow f_{j+1}^s(\cdot)$
8   **end**
9 **end**

Fig. 5. Top level architecture design of DDC-PIM.

demonstrates an illustrative example of our proposed FCC algorithm. The procedure of FCC-aware pre-training mainly consists of the following two steps:

① First, we initialize the original filters by randomly sampling a normal distribution and then calculate $\mathcal{M}$ for each adjacent filter pair. In this case, $\mathcal{M}_0 = 1.0$, $w_{0,0} = -1.5$, and $w_{0,1} = 6.5$.

② Then, we symmetrize these twin-weights according to $\mathcal{M}$ by replacing the weight closer to $\mathcal{M}$ with the mirror image of the other, as shown in Fig. 4. After symmetrization, $w_{0,0}^s$ and $w_{0,1}^s$ are represented as -4.5 and 6.5, respectively.

As a result of pre-training, the obtained twin-weights in *symmetric filters* exhibit an opposite correlation when $\mathcal{M}$ is extracted, which satisfies:

$$w_{0,0}^s - \mathcal{M}_0 = -(w_{0,1}^s - \mathcal{M}_0). \tag{5}$$

2) FCC-aware QAT: Quantization-aware training (QAT) simulates the quantization effect by applying quantization and subsequent de-quantization during the training process. In order to perceive the impact of complementary operations on model accuracy, we modify QAT to a FCC-aware manner for obtaining quantization parameters of the above pre-trained model. Specifically, the procedure of FCC-aware QAT includes the following four steps:

① *Quantization*: We quantize a floating point model (*symmetric filters*) to an 8-bit precision model (*quantized filters*) for calculating the quantization parameters.

② *Symmetrization*: On account of the weakened symmetric correlation within the *quantized filters*, we perform symmetrization for a second time. When symmetrization is performed, $\mathcal{M}$ is rounded to ensure that $\mathcal{M}$ is an integer. As illustrated in Fig. 4, after quantization and symmetrization, $w_{0,0}^s = -4$, $w_{0,1}^s = 6$, and $\mathcal{M}_0 = 1$.

③ *Complementization*: Due to the relationship between the opposite and the bitwise complementary value of $w_{i,j}$ presented in Eq. 4, we exert *Complementization* on *symmetric filters* (INT8) to obtain *Biased-Comp filters*, as shown in Alg. 2. Specifically, we subtract 1 from the smaller twin-weight in *symmetric filters* to obtain the bitwise complementary correlation after $\mathcal{M}$ is extracted.
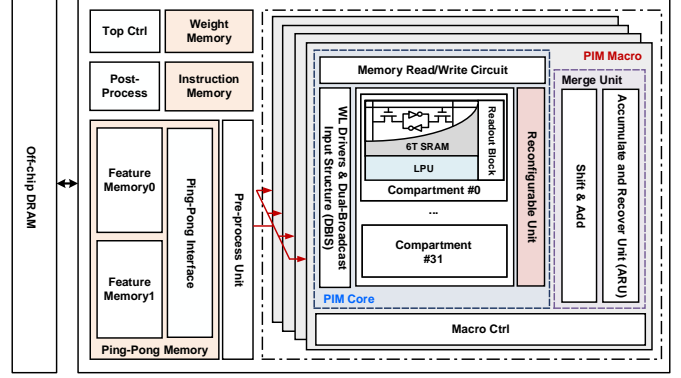
After complementization, $w_{0,0}^{bc}$ and $w_{0,1}^{bc}$ become -5 and 6 respectively.

④ *De-quantization*: We perform de-quantization operation on *Biased-Comp filters*, in order to ensure accurate gradient calculations and parameter updates.

We refer to the above process of quantization, symmetrization and complementization as FCC quantization. At the end of FCC training, we perform FCC quantization again to obtain *biased-comp filters*. The FCC algorithm leverages the complementary characteristics of weights in Conv layers to accommodate the DDC-PIM architecture, which offers a novel paradigm for deep learning computation. However, the fully connected (FC) layers do not benefit from this algorithm as much as the Conv layers do, which will be detailed in Section IV. Therefore, we exclude the FC layers from the scope of the FCC algorithm. After the biased-comp. filters generation module, the *Biased-Comp filters* will be decomposed and mapped onto the DDC-PIM architecture, which enables parallel computation by employing the cross-coupled structures of 6T SRAM.

C. Architecture Design of DDC-PIM

1) Top level architecture: Fig. 5 shows the overall architecture of DDC-PIM, which is composed of a top controller, a pre-process unit, four PIM macros, an instruction memory, a weight memory, a ping-pong memory, and a post-process unit. The PIM macro is an extension of the ADC-less SRAM PIM macro proposed in [14]. The DDC-PIM is interfaced with an off-chip DRAM that stores the input features and weights of a neural network. To accomplish the execution of a convolutional layer, the input features, weights and instructions are fetched from off-chip DRAM to on-chip memories. After transmission, the top controller first processes instructions fetched from instruction memory and sends corresponding control signals to the whole system. $\mathcal{I}$ stored in ping-pong memory can be accessed by pre-process unit for converting into bit-serial form and broadcasting to four PIM macros. The PIM core in each macro receives weights from *weight memory* and performs bitwise MAC operation with $\mathcal{I}$ to generate the intermediate results. These results are shifted and accumulated by shift & add unit based on their respective bit position for producing the *partial sum* (Psum). Then, we accumulate

Config	INP	INN	$w^c_{0,0}[0]$	$w^c_{0,1}[0]$	$O_{ch0}[0] =$ $INP\&w^c_{0,0}[0]$	$O_{ch1}[0]=$ $INN\&w^c_{0,1}[0]$
std-conv / pw-conv	0	0	0	1	0	0
			1	0	0	0
	1	1	1	0	1	0
			0	1	0	1
dw-conv	0	0	0	1	0	0
			1	0	0	0
	0	1	0	1	0	1
			1	0	0	0
	1	0	0	1	0	0
			1	0	1	0
	1	1	0	1	0	1
			1	0	1	0

(a) Truth Table   **(b) Double Bitwise Multiply Unit**   **(c) PIM Core**
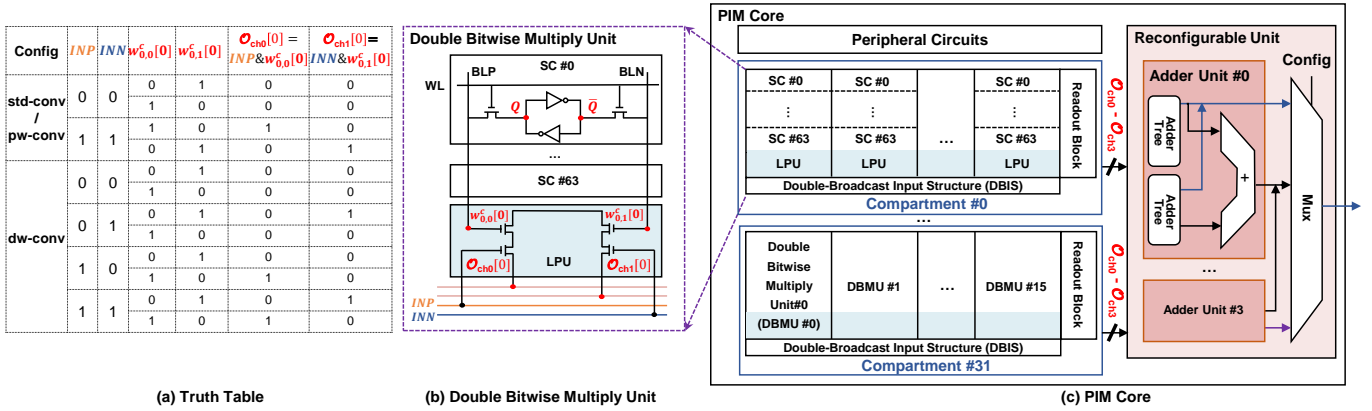
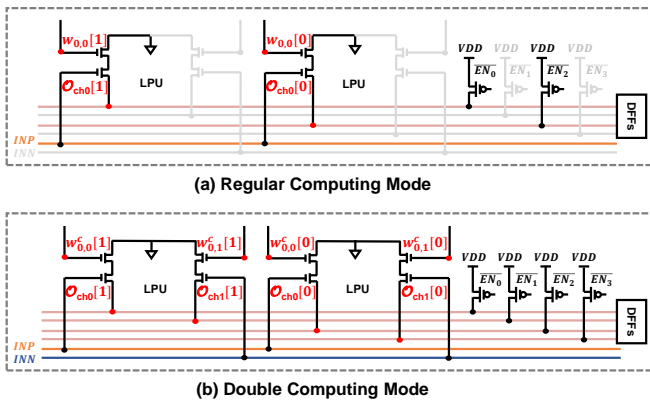Fig. 6. Circuit design of PIM core and its corresponding truth table.

Fig. 7. Illustration of computation modes for PIM core.

$(\sum \mathcal{I}) \times \mathcal{M}$ and Psum in ARU to recover the convolution results. The results are sent to post-process unit for performing pooling and other operations. Finally, $\mathcal{O}$ acquired from the post-process unit is written back to the ping-pong memory. Details of some key modules are presented below.

2) PIM core with dual-broadcast input structure (DBIS) and reconfigurable unit: As listed in Fig. 6(c), each PIM core consists of 32 compartments, a reconfigurable unit, and other peripheral circuits. Each compartment comprises 16 double-bitwise multiply units (DBMUs) for four signed 8 bit weights configurations $(w^c_{0,0}, w^c_{0,1}, w^c_{0,2}, w^c_{0,3})$, a DBIS supporting two different inputs (INN and INP), and a readout block. Each DBMU consists of sixty-four 6T SRAM cells (SC #0 – SC #63) and one local processing unit (LPU), as illustrated in Fig. 6(a) and Fig. 6(b). Sixty-four SCs in one column share one LPU and perform two bitwise AND operations of 1 bit input and 1 bit weight with the support of DBIS. The results generated by LPU are sampled and held by the readout block. The reconfigurable unit and the DBIS endow the PIM core with the flexibility of executing various convolution operations. Each PIM core can be operated in three operation modes: ① normal SRAM mode supporting either read or write operations, ② regular computing mode for multi-bit MAC operations, and ③ double computing mode with dual-broadcast inputs for multi-bit MAC operations.

In normal SRAM mode, the corresponding bits stored in Q and $\overline{Q}$ can be accessed through BL pairs (BLP/BLN) in each standard 6T SRAM bitcell for read and write operation.

In regular computing mode, only dynamic logic switches $\overline{EN_0}$ and $\overline{EN_2}$ are set to ground for pre-charging output to VDD during each computing cycle. As illustrated in Fig. 7(a), the black lines represent an available path and the gray lines represent an inaccessible path, indicating only half of the LPU enable and perform multiplication of binary inputs and weights. A corresponding DFF samples and holds the computational result, which satisfies:

$$O_{ch0}[1] \;=\; w_{0,0}[1] \;\&\; INP. \qquad (6)$$

In double computing mode, overall dynamic logic switches $\overline{EN_0} \sim \overline{EN_3}$ are set to ground for pre-charging output to VDD during each computing cycle. As illustrated in Fig. 7(b), both two paths could contribute current to the computational results. For example, the left path in LPU opens to perform multiplication of $O_{ch0}[1]$ ($O_{ch0}[1] = w^c_{0,0}[1]\&INP$) and the right path opens to perform multiplication of $O_{ch1}[1]$ ($O_{ch1}[1] = w^c_{0,1}[1]\&INN$). With the support of the FCC algorithm and DBIS, two different inputs (INN and INP) are broadcasted to LPU for two independent MAC operations. In each compartment, only one row is activated every cycle to avoid read disturbance issues. Bitwise AND operations are performed in all compartments simultaneously, and their results are accumulated vertically in the reconfigurable unit.

The reconfigurable unit consists of four adder units and a multiplexer (Mux) for flexibly combining the sum of the output elements. Each adder unit comprises two adder trees, and each adder tree is responsible for accumulating AND results of 16 compartments. The output of the adder unit is either directly extracted from two individual adder trees or the combination of the results from two adder trees. The former output scheme represents two different output channels of $\mathcal{O}$ while the latter represents a single output channel. This flexibility allows for optimizing the performance and efficiency of the adder unit in different scenarios.

For std-conv and pw-conv, each adder unit accumulates the AND results in the same row of 32 compartments by combining the results of two adder trees. For dw-conv, convolutional operations are performed on a per-channel basis without sharing $\mathcal{I}$ among filters. Despite the employment
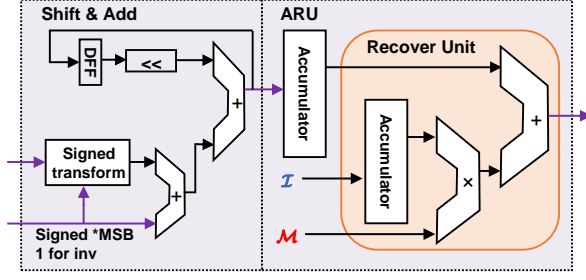
Fig. 8. Circuit design of merge unit.

Fig. 9. Demonstration of decomposing Biased-Comp filters into Comp filters.

of DBIS, the computational constraints dictate that only a pair of output channels can be concurrently computed. To mitigate load latency, we load two pairs of twin-weights in one compartment row during each write operation. Consequently, we perform a two-stage computation and alternately activate two adder units during each stage to ensure the correctness of the results. Moreover, for $3 \times 3 \times 1$ filters, the spatial utilization ratio of compartments is only $9/32$. To improve spatial utilization, each adder unit is responsible for computing two different channels and outputting the results simultaneously. The proposed methodology enables parallel computation of four output channels of $\mathcal{O}$ during `dw-conv` operation, which is analogous to the concurrent processing of `std-conv` and `pw-conv`.

3) Merge unit: Fig. 8 presents the circuit design of the merge unit that performs shift & add operations and then recovers the convolution results. The partial results generated by the PIM core are first shifted and accumulated based on their respective bit position in shift & add unit. Given that the *Biased-Comp filters* are decomposed into $\mathcal{M}$ and *Comp filters* during data mapping, as shown in Eq. 7, ARU is adopted to recover the final convolution results ($\mathcal{O}$). In ARU, the Psums provided by the shift & add unit are first accumulated in the vector-wise direction. For `Conv` layers that are subjected to the FCC algorithm, the convolution results are obtained by assembling the accumulated Psums and the multiplication results ($(\sum \mathcal{I}) \times \mathcal{M}$). For `FC` layers, we disable the recover unit and only accumulate Psum for the final convolution results.

$$
\begin{aligned}
\mathcal{O} &= \sum \left(\mathcal{I} * f^{bc} \right) \\
&= \sum \left(\mathcal{I} * (f^c + \mathcal{M}) \right) \\
&= \sum \left(\mathcal{I} * f^c \right) + \left(\sum \mathcal{I} \right) \times \mathcal{M}.
\end{aligned} \tag{7}
$$

D. Data Mapping

To bridge the gap between FCC training algorithm and DDC-PIM architecture, we propose a versatile data mapping method, including offline data decomposition and mapping strategies for various kinds of computation. Fig. 9 presents how the *Biased-Comp filters* obtained from FCC algorithm are decomposed into *Comp filters* and $\mathcal{M}$. The cross-coupled structures are granted the capability of fully utilizing their complementary states. For example, here $w_{0,0}^{bc} = -5$, $w_{0,1}^{bc} = 6$, and $\mathcal{M}_0 = 1$. After decomposition, $w_{0,0}^c$ and $w_{0,1}^c$ are -6 (i.e. 11111010_2) and 5 (i.e. 00000101_2) respectively. The
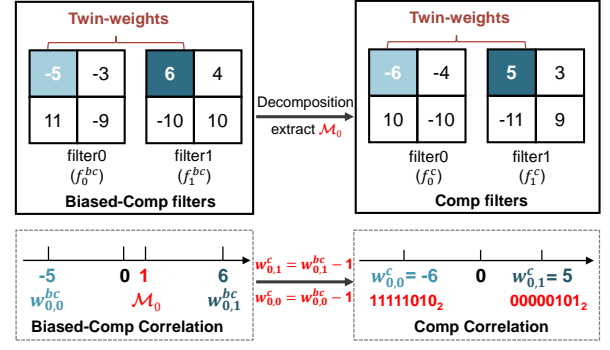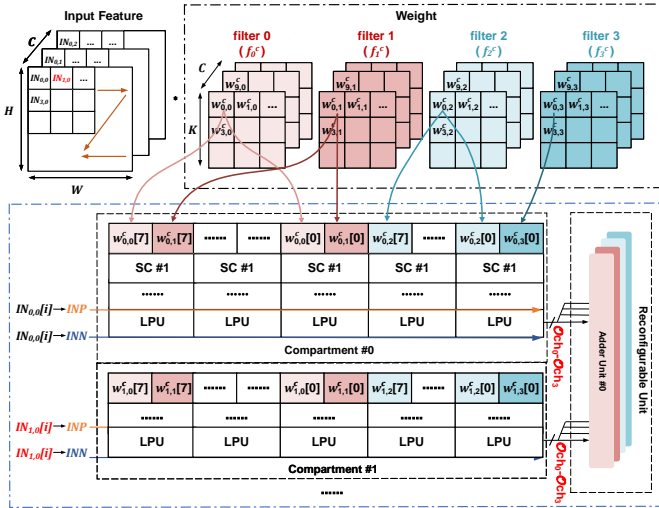
twin-weights in *Comp filters* exhibit bitwise complementary properties, only one of them needs to be transferred. Hence, we first extract half of the *Comp filters* ($f_0^c, f_2^c, f_4^c, ...$), convert them into 1-dimensional vectors using the $im2col$ function, and splice every two `8 bit` vectors into a `16 bit` vector.

Due to the limited on-chip resources, these vectors and the corresponding mean values are transferred from off-chip DRAM to weight memory layer-by-layer. Each time a part of the weights is fetched from weight memory and loaded to PIM macros for performing matrix vector multiplication (MVM). Once these cached weights in weight memory are nearly exhausted for computation within the PIM macros, our system proactively pre-fetches the weights for the subsequent layer, effectively masking the latency typically associated with off-chip DRAM access. For `std-conv` and `pw-conv`, DDC-PIM tiles these vectors in weight memory according to the total capacity of 4 PIM macros, and then splits them into sub-vectors based on the number of compartments.

For `dw-conv`, we adopt another mapping strategy with padding technique to solve the spatial under-utilization problem of the PIM core. For `FC` layer, we can regard it as a special case since we exclude it from the scope of the FCC algorithm. We transfer all weights of the `FC` layer, tile them according to the total capacity of 4 PIM macros, and compute in regular computing mode, as shown in Fig. 7.

1) Standard or pointwise convolution: For `std-conv` and `pw-conv`, two pairs of the twin-weights are mapped to the same row of the destined compartment. Meanwhile, the weights within each sub-vector are allocated to rows with the same position across 32 compartments. Therefore, the results of `AND` operation between weights and vector-wise input are accumulated vertically. We illustrate our method using a `std-conv` layer of 3D filters in Fig. 10. By exploiting the bitwise complementary property of the twin-weights in f_0^c and f_1^c, f_2^c and f_3^c, we only transmit the corresponding weights in f_0^c and f_2^c. For the first row of Compartment #0, we transform $w_{0,0}^c$ in f_0^c and $w_{0,2}^c$ in f_2^c into $\{w_{0,0}^c, w_{0,2}^c\}$, and load them to this row in normal SRAM mode. The data stored in first row from Compartment #0 can then represent two pairs of twin-weights from four filters ($w_{0,0}^c, w_{0,1}^c, w_{0,2}^c, w_{0,3}^c$), due to the cross-coupled structure of 6T SRAM and the bitwise complementary correlation ($w_{0,0}^c = \sim w_{0,1}^c$ and $w_{0,2}^c = \sim w_{0,3}^c$). In this case, the PIM core performs in double computing mode, where INN and INP receive the same vector-wise input, as

Fig. 10. Data mapping demonstration of `std-conv`.

Fig. 11. Data mapping demonstration of `dw-conv`.

different filters share the same $\mathcal{I}$ in `std-conv`. Thus, the maximum computation parallelism $(X \times Y \times B)$ supported by the DDC-PIM is $32 \times 4 \times 32$. Among them, X, Y, and B denote the number of compartments, macros, and bits for parallel computing.

2) Depthwise convolution: Like `std-conv`, we apply filter transformation and assign two pairs of twin-weights to the same row of each compartment in `dw-conv`. As we mentioned in the reconfigurable unit design in Section III-C2, only half of the compartments are activated during each computation stage. We double the spatial utilization with the support of padding and reconfigurable unit. For example, as shown in Fig. 11, we map $f_0^c \sim f_3^c$ into Compartment #0 $\sim$ Compartment #8, and $f_4^c \sim f_7^c$ into Compartment #16 $\sim$ Compartment #24. When performing computation without the optimization techniques in DDC-PIM, the maximum computation parallelism is $9 \times 1 \times 8$ since only 9 out of 32 compartments are utilized. Based on the FCC algorithm and PIM core with DBIS, INN and INP can receive distinct vector-wise inputs, doubling the computation parallelism to $9 \times 1 \times 16$. Meanwhile, the reconfigurable unit enables DDC-PIM to load four different filters $(f_0^c, f_2^c, f_4^c, f_6^c)$ simultaneously, representing eight filters $(f_0^c$ to $f_7^c)$ for two alternating computations to further double the parallelism. Thus, the overall maximum computation parallelism for `dw-conv` is $18 \times 1 \times 16$, equivalent to $4\times$ acceleration.

IV. EVALUATION RESULTS

A. Experimental Setup

Hardware implementation. DDC-PIM is evaluated on 14 nm technology, with a 128 KB ping-pong memory, a 256 KB weight memory, and four 4 KB PIM macros. The power consumption, latency, and area of PIM macros are extracted from the post-layout of customized design extension from [14]. The area and power consumption of weight memory and ping-pong memory are estimated by PCACTI [32]. The remaining digital modules are implemented with Verilog HDL and synthesized by Design Compiler, while the power consumption
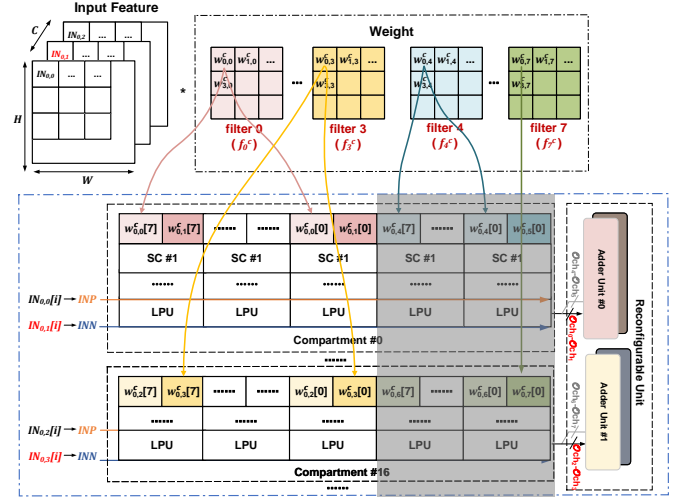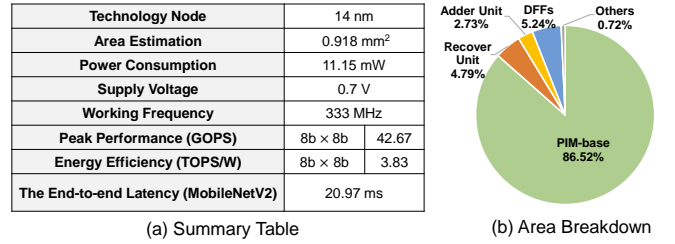

Fig. 12. Summary table of DDC-PIM and area breakdown of PIM macro.

is obtained by PTPX. Aiming to evaluate the performance of whole architecture and validate the data mapping, we implement a customized cycle-accurate C++ simulator and dataflow mapper.

PIM baseline. In order to assess the performance improvement achieved by our algorithm/architecture co-design, we implement a digital PIM as a baseline. Compared with our DDC-PIM, the baseline design does not include reconfigurable unit, dual-broadcast input structure and recover unit. Furthermore, the PIM core in the baseline design is constrained to only operate in regular computing mode. The rest of hardware settings are identical to those of our DDC-PIM architecture.

Benchmarks and models. We evaluate two popular compact NN models, MobileNetV2 and EfficientNet-B0, on the CIFAR10 dataset benchmark. In addition, to demonstrate the generality and applicability of the FCC algorithm across different neural network architectures, we also perform experiments on three representative regular NN models, namely AlexNet, VGG19 and ResNet18. All of these models are trained for 1000 epochs. We apply `INT8` quantization on inputs and weights for all layers.

B. DDC-PIM Implementation Summary

As shown in Fig. 12(a), the total area and power of DDC-PIM are about 0.918 mm^2 and 11.15 mW. The clock frequency is 333 MHz and the peak performance is about 42.67 GOPS at $8b \times 8b$. The end-to-end latency of MobileNetV2 is 20.97 ms, with the MVM operations exhibiting a latency

TABLE II
COMPARISON WITH PRIOR WORKS FOR PIM MACROS.

PIM Macro Type	Analog PIM					Digital PIM		
	Nat. Elec.'22 [33]	JETCAS'22 [34]	Nat. Elec.'21 [35]	VLSI'21 [11] (10T)	ISSCC'20 [24] (6T+LCC)	ISSCC'21 [26] (6T+LCC)	ISSCC'22 [14] (6T+LCC)	**This Work**
PIM Device	PCM	PCM	RRAM	SRAM	SRAM	SRAM	SRAM	SRAM
Technology Node	14nm	22nm	22nm	28nm	28nm	22nm	28nm	14nm
Array Size	**64Kb**	**64Kb**	**4Mb**	**3456Kb**	**64Kb**	**64Kb**	**32Kb**	**32Kb**
Weight Capacity	**64Kb**	**64Kb**	**4Mb**	**3456Kb**	**64Kb**	**64Kb**	**32Kb**	**64Kb**
Cell Type	8T4R	/	1T1R	10T1C	6T	6T	6T	6T
Macro Area (mm^2)	1.392	0.83	6	20.9	0.362	0.202	0.040	0.0115
Integration Density* (Kb/mm^2)	45.98@14nm	77.11@22nm	682.67@22nm	165.4@28nm	177@28nm	317@22nm	800@28nm	2783@14nm
Integration Density (Kb/mm^2) (Normalized to 28nm)	11.52	47.68	422.09	165.4	177	196	**800**	**697**
Weight Density† (Kb/mm^2)	**45.98@14nm**	**77.11@22nm**	**682.67@22nm**	**165.4@28nm**	**177@28nm**	**317@22nm**	**800@28nm**	**5565@14nm**
Weight Density (Kb/mm^2) (Normalized to 28nm)	**11.52**	**47.68**	**422.09**	**165.4**	**177**	**196**	**800**	**1391**
Computing Units	ADC	ADC	CW-CVS	Flash ADC	LMAR-SAR-ADC	Logic Circuit	Logic Circuit	Logic Circuit
Area Efficiency (GOPS/mm^2) (Normalized to 28nm)	**177.38/63 (8b/8b)**	**712.15 (8b/4b)**	**3.47 (8b/8b)**	**234 (1b/1b)**	**84.2 (8b/8b)**	**2802.5 (8b/8b)**	**133.3 (8b/8b)**	**231.9 (8b/8b)**
Energy Efficiency (TOPS/W)	**9.76/2.48 (8b/8b)**	**6.39 (8b/4b)**	**15.60 (8b/8b)**	**588 (1b/1b)**	**14.1 (8b/8b)**	**24.7 (8b/8b)**	**27.38 (8b/8b)**	**72.41 (8b/8b)**

* Integration Density = Array Size / Macro Area
† Weight Density = Weight Capacity / Macro Area

of 18.02 ms. The PIM macro area breakdown analysis is shown in Fig. 12(b). PIM macro can be divided into base digital PIM logic (PIM-base), additional logic introduced by our techniques (DFFs, adder units and the recover unit), and others. PIM-base in PIM macro is equivalent to the counterpart in [14], and the additional logic is designed for supporting our co-design. In this work, the complementary states ($Q/\overline{Q}$) in 6T SRAM represent two individual bits for parallel computation. Hence, compared with PIM-base in [14], DDC-PIM needs extra storage units (DFFs) and computation units (adder units) to process the additional information. These units only incur a minor area overhead of about 5.24% and 2.73%, respectively. Meanwhile, DDC-PIM decomposes *Biased-Comp filters* into *Comp filters* and $\mathcal{M}$, which requires extra computation units to recover the final results, costing only about 4.79% overhead in area consumption. The energy efficiency in our PIM macro is 72.41 TOPS/W at 8b $\times$ 8b.

C. Comparison with PIM Macros in Prior Works

Tab. II presents a comprehensive comparison of PIM macros among DDC-PIM and other state-of-the-art studies, which can be categorized into the analog domain [11, 24, 33–35] and digital domain [14, 26]. We mainly focus on comparing weight capacity, integration density, weight density, and area efficiency of DDC-PIM macro against the others. Generally, the weight capacity of a PIM macro is equal to its array size, thus its integration density and weight density are also the same. For example, the array size and weight capacity of [11] are both 3456 Kb. The integration density and weight density are about 165.4 Kb/mm^2 at 28 nm. Meanwhile, with a fixed process and array size, the weight density of analog PIM is usually much lower than digital PIM, due to the extra area overhead introduced by ADC/DAC. For example, the integration density of [14] and [26] are much larger than that of [11] and [24]. In this work, by exploiting the complementary state pairs as independent bits of information, the weight capacity of DDC-PIM is twice its array size. To facilitate a fair comparison, we scale both the integration density and weight
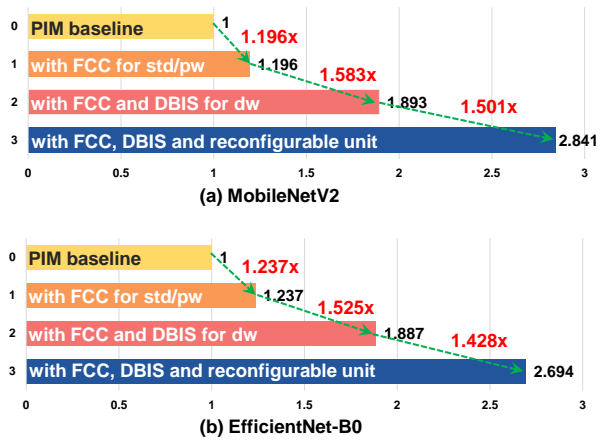

Fig. 13. Speedup analysis for MobileNetV2 and EfficientNet-B0. Here we illustrate the speedup of the FCC algorithm for `std-conv` and `pw-conv`, FCC algorithm for `dw-conv` with DBIS and reconfigurable unit over PIM baseline.

density to 28 nm technology node. As PIM-base in PIM macro is equivalent to the counterpart in [14], the additional logic for supporting our co-design brings a slight decrease in integration density. Meanwhile, due to that a pair of computing units in DDC-PIM can support two independent AND operations, the area efficiency is also improved by about $1.74\times$ compared with [14].

D. Speedup for MobileNetV2 and EfficientNet-B0

Fig. 13 clearly illustrates the acceleration in MobileNetV2 and EfficientNet-B0 gained by co-designing the FCC algorithm, data mapping, and architecture. By applying the FCC algorithm to both `std-conv` and `pw-conv`, we achieve a speedup of about $1.196\times$ for MobileNetV2 and $1.237\times$ for EfficientNet-B0, respectively. Moreover, the combination of the FCC algorithm and DBIS improves the efficiency of `dw-conv`, achieving a speedup of $1.583\times$ for MobileNetV2 and $1.525\times$ for EfficientNet-B0. Furthermore, the proposed DDC-PIM architecture, which incorporates the FCC algo-

TABLE III
ACCURACY EVALUATION OF FCC ALGORITHM APPLIED ON DIFFERENT LAYERS AND DIFFERENT MODELS ON CIFAR10 DATASET.

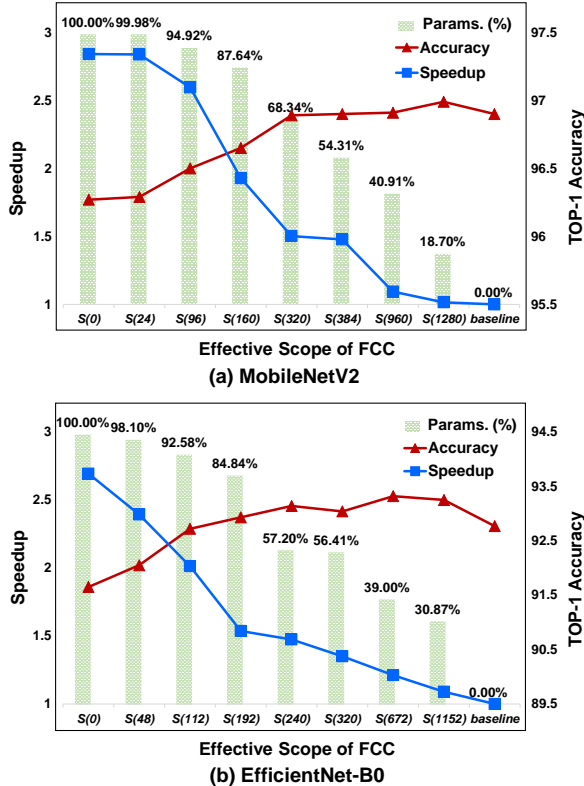| Model		FCC Not Applied	FCC Applied on `Conv` Layers		FCC Applied on both `Conv` and `FC`		Param. Ratio of `FC` Layers
		Top-1 Accu. Baseline (%)	Top-1 Accu. for `Conv` Layers (%)	Accu. Drop for `Conv` Layers (%)	Top-1 Accu. for `Conv` & `FC` (%)	Accu. Drop for `Conv` & `FC` (%)	
Compact NNs	MobileNetV2	96.71	95.99	0.72	95.69	1.02	0.57%
	Efficient-B0	92.77	91.65	1.12	90.87	1.90	0.11%
Regular NNs	AlexNet	93.08	92.52	0.56	91.20	1.88	79.12%
	VGG19	96.29	95.64	0.65	95.11	1.18	55.71%
	ResNet18	97.15	96.73	0.42	95.97	1.18	0.04%

Fig. 14. Speedup and accuracy tradeoff for FCC algorithm on MobileNetV2 and EfficientNet-B0. The height of bars in the bar diagram denotes the proportion of parameters within $S(i)$ to the overall parameters.

rithm, DBIS, and a reconfigurable unit, attains a speedup of about $1.501\times$ for MobileNetV2 and $1.428\times$ for EfficientNet-B0. For `dw-conv`, the speedup brought by the FCC algorithm solely relies on the support of DBIS, thus these two approaches must be bonded during execution. Although `dw-conv` has fewer parameters and fewer computation requirements than `pw-conv` and `std-conv`, the overall latency of compact NNs is still dominated by `dw-conv` due to its low computation parallelism. With the optimization techniques tailored for `dw-conv`, DDC-PIM can achieve about $2.841\times$ and $2.694\times$ speedup for MobileNetV2 and EfficientNet-B0 respectively.

E. Evaluation of FCC Algorithm

Through a comprehensive set of experiments, we demonstrate that the FCC algorithm has a heterogeneous impact on the performance of NN model's layers, indicating a varying degree of susceptibility across different levels of abstraction. This impact translates into distinct acceleration effects. For simplicity, we introduce the effective scope $S(i)$ as a set composed of all the layers in a given model with more than i filters, and apply the FCC algorithm to the layers in $S(i)$. Therefore, we assess the tradeoff between speedup and accuracy by altering the $S(i)$ of the FCC algorithm and present the results in Fig. 14. In this case, all layers represent all `Conv` layers. The results show that when FCC is applied to all layers, DDC-PIM can obtain $2.841\times$ and $2.694\times$ speedup with only 0.72% and 1.12% accuracy drop for MobileNetV2 and EfficientNet-B0 respectively. To gain more insights into the benefits of our approach, we scrutinize the number of parameters, accuracy, and speedup at $S(112)$ in Fig. 14(b). DDC-PIM can apply FCC to of most 92.58% parameters, achieving a $2.01\times$ speedup without any accuracy degradation. Furthermore, we observe that applying the FCC algorithm to a smaller $S(i)$ may result in a deterioration of the inference accuracy. However, applying the FCC algorithm to a larger $S(i)$ may exceed the baseline accuracy. For applications with stringent requirements on model accuracy, we can investigate a layered application of the FCC algorithm.

To demonstrate the generality of the FCC algorithm, we also perform experiments on regular NNs such as AlexNet, VGG19, and ResNet18. Tab. III illustrates the versatility and robustness of FCC algorithm applied on different layers and NN models. First, we compare the accuracy of applying the FCC algorithm only to `Conv` layers as well as applying it to both the `Conv` layers and `FC` layers. The findings indicate that `FC` layers are more susceptible to the FCC algorithm than the `Conv` layers, as the former setting yields higher accuracy than the latter. Moreover, the results imply that the FCC algorithm can handle different levels of complexity in the NNs. Generally, regular NN models exhibit a higher degree of compatibility with the FCC algorithm with lower accuracy degradation due to its higher redundancy. However, due to `FC` layers accounting for a significant proportion of the total number of parameters in regular NNs, applying the FCC algorithm both to `Conv` layers and `FC` layers obtains larger accuracy deterioration than in compact NNs. Taking AlexNet as an example, the accuracy drop for `Conv` layers is 0.56%, but for `Conv` layers and `FC` layers is up to 1.88%. Although applying the FCC algorithm to all layers, including the `Conv` layers and `FC` layers, can achieve better acceleration. However, this often comes at the expense of significant accuracy degradation. Hence, the choice of FCC algorithms should be based on the

TABLE IV
COMPARISON OF ACCURACY AND COMPRESSION RATIO OF
MOBILENETV2 WITH DIFFERENT METHODS ON CIFAR-100.

Model	Method	Top-1 Accu. (%)	Accu. Drop (%)	Compression Ratio
MobileNetV2	Original	80.48	0	0%
	$2\colon 4$ Pruning	79.94	0.54	50%
	FCC algorithm with $2\colon 4$ Pruning	78.81	1.13	$\sim 75\%$

TABLE V
ACCURACY COMPARISON OF MOBILEVIT-XS ON CIFAR-10.

Model	Method	Top-1 Accu.(%)
MobileViT-XS	Original	90.88
	FCC algorithm for `Conv` layers	89.04

trade-off between acceleration and accuracy requirements.

V. DISCUSSIONS

Existing pruning methods. Apart from designing compact NNs, network sparsification is another promising compression algorithm to reduce data and computation requirements. Numerous effective methods have been proposed to introduce sparsity [36–38]. Weight pruning, indeed, holds a prominent position among these methods. Generally, weight pruning can be divided into fine-grained pruning and coarse-grained pruning. Fine-grained pruning allows for precise removal of unnecessary weights, leading to higher compression rates compared to coarse-grained pruning. However, pruning individual weights may introduce irregularities in the network, leading to increased computational overhead during inference due to non-contiguous memory access or additional indexing operations. Coarse-grained pruning achieves significant computational savings by removing entire neurons, channels, or layers from a neural network. However, it will introduce non-negligible accuracy loss. To further reduce memory access and computation cost, some works apply the joint-way compression with multiple approaches. A usual case is using compact models such as MobileNet as the base model to do additional compression such as quantization or sparsification. However, compact models are already size-efficient than normal models, which usually impedes further model compression. For example, the weight pruning on MobileNet can achieve only $50\% \sim 60\%$ sparsity without paying significant accuracy loss [36].

Apply FCC algorithm on pruned models. FCC algorithm is a novel compression method that is designed to reduce the memory access and mitigate the effects of computational irregularity. The compatibility between the FCC algorithm and traditional pruning techniques is an intriguing question that merits further investigation. In order to thoroughly examine and confirm this compatibility, we have conducted additional supplementary experiments. We choose the well-known fine-grained $2\colon 4$ structured pruning algorithm proposed by NVIDIA to verify the orthogonal nature of our algorithm relative to traditional pruning techniques [39]. Specifically, we have employed the $2\colon 4$ pruning algorithm proposed by NVIDIA with an accuracy of 79.94% and achieved a 50% compression ratio, as shown in Tab. IV. Then, we have further conducted experiments incorporating the FCC algorithm with the $2\colon 4$ pruning algorithm. The experimental accuracy is 78.81%. This result demonstrates the compatibility between the FCC algorithm and the $2\colon 4$ structured pruning technique.

This compression method achieves an additional compression of nearly 50% on top of the initial 50% achieved by the original $2\colon 4$ pruning algorithm, without introducing significant accuracy loss.

Applicability for other DNNs. To verify the applicability of the FCC algorithm, we choose MobileViT-XS, a variant of lightweight transformer models, for supplementary experiments. The experimental results, as shown in Tab. V, illustrate that applying the FCC algorithm to the convolutional layer of MobileViT-XS will not introduce a large accuracy loss. Exploring the potential application of our algorithm to alternative neural networks constitutes a prospective avenue of our research. We plan to conduct a more comprehensive analysis in the future to provide additional perspectives on the advantages and constraints of our approach within a more expansive framework.

VI. CONCLUSIONS

This paper presents DDC-PIM, a novel co-design of algorithm and architecture that enhances the data capacity of SRAM without altering its structure. At the algorithm level, the proposed FCC algorithm leverages the complementary property of filters to transform adjacent filter pairs into the bitwise complementary format with negligible accuracy loss. The DDC-PIM architecture exploits the intrinsic cross-coupled structures of 6T SRAM cells to double the equivalent data capacity. Moreover, DDC-PIM adopts a flexible data mapping method that adapts to various convolution operations. Experimental results demonstrate that DDC-PIM yields about $2.84\times$ and $2.69\times$ speedup on MobileNetV2 and EfficientNet-B0, compared with baseline implementations, and can increase weight density and area efficiency up to $8.41\times$ and $2.75\times$, compared with state-of-the-art SRAM-based PIM.

REFERENCES

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.

[2] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," in *Proceedings of the Association for the Advancement of Artificial Intelligence (AAAI)*, 2017.

[3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[4] M. Burchi and R. Timofte, "Audio-Visual Efficient Conformer for Robust Speech Recognition," in *Proceedings of the Winter Conference on Applications of Computer Vision (WACV)*, 2023.

[5] Y. Zhang, W. Chan, and N. Jaitly, "Very deep convolutional networks for end-to-end speech recognition," in *Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2017.

[6] W. Wang, J. Dai, Z. Chen, Z. Huang, Z. Li, X. Zhu, X. Hu, T. Lu, L. Lu, H. Li *et al.*, "Internimage: Exploring large-scale

vision foundation models with deformable convolutions," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023, pp. 14 408–14 419.

[7] Y. Fang, W. Wang, B. Xie, Q. Sun, L. Wu, X. Wang, T. Huang, X. Wang, and Y. Cao, "Eva: Exploring the limits of masked visual representation learning at scale," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023, pp. 19 358–19 369.

[8] W. Su, X. Zhu, C. Tao, L. Lu, B. Li, G. Huang, Y. Qiao, X. Wang, J. Zhou, and J. Dai, "Towards all-in-one pre-training via maximizing multi-modal mutual information," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023, pp. 15 888–15 899.

[9] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," *arXiv preprint arXiv:1704.04861*, 2017.

[10] M. Tan and Q. Le, "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks," in *Proceedings of the International Conference on Machine Learning (ICML)*, 2019.

[11] S. Yin, B. Zhang, M. Kim, J. Saikia, S. Kwon, S. Myung, H. Kim, S. J. Kim, M. Seok, and J.-s. Seo, "PIMCA: A 3.4-Mb Programmable In-Memory Computing Accelerator in 28nm for On-Chip DNN Inference," in *Proceedings of the Symposium on VLSI Circuits (VLSIC)*, 2021.

[12] S. K. Gonugondla, M. Kang, and N. Shanbhag, "A 42pJ/decision 3.12TOPS/W Robust In-Memory Machine Learning Classifier with On-Chip Training," in *Proceedings of the International Solid-State Circuits Conference (ISSCC)*, 2018.

[13] J.-W. Su, X. Si, Y.-C. Chou, T.-W. Chang, W.-H. Huang, Y.-N. Tu, R. Liu, P.-J. Lu, T.-W. Liu, J.-H. Wang *et al.*, "A 28nm 64Kb Inference-Training Two-Way Transpose Multibit 6T SRAM Compute-In-Memory Macro for AI Edge Chips," in *Proceedings of the International Solid-State Circuits Conference (ISSCC)*, 2020.

[14] B. Yan, J.-L. Hsu, P.-C. Yu, C.-C. Lee, Y. Zhang, W. Yue, G. Mei, Y. Yang, Y. Yang, H. Li *et al.*, "A 1.041-Mb/mm^2 27.38-TOPS/W Signed-INT8 Dynamic-Logic-Based ADC-less SRAM Compute-In-Memory Macro in 28nm with Reconfigurable Bitwise Operation for AI and Embedded Applications," in *Proceedings of the International Solid-State Circuits Conference (ISSCC)*, 2022.

[15] M. Imani, S. Gupta, Y. Kim, M. Zhou, and T. Rosing, "DigitalPIM: Digital-based Processing In-Memory for Big Data Acceleration," in *Proceedings of Great Lakes Symposium on VLSI (GLSVLSI)*, 2019.

[16] M. Imani, S. Gupta, Y. Kim, and T. Rosing, "FloatPIM: In-Memory Acceleration of Deep Neural Network Training with High Precision," in *Proceedings of International Symposium on Computer Architecture (ISCA)*, 2019.

[17] X. Chen, X. Wang, X. Jia, J. Yang, G. Qu, and W. Zhao, "Accelerating Graph-Connected Component Computation with Emerging Processing-In-Memory Architecture," *Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 41, no. 12, pp. 5333–5342, 2022.

[18] Y. Zhang, J. Wang, C. Lian, Y. Bai, G. Wang, Z. Zhang, Z. Zheng, L. Chen, K. Zhang, G. Sirakoulis *et al.*, "Time-Domain Computing in Memory Using Spintronics for Energy-Efficient Convolutional Neural Network," *Transactions on Circuits and Systems I: Regular Papers (TCAS-I)*, vol. 68, no. 3, pp. 1193–1205, 2021.

[19] X. Wang, J. Yang, Y. Zhao, Y. Qi, M. Liu, X. Cheng, X. Jia, X. Chen, G. Qu, and W. Zhao, "TCIM: Triangle Counting Acceleration With Processing-In-MRAM Architecture," in *Proceedings of the Design Automation Conference (DAC)*, 2020.

[20] T. Kim, Y. Jang, M.-G. Kang, B.-G. Park, K.-J. Lee, and J. Park, "SOT-MRAM Digital PIM Architecture With Extended Parallelism in Matrix Multiplication," *Transactions on Computers (TC)*, vol. 71, no. 11, pp. 2816–2828, 2022.

[21] M. Kang, S. K. Gonugondla, A. Patil, and N. R. Shanbhag, "A Multi-Functional In-Memory Inference Processor Using a Standard 6T SRAM Array," *Journal of Solid-State Circuits (JSSC)*, vol. 53, no. 2, pp. 642–655, 2018.

[22] M. E. Sinangil, B. Erbagci, R. Naous, K. Akarvardar, D. Sun, W.-S. Khwa, H.-J. Liao, Y. Wang, and J. Chang, "A 7-nm Compute-In-Memory SRAM Macro Supporting Multi-Bit Input, Weight and Output and Achieving 351 TOPS/W and 372.4 GOPS," *Journal of Solid-State Circuits (JSSC)*, vol. 56, no. 1, pp. 188–198, 2020.

[23] Y. Zhang, L. Xu, Q. Dong, J. Wang, D. Blaauw, and D. Sylvester, "Recryptor: A Reconfigurable Cryptographic Cortex-M0 Processor With In-Memory and Near-Memory Computing for IoT Security," *Journal of Solid-State Circuits (JSSC)*, vol. 53, no. 4, pp. 995–1005, 2018.

[24] X. Si, Y.-N. Tu, W.-H. Huang, J.-W. Su, P.-J. Lu, J.-H. Wang, T.-W. Liu, S.-Y. Wu, R. Liu, Y.-C. Chou *et al.*, "A 28nm 64Kb 6T SRAM Computing-in-Memory Macro with 8b MAC Operation for AI Edge Chips," in *Proceedings of the International Solid-State Circuits Conference (ISSCC)*, 2020.

[25] Z. Yue, Y. Wang, Y. Qin, L. Liu, S. Wei, and S. Yin, "BR-CIM: An Efficient Binary Representation Computation-In-Memory Design," *Transactions on Circuits and Systems I: Regular Papers (TCAS-I)*, vol. 69, no. 10, pp. 3940–3953, 2022.

[26] Y.-D. Chih, P.-H. Lee, H. Fujiwara, Y.-C. Shih, C.-F. Lee, R. Naous, Y.-L. Chen, C.-P. Lo, C.-H. Lu, H. Mori *et al.*, "An 89TOPS/W and 16.3TOPS/mm^2 All-Digital SRAM-Based Full-Precision Compute-In Memory Macro in 22nm for Machine-Learning Edge Applications," in *Proceedings of the International Solid-State Circuits Conference (ISSCC)*, 2021.

[27] F. Tu, Z. Wu, Y. Wang, L. Liang, L. Liu, Y. Ding, L. Liu, S. Wei, Y. Xie, and S. Yin, "TranCIM: Full-Digital Bitline-Transpose CIM-based Sparse Transformer Accelerator With Pipeline/Parallel Reconfigurable Modes," *Journal of Solid-State Circuits (JSSC)*, pp. 1–12, 2022.

[28] J. Yue, C. He, Z. Wang, Z. Cong, Y. He, M. Zhou, W. Sun, X. Li, C. Dou, F. Zhang *et al.*, "A 28nm 16.9-300TOPS/W Computing-in-Memory Processor Supporting Floating-Point NN Inference/Training with Intensive-CIM Sparse-Digital Architecture," in *IEEE International Solid-State Circuits Conference (ISSCC)*, 2023, pp. 1–3.

[29] S. Liu, P. Li, J. Zhang, Y. Wang, H. Zhu, W. Jiang, S. Tang, C. Chen, Q. Liu, and M. Liu, "A 28nm 53.8TOPS/W 8b Sparse Transformer Accelerator with In-Memory Butterfly Zero Skipper for Unstructured-Pruned NN and CIM-Based Local-Attention-Reusable Engine," in *IEEE International Solid-State Circuits Conference (ISSCC)*, 2023, pp. 250–252.

[30] F. Tu, Z. Wu, Y. Wang, W. Wu, L. Liu, Y. Hu, S. Wei, and S. Yin, "MulTCIM: A 28nm 2.24μJ/Token Attention-Token-Bit Hybrid Sparse Digital CIM-Based Accelerator for Multimodal Transformers," in *IEEE International Solid-State Circuits Conference (ISSCC)*, 2023, pp. 248–250.

[31] F. Tu, Y. Wang, Z. Wu, W. Wu, L. Liu, Y. Hu, S. Wei, and S. Yin, "TensorCIM: A 28nm 3.7nJ/Gather and 8.3TFLOPS/W FP32 Digital-CIM Tensor Processor for MCM-CIM-Based Beyond NN Acceleration," in *IEEE International Solid-State Circuits Conference (ISSCC)*, 2023, pp. 254–256.

[32] A. Shafaei, Y. Wang, X. Lin, and M. Pedram, "FinCACTI: Architectural Analysis and Modeling of Caches with Deeply-scaled FinFET Devices," in *Proceedings of the Computer Society Annual Symposium on VLSI (ISVLSI)*, 2014.

[33] M. Le Gallo, R. Khaddam-Aljameh, M. Stanisavljevic, A. Vasilopoulos, B. Kersting, M. Dazzi, G. Karunaratne, M. Brändli, A. Singh, S. M. Mueller *et al.*, "A 64-core mixed-signal in-memory compute chip based on phase-change memory for deep neural network inference," *Nature Electronics*, pp. 1–14, 2023.

[34] A. Garofalo, G. Ottavi, F. Conti, G. Karunaratne, I. Boybat, L. Benini, and D. Rossi, "A heterogeneous in-memory com-

puting cluster for flexible end-to-end inference of real-world deep neural networks," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems (JETCAS)*, vol. 12, no. 2, pp. 422–435, 2022.

[35] J.-M. Hung, C.-X. Xue, H.-Y. Kao, Y.-H. Huang, F.-C. Chang, S.-P. Huang, T.-W. Liu, C.-J. Jhang, C.-I. Su, W.-S. Khwa *et al.*, "A four-megabit compute-in-memory macro with eight-bit precision based on cmos and resistive random-access memory for ai edge devices," *Nature Electronics*, vol. 4, no. 12, pp. 921–930, 2021.

[36] L. Deng, G. Li, S. Han, L. Shi, and Y. Xie, "Model compression and hardware acceleration for neural networks: A comprehensive survey," *Proceedings of the IEEE*, vol. 108, no. 4, pp. 485–532, 2020.

[37] J. Yang, W. Fu, X. Cheng, X. Ye, P. Dai, and W. Zhao, "S2Engine: A novel systolic architecture for sparse convolutional neural networks," *IEEE Transactions on Computers (TC)*, vol. 71, no. 6, pp. 1440–1452, 2021.

[38] P. Dai, J. Yang, X. Ye, X. Cheng, J. Luo, L. Song, Y. Chen, and W. Zhao, "SparseTrain: Exploiting dataflow sparsity for efficient convolutional neural networks training," in *Proceedings of 57th ACM/IEEE Design Automation Conference (DAC)*, 2020, pp. 1–6.

[39] A. Mishra, J. A. Latorre, J. Pool, D. Stosic, D. Stosic, G. Venkatesh, C. Yu, and P. Micikevicius, "Accelerating sparse deep neural networks," *arXiv preprint arXiv:2104.08378*, 2021.

Yingjie Qi received the B.S. degree in computer science and technology from Beihang University, Beijing, China, in 2020. He is currently pursuing the Ph.D. degree at the School of Computer Science and Engineering, Beihang University, China. His research interests include graph neural networks acceleration, processing-in-memory architectures and deep learning compilers.

Yikun Wang received the B.S. degree in computer science and technology from Beihang University, Beijing, China, in 2022. He is currently working toward the M.S. degree at the School of Computer Science and Engineering, Beihang University, China. His current research interests include computing-in-memory architectures and deep learning accelerators.

Yiou Wang received the B.S. degree in computer science and technology from Beijing University of Technology, Beijing, China, in 2022. He is currently working toward the M.S. degree at the School of Computer Science and Engineering, Beihang University, China. His current research interests include deep learning compilers.

Ziyan He received the B.S. degree in telecommunication engineering from Xidian University, Xi'an, China, in 2022. He is currently working toward the M.S. degree at the School of Telecommunication Engineering, Xidian University, Xi'an, China. His current research interests include processing-in-memory architecture and domain-specified accelerators.

Cenlin Duan received the B.S. degree in Electronic Science and Technology from University of Electronic Science and Technology of China, Chengdu, China, in 2015, and the M.S. degree in Software Engineering from Xidian University, Xi'an, China, in 2018. She is currently pursuing the Ph.D. degree at the School of Integrated Circuit Science and Engineering, Beihang University, Beijing, China. Her current research interests include processing-in-memory architectures and deep learning accelerators.

Jianlei Yang (S'11-M'14-SM'20) received the B.S. degree in microelectronics from Xidian University, Xi'an, China, in 2009, and the Ph.D. degree in computer science and technology from Tsinghua University, Beijing, China, in 2014.
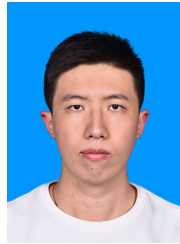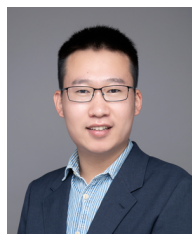
He is currently an Associate Professor in Beihang University, Beijing, China, with the School of Computer Science and Engineering. From 2014 to 2016, he was a post-doctoral researcher with the Department of ECE, University of Pittsburgh, Pennsylvania, USA. His current research interests include deep learning accelerators and neuromorphic computing systems.

Dr. Yang was the recipient of the First/Second place on ACM TAU Power Grid Simulation Contest in 2011/2012. He was a recipient of IEEE ICCD Best Paper Award in 2013, ACM GLSVLSI Best Paper Nomination in 2015, IEEE ICESS Best Paper Award in 2017, ACM SIGKDD Best Student Paper Award in 2020.

Bonan Yan is currently an assistant professor at Institute for Artificial Intelligence, Peking University. He received his PhD degree from Department of Electrical and Computer Engineering, Duke University in 2020. Hi research interests include circuits and systems for artificial intelligence chips, VLSI design for emerging memory, especially processing-in-memory technology.

He has published more than 40 papers in renowned academic journals and conferences, including ISSCC, Symposium on VLSI Technology, IEDM, DAC, etc. He actively serves as a TPC member for the conferences, including DAC, AICAS, and EDTM.

Xiaolin He received the B.S. degree in software engineering from Beihang University, Beijing, China, in 2016. He is currently pursuing the Ph.D. degree at the School of Computer Science and Engineering, Beihang University, China. His research interests include in-memory computing architectures and compiler optimization techniques.

Xiaotao Jia (S'13-M'17) received the B.S. degree in mathematics from Beijing Jiao Tong University, Beijing, China, in 2011, and the Ph.D. degree in computer science and technology from Tsinghua University, Beijing, China, in 2016. He is currently an associate professor in Beihang University, Beijing, China. His current research interests include spintronic circuits and Bayesian learning systems.

14

Xueyan Wang received the B.S. degree in computer science and technology from Shandong University, Jinan, China,in 2013, and the Ph.D. degree in computer science and technology from Tsinghua University, Beijing, China, in 2018. From 2015 to 2016, she was a visiting scholar in University of Maryland, College Park, MD, USA.

She is currently an Assistant Professor with the School of Integrated Circuit Science and Engineering in Beihang University, Beijing, China. Her current research interests include processing-in-memory architectures and hardware security.

Weitao Pan received the B.S. degree from School of Technical Physics of Xidian University in 2004. His Ph.D. degree was received from School of Microelectronics of Xidian University in 2010. Now he is an associate professor in State Key Laboratory of Integrated Service Networks of Xidian University. His current research interests include VLSI design methods and post-silicon verification.

Weisheng Zhao (Fellow, IEEE) received the Ph.D. degree in physics from the University of Paris Sud, Paris, France, in 2007.

He is currently a Professor with the School of Integrated Circuit Science and Engineering, Beihang University, Beijing, China. In 2009, he joined the French National Research Center, Paris, as a Tenured Research Scientist. Since 2014, he has been a Distinguished Professor with Beihang University. He has published more than 200 scientific articles in leading journals and conferences, such as *Nature Electronics*, *Nature Communications*, *Advanced Materials*, IEEE Transactions, ISCA, and DAC. His current research interests include the hybrid integration of nanodevices with CMOS circuit and new nonvolatile memory (40-nm technology node and below) like MRAM circuit and architecture design.

Prof. Zhao is currently the Editor-in-Chief for the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEM I: REGULAR PAPER.