SME: ReRAM-based Sparse-Multiplication-Engine to Squeeze-Out Bit Sparsity of Neural Network

Fangxin Liu[1,2], Wenbo Zhao[1,2], Zhezhi He[1], Zongwu Wang[1], Yilong Zhao[1], Tao Yang[1], Jingnai Feng[1]
Xiaoyao Liang[1], and Li Jiang[1,2,3]
1. Shanghai Jiao Tong University, Shanghai, China, 2. Shanghai Qi Zhi Institute, Shanghai, China
3. MoE Key Lab of Artificial Intelligence, AI Institute, Shanghai Jiao Tong University, China
{liufangxin, zhaowenbo, zhezhi.he, wangzongwu, ljiang_cs}@sjtu.edu.cn

***Abstract*—Resistive Random-Access-Memory (ReRAM) crossbar is a promising technique for deep neural network (DNN) accelerators, thanks to its in-memory and in-situ analog computing abilities for Vector-Matrix Multiplication-and-Accumulations (VMMs). However, it is challenging for crossbar architecture to exploit the sparsity in DNNs. It inevitably causes complex and costly control to exploit fine-grained sparsity due to the limitation of tightly-coupled crossbar structure.**

As the countermeasure, we develop a novel ReRAM-based DNN accelerator, named Sparse-Multiplication-Engine (SME), based on a hardware and software co-design framework. First, we orchestrate the bit-sparse pattern to increase the density of bit-sparsity based on existing quantization methods. Second, we propose a novel weight mapping mechanism to slice the bits of a weight across the crossbars and splice the activation results in peripheral circuits. This mechanism can decouple the tightly-coupled crossbar structure and cumulate the sparsity in the crossbar. Finally, a superior squeeze-out scheme empties the crossbars mapped with highly-sparse non-zeros from the previous two steps. We design the SME architecture and discuss its use for other quantization methods and different ReRAM cell technologies. Compared with prior state-of-the-art designs, the SME shrinks the use of crossbars up to $8.7\times$ and $2.1\times$ using ResNet-50 and MobileNet-v2, respectively, with $\leq 0.3\%$ accuracy drop on ImageNet.

***Index Terms*—ReRAM, sparsity, neural network, accelerator**

I. Introduction

Resistive Random-Access-Memory (ReRAM) crossbar emerges as a promising solution to accelerate the inference of Deep Neural-networks (DNNs) [1]–[7]. One of the reasons is that ReRAM accelerators adopt an in-situ scheme that fastens the weights of DNN on ReRAM crossbars, greatly reducing the massive cost of data movement in tradition Von-Neumann structures. In addition, ReRAM crossbar can perform highly-parallel Vector-Matrix Multiplication (VMM) by sharing inputs in rows and gathering currents in column using Kirchhoff's laws [1], [8]–[10]. However, with the development of surprisingly large DNN models such as GPT-3 [11] with 175B parameters, the increasing requirement of ReRAM crossbars to accommodate enormous DNN weights become the main hurdle of this technology.

One of the sources that contributes to the vast demand of ReRAM crossbar is the tightly coupled layout of the weight operands on ReRAM cells required by the highly-parallel VMM calculation, which makes it difficult to exploit sparsity in DNNs. Extensive works propose weight sparsification methods to reduce the number of ReRAM crossbars. Hardware-independent pruning algorithms, such as filter-/channel-wise sparsity, can derive a "smaller" dense weight matrices and directly map them to crossbars [12], [13]. These algorithms, however, are too coarse-grained and lead to limited sparsity utilization. Finer-grained sparsification methods adopt the software-hardware co-optimization fashion [14]–[16]. These works first map the weights to crossbars and then prune the whole crossbar-columns or crossbar-rows. The weight matrix change asks for extra peripheral circuits to coordinate the shape of input and output feature maps. Moreover, DNNs have to be retrained, which is not always feasible in a real-world scenario [8].

The second source of the vast requirement of ReRAM crossbars derives from the small bit-width, i.e., 1-3 bits [17], that a ReRAM cell can stably store due to process imperfections and limitation. Thus, quantization is necessary to reduce the bit-width of the weight. Most existing ReRAM-crossbar accelerators quantize the weight in 8-bits and decompose the 8 bits to 8 cells, respectively. The resulting high-degree *bit-level sparsity* is difficult to exploit. A sparse ReRAM-crossbar architecture SRE [18] attempts to exploit fine-grained sparsity derived by exchanging the crossbar-columns and crossbar-rows. This design can also exploit the bit-level sparsity. However, the hardware overhead, which is brought by the complex control, costly indexing and routing, almost offsets the area-efficiency derived from the reduced crossbars.

The fundamental limit of exploiting the sparsity is because *the data mapping and the VMM computation are tightly coupled with the crossbar structure*, denoted as *structural-coupling problem*. To solve this problem, in this paper, we devote to exploit the bit-level sparsity to improve the area- and energy-efficiency of ReRAM-crossbar based DNN accelerators. We propose an algorithm-hardware co-design framework called SME by novel weight mapping schemes and data path design to squeeze out the bit-wise sparsity. SME can apply to many quantization methods, and it is training-free and orthogonal to existing pruning methods. The contributions are summarized as follows:

- We propose a bit-wise sparse pattern and an inter-crossbar bit-slicing scheme to accumulate the 0-bits to the same

Li Jiang is the corresponding author.

crossbars.

- We propose a squeeze-out scheme that empties highly sparse crossbars by sacrificing limited amount of least-significant bits.
- We design the hardware architecture of SME with a limited 2 Kb overhead. The proposed SME reduces up to $8.7\times$ and $2.1\times$ crossbars for ResNet-50 and MobileNet-v2, respectively, compared with the SOTA method.

II. Background and Motivation

A. ReRAM-based Sparse NN Accelerators

The *Structural-coupling problem* manifests itself as the inability to freely skip the multiplication of zero operands because weight-bits in the same crossbar-row share the same input, and the current derived by multiplication in cells are accumulated in the same crossbar-column. In Fig. 2(a), suppose each weight has 4-bit and is partitioned into four cells. If a single cell containing 0-bit is removed, other cells can not fill their position since they are from a different row or column. Moving weight-bits across crossbar-rows/columns leads to wrong MAC results without modifying peripheral circuits, as shown in Fig. 2(b).

Structural pruning methods avoid this problem by pruning the weights in a granularity that the whole crossbar-column (or -row) can be removed at the cost of extra peripheral circuits [14], [16]. The extra peripheral circuits, including input-fetching and output-alignment modules, process the raw feature maps into the pruned weight matrices mapped on the crossbars. We break down the peripherals' area overhead and find that PIM-Prune [16] needs 4KB index storage to skip fetching the unnecessary activation (multiplied by zero weight) for ResNet-50. To achieve finer pruning granularity, excessive peripheral circuits will be introduced to route modified matrices, while a coarse pruning granularity will inevitably modify weight values and thus requires finetuning to retrieve accuracy.

The weight matrix after structural pruning still contain many 0-bits. As a result, enormous ReRAM-cells, denoted as sparse cells, are mapped with 0-bits and do not contribute to the final result. SRE [18] breaks up crossbars into smaller parts, namely Operating Units (OUs), to exploit finer-grained sparsity. SRE utilizes the sparsity of empty OU rows and columns, which is easier to find than larger empty crossbar rows and columns. Thus, they can utilize more sparsity that is impossible to exploit before. However, SRE dramatically increases the peripheral circuit's overhead to accumulate the correct result and introduces 778KB index storage for ResNet-50. There is a dilemma between the sparse utilization and the OU size: shrinking the size of OUs can exploit finer-grained sparsity but significantly increase the index overhead and the routing overhead of control circuits (see the extra routing in Fig. 2(b)).

B. Weight Quantization for ReRAM-crossbar

The ReRAM cell is programmed into multiple conductance levels to represent a value, e.g., a 4-bit value requires 2^4 different levels. The ReRAM cell's process limitations [18] constrain the bit-width of the value a ReRAM cell can store. Consequently, a weight is conventionally segmented into multiple subwords and each subword is deployed on a ReRAM cell. This segmentation results in high crossbar costs. For example, ResNet-18 with 32-bit weights consumes more than $20,000$ crossbars of 128×128 size [17]. Thus, quantization is compulsory to reduce the bit-width of DNNs's weight.

On the one hand, some quantization methods only reduce the variety of weights to shorten the encoding bit-widths of weights. Quantization using weight clustering [20] and sharing [21] strive to reduce the number of values that represent weights, but these values are still floating numbers, which can not shrink the crossbar amount.

On the other hand, some quantization methods can actually reduce the bit-width of weights on crossbars. Uniform quantization like INT8 quantization [22] is thus a well-accepted quantization method for ReRAM-crossbar [6]. The resulting integer values can be well aligned and mapped to ReRAM-crossbar. Adaptive quantization methods, like HAQ [20], uniformly quantize weights using different bit-widths to optimize both the memory occupation and accuracy. Some other quantization methods such as POWER-OF-2 based quantization (PO2) [8], [21], [23] quantize values in the form of exponents. However, they may incur complex finetuning process and they also need to align the weights to the most and least significant bit of the whole crossbar, which means that the bit-width on crossbars will be larger than their encoding bit-width.

C. Motivation for Bit-Wise Sparsity Exploitation

As shown in Fig. 1, we can see a high degree of sparsity in the quantized weights, especially in the three most significant bits (MSB). In PO2 quantization, the least-significant bits (LSBs) also contains many sparsity. However, such bit-wise sparsity can hardly be exploited by ReRAM-crossbar accelerators due to the structural-coupling problem. The key is to decouple the crossbar structure (Section III).

III. SME Algorithm

We propose our SME scheme to decouple the crossbar structure by novel weight mapping algorithms. The SME mapping algorithm has three steps: quantization, inter-crossbar bit slicing and squeezing, as shown in Fig. 3.

A. Quantization and Encoding Scheme

We assume single-level ReRAM cell as an example throughout this paper for simplicity. We quantize and encode the weights by extending the APT quantization[1] [21] that represents the weight with the sum of several power-of-twos so that we can increase the bit-level sparsity in crossbars while retaining the values well to avoid finetuning. We map each N_q-bit weight onto N_q cells, $b_{1:N_q}$:

$$w^q = \sum_{i=1}^{N_q} b_i 2^{-i}, \quad b_i \in \{0, 1\} \tag{1}$$

[1]Other quantization methods, such as adaptive quantization [8], [20] can also apply to SME, which is discussed in Section V-C.

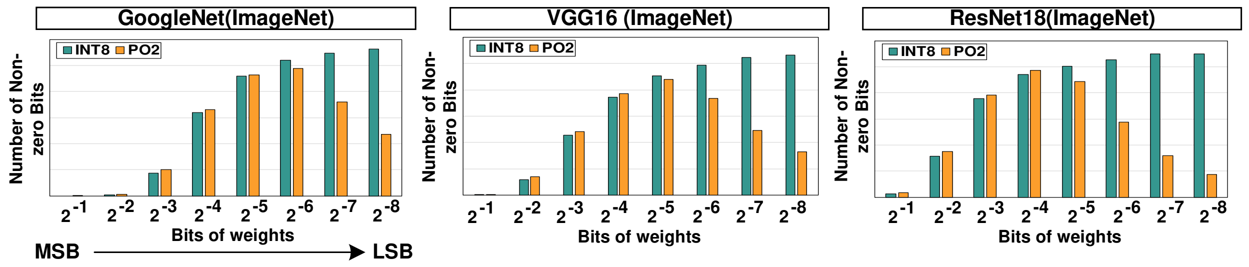


Fig. 1: Comparison of the INT8 and PO2 quantization in terms of the bit-level sparsity of weights in the quantized DNNs on ImageNet [19].

In Step ❶, we *quantize the weights into sum of power-of-twos, whose exponent are among S consequtive integers.* The above quantization can be derived by rounding the APT quantization result as follow:

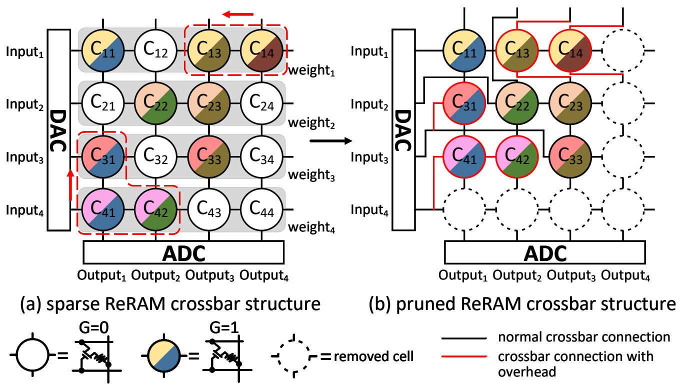
$$w^q = \sum_{i=k}^{\min\{N_q, k+S-1\}} b_i 2^{-i},\ k \in \{1, 2, \dots, N_q\} \qquad (2)$$

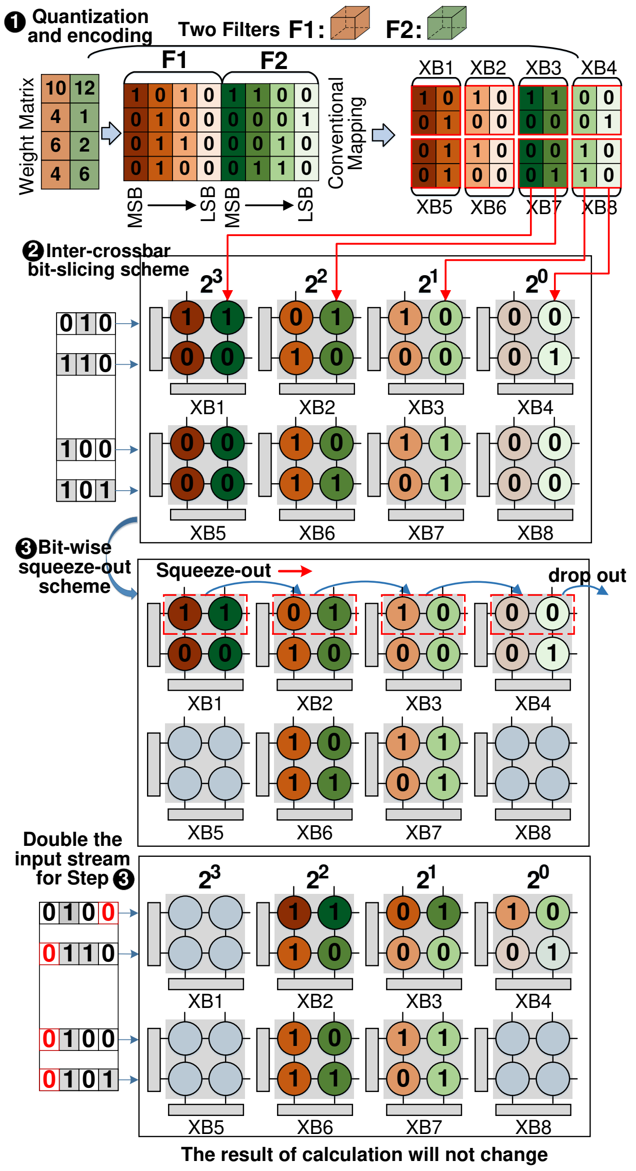
The maximum absolute value Eq. 2 can represent is $|w^q| \leq 1{-}2^{-S}$. Consequently, we scale all the weights into that range using a simple shift operation in the architecture (described in Section IV). Compared with the INT8 quantization, this quantization method can increase and accumulate the bit-level sparsity in a codeword, as shown in Fig 4.

B. Inter-crossbar Bit-slicing Scheme

To decouple the crossbar structure, we propose the **inter-crossbar bit-slicing** scheme. The key idea of *bit-slicing* is to map the same bit of quantized weights into the same bit crossbar, as shown in Step ❷. In this mapping process, a $W{\times}H$ weight matrix quantized with N_q bits is sliced into N_q bit-sliced matrices of size $W{\times}H$. Then, each bit-sliced matrix is further partitioned and mapped to ReRAM crossbars with size $xw \times xh$.

For example, in step ❷ of Fig 3, the bit matrix consisting of two filters is first bit-sliced into four bit matrices, $XB_{1,5}$ for the most significant bit, $XB_{2,6}$ for the second one and after that $XB_{3,7}$ and $XB_{4,8}$. Specifically, the first weight 1010 in $F1$ is sliced and mapped onto the top-left cells of XB_{1-4}. MSBs in filter $F1$ and $F2$ are mapped to crossbar XB_1 and XB_5. The above mapping scheme can aggregate the sparsity in a crossbar (e.g., XB_5 and XB_8), so that these empty crossbars can be saved by the mechanism of light-weight index [16], [18], [24].


Input₁
Input₂
Input₃
Input₄
DAC
ADC
weight₁
weight₂
weight₃
weight₄
Output₁ Output₂ Output₃ Output₄
(a) sparse ReRAM crossbar structure
(b) pruned ReRAM crossbar structure
G=0
G=1
= removed cell
normal crossbar connection
crossbar connection with overhead
Fig. 2: Crossbar row/column exchange and structural-coupling problem. The same color of the upper/lower semicircle indicates that the cells share the same input, and the cells' results are accumulated in the same bit-line, respectively.


❶ Quantization and encoding
Two Filters F1: F2:
Weight Matrix
F1
F2
MSB → LSB
Conventional Mapping
XB1 XB2 XB3 XB4
XB5 XB6 XB7 XB8
❷ Inter-crossbar bit-slicing scheme
2³ 2² 2¹ 2⁰
❸ Bit-wise squeeze-out scheme
Squeeze-out
drop out
Double the input stream for Step ❸
The result of calculation will not change
Fig. 3: The overview of proposed algorithm framework is composed of three parts. ①: the quantization and encoding scheme generates massive structured bit-level sparsity while retaining precision. ②: bit-wise and inter-crossbar mapping that utilize different level of sparsity in different bits. ③: squeeze-out scheme that decouples cell site and output result.

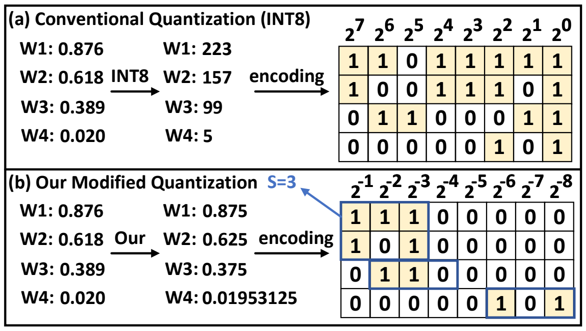


Fig. 4: The sparsity generated by INT8 and modified APT quantization whose '1' bits are constrained in consecutive 3 positions (marked in blue frame).

It is worth noting that this bit-slicing mapping scheme requires minor modification on the peripherals (refer to Section IV-B), and demands the same amount of peripheral circuits, such as ADCs, shifters, and adders, as conventional mapping method.

C. Bit-wise Squeeze-out Scheme

In the previous section, we aggregated a large amount of sparsity by the bit-slicing scheme so that some of the crossbars become empty and can be saved directly, but there are still crossbars that are very sparse and cannot be saved directly. We shrink the sparsity from the full crossbar to a smaller granularity, rows, since all the cells in the same crossbar-row share the same input. Fortunately, our SME approach make sure that the first few most-significant bit matrices are highly sparse. Fig. 5 shows that less than 10% non-empty rows in the most significant bit matrix in average. Based on above observation, we propose a clever squeeze-out scheme that circumvents the structure couple problem. The essence is row swapping among the crossbar group, but without introducing either overhead or large accuracy loss.

In Step ❸, we squeeze the crossbar-rows containing non-zeros in preceding XBs to the subsequent XBs until these rows in tailing XBs are dropped out. For example, the first crossbar-row in XB_1 is remapped to XB_2, whose first crossbar-row is shifted to XB_3. And the LSB crossbar XB_4 drops its first crossbar-row. Based on step ❶, release these crossbars will lead to no loss on network's accuracy. A corresponding operation on the input of these rows is performed. This step does not introduce extra indices for accumulating the output.

The following observations inspire the bit-wise squeeze-out scheme: the first few bit matrices are too sparse to compose

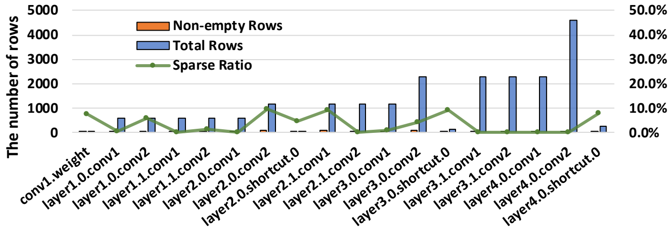


Fig. 5: Non-empty row statistics of crossbars stored MSB of the ResNet-18.

a single crossbar. According to our bit-wise sparse pattern, '1's will only appear in successive positions after its most significant bit, which means that for the weights whose first few bits are '1', their last few bits must be '0'. We can empty crossbars that store the MSB by squeezing-out these bits without changing the actual quantized weight.

After squeeze-out by one bit, the corresponding non-empty contents in XB_i^j are moved to XB_i^{j+1} $(j \in \{1, 2, \cdots, N_q\})$ and these contents in the last bit crossbar $XB_i^{N_q}$ are abandoned. According to Eq. (2), the value in that row is approximately halved with 1-bit squeezed. To ensure the invariance of the calculation results, we propose a scheme to double the input (refer to Step ❸ in Fig. 3). Note that we can even perform this step iteratively to squeeze-out multiple bits.

In this paper, we use the same style of input as in ISAAC [1]. The input is converted into bit-serial voltage and the number of cycles required is equal to the bit-width of the input. If we squeeze out with x-bit, we delay the input of these rows for x clocks, which implemented by Fig. 6 🅑.

Assume that the weights and inputs are both 4-bit and we perform the squeeze-out scheme for 1-bit. For example, in Fig. 3 ❷, the first row in XB_1 is non-empty. Starting from XB_1, the crossbar's non-empty rows are placed at the same position in the latter crossbar, and the rows of the last crossbar (XB_4 in Fig. 3) are dropped out. After that, the crossbar XB_1 storing MSB (i.e., 2^{-1}) can be saved, and the bit-width of weights changes from 4- to 3-bit. The bits representing 2^{-1} in XB_1 is moved to XB_2 and represent 2^{-2}, which means the part of weight shrinks by half (i.e., W_1 changes from $10 = 1010_{(2)}$ to $5 = 0101_{(2)}$). Thus, we shift the input of the first row one bit to the left ($input \times 2$), which acts on the first row in each remaining crossbar (i.e., XB_2 $\sim$$\mathrm{XB}_4$) to make up for the changes in weight, i.e. $I_1 \times W_1 = (I_1 \times 2) \times (W_1/2)$. This process equals to delaying the input one cycle, which means that the cycle of input changes from 4 to 5. As a result, the total amount of computation changes from $4 \times H \times W \times 4$ to $5 \times H \times W \times 3$, the computation has been reduced.

IV. SME Architecture

A. Architecture Overview

We present the overview of SME architecture, aiming at inference in edge devices. As shown in Fig. 6, each bank consists of three parts, all of which are connected to the shared bus: 1) the controller decodes instructions and provides control signals to all the peripheral circuits; 2) in-situ Computation Units (CU) is the core computing and storage unit; 3) the shared blocks contain the activation unit, pooling unit, and eDRAM buffer for storing activations (i.e., intermediate computing results). The SME add-on hardware implements the computation function of matching our algorithm, including simple modifications to the existing crossbar peripheral circuits, which is easier to manufacture than integrate complex logic into the chip.

B. Module and Dataflow

Controller. Controller in Fig. 6 🅐 provides control signals to all the peripheral circuits and drive the finite state machines

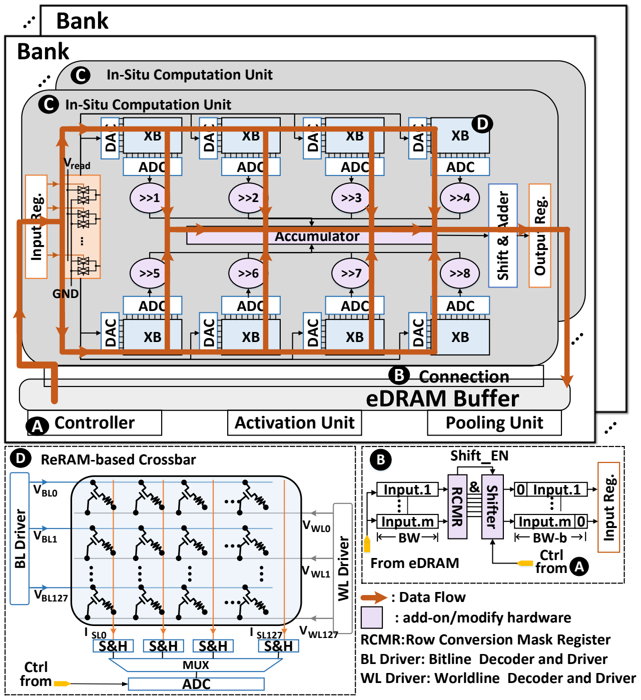


Fig. 6: Architecture Overview and Data Path of SME.

that steer the inputs and outputs correctly after every cycle based on the technique configurations.

In-situ Computation Unit. Fig. 6 C shows the CU that is composed of crossbars and peripheral circuits. It includes DAC/ADCs for data conversion, accumulator and shifters, which sum up the partial sums of crossbars, and shift-adder unit for aggregating the partial results of input cycles.

First, the input is in the form of the bit sequence, and then each cycle enters respectively 8 crossbars to the same in-situ CU to carry out MAC operation, and each gets the output currents. After output current generated on SL, first, latch the computation results by S&H circuits, and then send 128 analog voltages to ADC to convert them into digital signals through a MUX. In each cycle, 8 crossbars are sampled in parallel, and a calculation result of different bits of the same weights is obtained through the corresponding shifter is connected to each crossbar. Then, these results are sent to the accumulator to generate the complete calculation results of the weight. Finally, the results transmit into the output register for updating the outputs by shifting and adding. After traversing the 128 samplings, the next cycle calculation is repeated. **ReRAM-based crossbar.** Fig. 6 D shows the ReRAM-based crossbar with 128 $\times$ 128 size, which perform parallel MAC operations. We adopt the SLC as the ReRAM cell since SLC is more reliable against process variation compare to the MLC counterpart. The ReRAM array is implemented with a one-transistor–one-memristor (1T1R) cell structure, in which one resistance cell with a transistor to control the write current to facilitate more precise writes to ReRAM cells. Moreover, the peripheral circuits, including the word-line driver, sample-and-hold blocks, multiplexer (MUX) and ADCs. The output analog signals are transmitted to ADC via MUX to control analog-to-digital conversion.

Buffer Connection, which supports the squeeze-out strategy. Fig. 6 B shows the communication between eDRAM Buffer and input register. The RCMR is used for fetching inputs from buffer to the register, which needs to convert the bit-width of input enable increase input cycle caused by the squeeze-out strategy (refer to Section III-C). Data from the buffer pass through the RCMR and the controller determines whether the squeeze-out scheme needs to be performed at the current layer, and if it does, the inputs bit-width will be extended to $(8+x)$-bit (we initially default the weights were quantized to 8-bit). The shifter follows the RCMR, '1' means to shift x-bit to the left, while '0' indicates padding x zeros in front of MSB. After that, data transmit into the input register.

C. Latency Analysis

In the proposed architecture, different configurations of the crossbar and our bit-wise squeeze-out scheme (mentioned in Section III-C) will directly influence the computing latency and the hardware overhead. The main design parameters of PEs include the design choices of the degree of squeeze-out and the crossbar size. This subsection will introduce the analysis of design choices and hardware performance.

Assume the crossbar size is $K \times K$, M_w is the bit-width of weights, and M_a is the bit-width of inputs. We map $K \times K$ weights in M_w crossbars and feed the input in through the DACs serially 1-bits at a time. The MAC operation is finished in M_a cycles. The squeeze-out operation begins with inter-crossbar bit-slicing scheme. Here, we index the sliced crossbars from MSB to LSB with XB_m, and $m = [1; M_w]$, such that $m = 1$ and $m = M_w$ are corresponding to MSB and LSB, respectively. D is the degree of being squeezed, representing the number of iteration in squeeze-out scheme. Specifically, the squeeze-out degree increases by 1 with squeezing 1-bit along the direction of bit-width. We take Fig. 3 as an instance to intuitively illustrate this process, $D = 1$ denoted as sliced crossbars change from $XB_1 \sim XB_8$ to $XB_2 \sim XB_8$. Those kernels mapped in sliced crossbars could be duplicated in different computation units and take multiple input data to generate independent outputs simultaneously. In this case, we need to duplicate $K \times M_w \times M_a$ kernels which allows to speed up without performing the squeeze-out scheme. If we further squeeze-out with D bits, resulting in a reduction in the number of crossbar by D. However, the cycles it takes for MAC operation to complete increase from M_a to $M_a + D$. The rows are partitioned into two parts according to the performed squeeze and the none performed, denoted as A and B respectively. Then we can calculate the number of duplicates needed to achieve the speedup. Only if the number of duplicates needed after the execution of squeeze is smaller than that of unexecuted ones, the optimization will have a performance benefit. Its formulated expression can be mathematically described as:

$$\begin{aligned} &\big((M_a + D) \times A + M_a \times B\big) \times (M_w - K) \\ &= KM_wM_a - KM_wD + DAM_w - D^2A \leq KM_wM_a \quad (3) \\ &\text{s.t. } A + B = K \end{aligned}$$

Without loss of generality, in this paper, we take $K = 128$, $M_a = 8$ and $M_a = 8$. If we squeeze 1-bit, i.e., set $D = 1$, then the overall performance gain increases as long as the squeezed rows are less than 146 (i.e., $A \leq 146$). As described in Fig. 1, almost all the layers in ResNet-18 satisfy this condition. According to Eq. (3), we can find that: (1) the larger the degree of being squeezed D, the size of crossbar K and input bit-width M_a, the more feasible the scheme; (2) when the weight bit-width M_w is getting smaller, the scheme becomes more feasible accordingly (i.e., A is larger).

V. EXPERIMENTS

A. Experimental Setup

TABLE I: Parameters of the CU in SME Architecture

In-situ Computation Unit Hardware Configuration (1GHz, 32nm process, 128 banks per chip, 12 CUs per bank)				
Components	**Param**	**Spec**	**Area** ($mm^2 \times 10^{-3}$)	**Power (mW)**
MUX Array	number	8	4.23	0.56
S&H	number	8×128	0.03125	0.0011
ADC	resolution	6-bit	4.7	5.14
	number	8		
	frequency	1.2GSps		
DAC	resolution	1-bit	0.34	4.25
	number	8×128		
Memristor Array	size	128×128	2.03	1.3
	bits-per-cell	1		
	number	8		

Table I summarizes the parameters and their power/area values of each CU in our SME. We redesign the data path since we add some peripheral circuits to integrate the techniques. Moreover, the energy consumption and area overhead of memories, including eDRAM buffer, input/output register, and registers stored mask (i.e., RCM Register), are calculated with CACTI [25] based on the 32-nm CMOS process. For ADC and DAC, the model from [26] is used. The memristors adapted SLC-based ReRAM devices with a resistance range of $10K\Omega \sim 100K\Omega$ and crossbar size set as 128×128. We model the power and area for ReRAM devices using the results from [27]. We modify NVSim [28] with these models to estimate time, area and energy consumption.

We use the ISAAC as the baseline. We evaluate our work on classical image classification task, using several representative DNNs (VGG-16 [29], ResNet-18/50 [30], MobileNet-v2 [31]) on CIFAR-10 [32] and ImageNet ILSVRC-2012 [19]. We compare with the PIM-Prune [16], SRE [18], SmartExchange [24]. For results that not available, we reproduce their experiments and report the results. Our method can also combine with other sparsity utilization solutions, such as SRE, PIM-Prune, etc. We implement our SME algorithm framework in the Pytorch framework [33] to valid it.

B. Results and Analysis

1) Accuracy and Sparsity: Tab. II first shows the NN accuracy for the networks on CIFAR-10 and ImageNet datasets. We can observe that our SME and other solutions are orthogonal and can further improve the effect when combined with

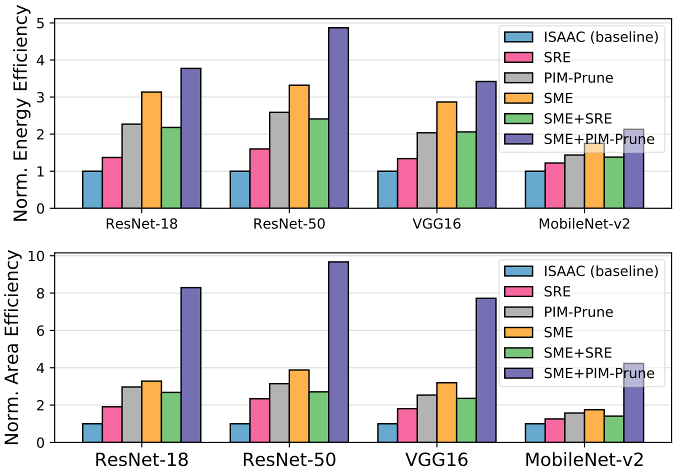


Fig. 7: Normalized energy- and area-efficiency on various neural networks.

the exciting methods with negligible accuracy loss, even on the large-scale dataset. Specifically, for ResNet-50 on the ImageNet, SME combined with PIM-Prune achieves 91.23% sparse rate with 0.6 accuracy loss compared to SmartExchange (58.6% sparse rate with 2.07% accuracy loss) and PIM-Prune (71.91% sparse rate with 1.10% accuracy loss) on ImageNet.

TABLE II: Inference accuracy on ImageNet

Model	Ori.Acc (%)	Method	Acc.(%)	Sparsity (%)
		CIFAR-10		
VGG-16	93.66	SmartExchange	92.87	92.80
	93.7	PIM-Prune	93.23	93.10
	93.7	**SME**	**93.6**	**84.15**
	93.7	**SME+PIM-Prune**	**93.18**	**97.11**
ResNet-18	94.58	SmartExchange	94.54	91.30
	94.14	PIM-Prune	93.84	91.71
	94.14	**SME**	**94.19**	**83.19**
	94.14	**SME+PIM-Prune**	**93.85**	**96.97**
		ImageNet ILSVRC-2012		
ResNet-50	76.13	SmartExchange	74.06	58.60
	76.13	PIM-Prune	74.91	71.91
	76.13	**SME**	**76.03**	**67.35**
	76.13	**SME+PIM-Prune**	**75.46**	**91.23**
MobileNet-v2	72.19	SmartExchange	70.16	79.79
	71.88	PIM-Prune	70.11	77.13
	71.88	**SME**	**71.57**	**78.74**
	71.88	**SME+PIM-Prune**	**71.02**	**84.51**

2) Energy- and Area-Efficiency: Fig. 7 shows the energy- and area-efficiency of different accelerators for the four networks. We normalize the energy-efficiency to that of the model without any compression. On average, for ImageNet, SME improves energy efficiency by 2.3× and area efficiency by 6.1× on ResNet-18/50 compared to PIM-Prune and SRE. Even on MobileNet-v2, our method is still superior to the existing methods. The reason is that the pruning-based methods are difficult to compress networks for large-scale datasets or compact networks with acceptable accuracy. However, high-degree bit-level sparsity always exists and can be used by our SME. Even on MobileNet-v2 the area-efficiency, our method also improves 2.7×. The energy and area reduction against PIM-Prune and SRE is mainly due to the reduced crossbar resources and less overhead. Compared to PIM-Prune and

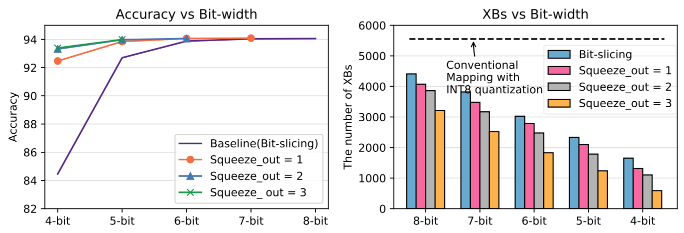


Fig. 8: Comparison of NN accuracy and the number of crossbars with our different scheme on ResNet-18. Baseline is quantized with INT8 method.

SRE, SME improves energy efficiency by $2.3\times$ and area efficiency by $6.1\times$ on average.

3) Varied squeeze-out schemes with crossbar resource: Fig. 8 reports the results with the different squeeze-out schemes. We respectively compare the accuracy and necessary crossbar resources of squeezing 1,2,3 bits. We use the squeeze-out scheme to reduce the number of cells representing weights far better than directly reducing because the MSBs are more critical than the LSBs. If we can reduce the error caused by the MSBs, the overall error can be effectively decreased [17].

4) Sweet-spot for the size of consecutive region containing '1': As our discussion in section III-A, Fig. 9 shows the trade-off between the sparsity and quantization error caused by different number S of consecutive '1'. We use mean square error (MSE) to measure the loss caused by quantization, which is defined as the absolute difference between the exact and the approximate weights [8]. In Fig. 9, we find if we set $S = 2$, the overall sparsity of the network began to decrease, while $S = 4$, the overall error of the model, is almost zero. However, we combine with the overall sparsity and the bit-level sparse distribution. We can find that $S = 3$, SME achieves an optimal point for ResNet-18.

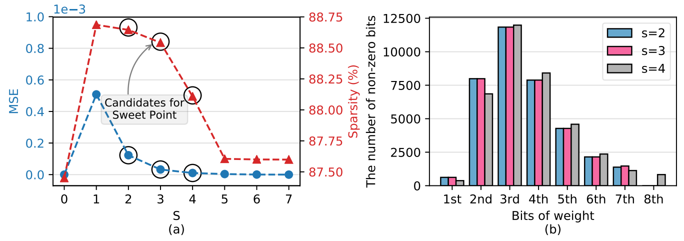


Fig. 9: (a) Trade-off between quantization error and bit-level sparsity along with the change of the consecutive region's size containing '1.' w.r.t. S. (b) The sparsity of bit-level under different S in the candidate set for sweet point.

5) Overhead Analysis: Fig. 10 shows the storage overhead of different networks. On average, SME achieves 84.6% and 98.1% register overhead reduction compared to PIM-Prune and SRE with only quantization and bit-slicing scheme; achieves 77.8% and 96.8% register overhead reduction compared to PIM-Prune and SRE further combining with the squeeze-out scheme. However, the significant reduction in overhead benefits from two parts: (1) our squeeze-out scheme solves the index's problem for aligning the output by processing the input. (2) we retain crossbars if the crossbar cannot be released, so the index is continuous. Specifically, for ResNet-50, compared to SRE, which requires the 778KB register for the index, our SME consumes nearly 100% (i.e., without input index) and 95.6% less storage of input index and output index. However, the significant reduction in overhead benefit from our algorithm framework design, the padded output is continuous by our SME, since we aim to reduce the crossbar resources. If the crossbar resources cannot be released, the full crossbar is retained, which is only represented by the special symbol that skips the current crossbar's routing.

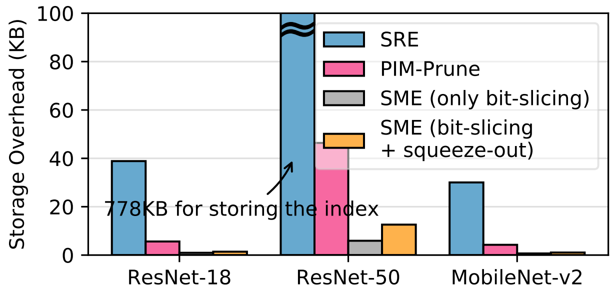


Fig. 10: Comparison of storage overhead with different networks.

C. Design Exploration

In this section, we discover our method can support the network with intra-layer mixed-precision [20], and also support MLC-based crossbar but perform better on SLC.

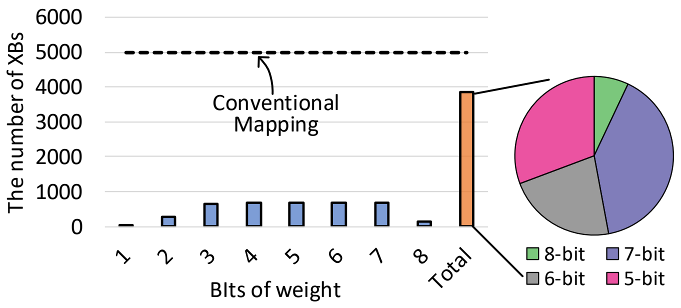
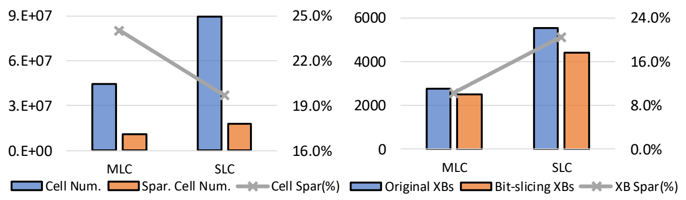


Fig. 11: The number of crossbar for ResNet-18 with mixed-precision.

1) Support intra-layer mixed-precision: Fig. 11 shows the crossbar (128×128) consumption of ResNet-18 quantized by intra-layer mixed-precision under conventional mapping and our SME method. The pie chart of the Fig. 11 also shows the mixed-precision contains 5 to 8-bit in layers of ResNet-18. The conventional mapping approach cannot take advantage of mixed-precision benefits due to structural-coupling. The weights within a filter are mapped to crossbars, and the maximum bit-width of weights determines the number of cells required per weight. So there are massive sparse cells. In contrast, the SME slices each bit of the weights into different crossbars, aggregating the bits' sparsity. This achieves decoupling of the crossbar structure, reducing over $1,000$ crossbars than the conventional mapping method.

2) Support MLC-based crossbar: Fig. 12 shows that our bit-slicing scheme is also applicable to MLC-based crossbar. The number of sparse cells is significantly decreased when the network is mapped onto the MLC-based crossbar. Thus, our bit-slicing scheme's benefit is also affected, as we reduce



(a) The cell statistics generated by deploying on SLC and MLC-based crossbars

(b) Comparison of crossbar savings mapped in SLC and MLC-based crossbar

Fig. 12: Comparison of ResNet-18 deployed on SLC/MLC-based cell.

approximately 11% crossbars compared to the conventional mapping scheme. Besides, we can use the squeeze-out scheme to save resources further, and squeeze one MLC-based cell is equivalent to squeezing 2-bit on the SLC-based crossbar.

VI. Conclusion

Bit-level sparsity cannot be utilized, leading to the limited performance of NN inference. We propose **SME**, an algorithm-hardware co-design framework that decouples the hardware dependence of multiplication to release the sparse cells in the crossbars for higher energy-/area-efficient inference of NNs. Besides, we design the architecture to efficiently support our algorithm through well-designed crossbars with the peripheral circuit. Our evaluation shows that the proposed SME outperforms other similar solutions in energy, area, and accuracy.

Acknowledgment

This work was partially supported by the National Natural Science Foundation of China (Grant No. 61834006, 62102257), National Key Research and Development Program of China (2018YFB1403400).

References

[1] A. Shafiee, A. Nag, N. Muralimanohar *et al.*, "Isaac: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," *ACM SIGARCH Computer Architecture News*, 2016.

[2] F. Liu, W. Zhao, Z. He *et al.*, "Bit-transformer: Transforming bit-level sparsity into higher preformance in reram-based accelerator," in *ICCAD*, 2021.

[3] L. Song, X. Qian, H. Li, and Y. Chen, "Pipelayer: A pipelined reram-based accelerator for deep learning," in *HPCA*. IEEE, 2017.

[4] F. Liu, W. Zhao *et al.*, "Im3a: Boosting deep neural network efficiency via in-memory addressing-assisted acceleration," in *Proceedings of the 2021 on Great Lakes Symposium on VLSI*, 2021.

[5] T. Chou, W. Tang, J. Botimer, and Z. Zhang, "Cascade: Connecting rrams to extend analog dataflow in an end-to-end in-memory processing paradigm," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019, pp. 114–125.

[6] P. Yao, H. Wu, B. Gao, J. Tang, Q. Zhang, W. Zhang, J. J. Yang, and H. Qian, "Fully hardware-implemented memristor convolutional neural network," *Nature*, vol. 577, no. 7792, pp. 641–646, 2020.

[7] R. Guo, Z. Yue, X. Si, T. Hu, H. Li, L. Tang, Y. Wang, L. Liu, M.-F. Chang, Q. Li *et al.*, "15.4 a 5.99-to-691.1 tops/w tensor-train in-memory-computing processor using bit-level-sparsity-based optimization and variable-precision quantization," in *2021 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 64. IEEE, 2021, pp. 242–244.

[8] 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.

[9] P. Chi, S. Li, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, and Y. Xie, "Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 27–39, 2016.

[10] S. Angizi, Z. He, A. S. Rakin, and D. Fan, "Cmp-pim: an energy-efficient comparator-based processing-in-memory neural network accelerator," in *DAC*, 2018, pp. 1–6.

[11] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *arXiv preprint arXiv:2005.14165*, 2020.

[12] J. Lin, Z. Zhu, Y. Wang, and Y. Xie, "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*, ser. ASPDAC '19, 2019.

[13] L. Liang, L. Deng, Y. Zeng, X. Hu, Y. Ji, X. Ma, G. Li, and Y. Xie, "Crossbar-aware neural network pruning," *IEEE Access*, 2018.

[14] P. Wang, Y. Ji, C. Hong, Y. Lyu, D. Wang, and Y. Xie, "Snrram: an efficient sparse neural network computation architecture based on resistive random-access memory," in *DAC*. IEEE, 2018, pp. 1–6.

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

[16] C. Chu, Y. Wang, Y. Zhao, X. Ma, S. Ye, Y. Hong, X. Liang, Y. Han, and L. Jiang, "Pim-prune: fine-grain dcnn pruning for crossbar-based process-in-memory architecture," in *DAC*. IEEE, 2020, pp. 1–6.

[17] Y. Cai, T. Tang, L. Xia, B. Li, Y. Wang, and H. Yang, "Low bit-width convolutional neural network on rram," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2019.

[18] T.-H. Yang, H.-Y. Cheng, C.-L. Yang, I.-C. Tseng, H.-W. Hu, H.-S. Chang, and H.-P. Li, "Sparse reram engine: Joint exploration of activation and weight sparsity in compressed neural networks," in *Proceedings of the 46th International Symposium on Computer Architecture*, 2019.

[19] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[20] K. Wang, Z. Liu, Y. Lin, J. Lin, and S. Han, "Haq: Hardware-aware automated quantization with mixed precision," in *CVPR*, 2019.

[21] Y. Li, X. Dong, and W. Wang, "Additive powers-of-two quantization: An efficient non-uniform discretization for neural networks," in *8th International Conference on Learning Representations (ICLR)*, 2020.

[22] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[23] F. Liu, W. Zhao, Z. He *et al.*, "Improving neural network efficiency via post-training quantization with adaptive floating-point," in *ICCV*, 2021.

[24] Y. Zhao, X. Chen, Y. Wang, C. Li, H. You, Y. Fu, Y. Xie, Z. Wang, and Y. Lin, "Smartexchange: Trading higher-cost memory storage/access for lower-cost computation," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2020.

[25] R. Balasubramonian, A. B. Kahng, N. Muralimanohar, A. Shafiee, and V. Srinivas, "Cacti 7: New tools for interconnect exploration in innovative off-chip memories," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 14, no. 2, pp. 1–25, 2017.

[26] D. Fujiki, S. Mahlke, and R. Das, "In-memory data parallel processor," *ASPLOS*, vol. 53, no. 2, pp. 1–14, 2018.

[27] M.-F. Chang, J.-J. Wu, T.-F. Chien, Y.-C. Liu, T.-C. Yang, W.-C. Shen, Y.-C. King, C.-J. Lin, K.-F. Lin, Y.-D. Chih *et al.*, "19.4 embedded 1mb reram in 28nm cmos with 0.27-to-1v read using swing-sample-and-couple sense amplifier and self-boost-write-termination scheme," in *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*. IEEE, 2014, pp. 332–333.

[28] X. Dong, C. Xu, Y. Xie, and N. P. Jouppi, "Nvsim: A circuit-level performance, energy, and area model for emerging nonvolatile memory," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 31, no. 7, pp. 994–1007, 2012.

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

[30] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016, pp. 770–778.

[31] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4510–4520.

[32] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.

[33] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *arXiv preprint arXiv:1912.01703*, 2019.