Bit-Transformer: Transforming Bit-level Sparsity into Higher Performance in ReRAM-based Accelerator

Fangxin Liu[1,2], Wenbo Zhao[1,2], Zhezhi He[1], Zongwu Wang[1], Yilong Zhao[1], Yongbiao Chen[1], and Li Jiang[1,2,*]
1. Shanghai Jiao Tong University, Shanghai, China
2. Shanghai Qi Zhi Institute, Shanghai, China
{liufangxin, zhaowenbo, wangzongwu, zhezhi.he, ljiang_cs}@sjtu.edu.cn

***Abstract*—Resistive Random-Access-Memory (ReRAM) crossbar is one of the most promising neural network accelerators, thanks to its in-memory and in-situ analog computing abilities for Matrix Multiplication-and-Accumulations (MACs). Nevertheless, the number of rows and columns of ReRAM cells for concurrent execution of MACs is constrained, resulting in limited in-memory computing throughput. Moreover, it is challenging to deploy Deep Neural Network(DNN) models with large model size in the crossbar, since the sparsity of DNNs cannot be effectively exploited in the crossbar structure.**

As the countermeasure, we develop a novel ReRAM-based DNN accelerator, named Bit-Transformer, which pays attention to the correlation between the bit-level sparsity and the performance of the ReRAM-based crossbar. We propose a superior bit-flip scheme combined with the exponent-based quantization, which can adaptively flip the bits of the mapped DNNs to release redundant space without sacrificing the accuracy much or incurring much hardware overhead. Meanwhile, we design an architecture that can integrate the techniques to massively shrink the crossbar footprint to be used. In this way, It efficiently leverages the bit-level sparsity for performance gains while reducing the energy consumption of computation. The comprehensive experiments indicate that our Bit-Transformer outperforms prior state-of-the-art designs up to $13\times$, $35\times$, and $67\times$, in terms of energy-efficiency, area-efficiency, and throughput, respectively.

***Index Terms*—Processing-in-memory, Neural Network, Hardware Accelerator, Sparsity**

I. Introduction

In the last few years, deep learning techniques (e.g. DNN) have achieved great success in various computer vision [1], [2] and natural language processing tasks [3]. The fast development of this field shows a significant momentum of drastically growing model size from groundbreaking model AlexNet [1] (61M parameters) to the surprisingly powerful GPT-3 [3] (175B parameters). On the one hand, the increasing model size leads to an increasing accuracy. On the other hand, the long-distance data communication in conventional Von-Neumann architecture is emerging as the computation bottleneck for data-intensive computation tasks (e.g., neural network inference). This bottleneck is well known as "memory wall" [4]. Thus, designing a more efficient accelerating platform is urgent to match the rapidly rising model size and computation workload of DNNs.

As the countermeasure, ReRAM crossbar [5] emerges as a promising solution, owing to its Computing-In-Memory (CIM) capability that can mitigate massive data movements between on/off-chip memory. Moreover, its in-situ current mode weighted summation operation intrinsically matches the dominant MAC operations in the neural network, making it one of the most promising DNN accelerators [5]–[8].

A certain number of ReRAM crossbar-based DNN accelerator designs are built in prior works, such as ISAAC [9], PRIME [4], PipeLayer [10], and CASCADE [6]. With the increasing model size, various model compression methods are developed to reduce the model size and increase throughput. Based on these accelerator designs, several efforts are made to utilize the sparsity in ReRAM crossbar structures, such as SRE [11], PIM-Prune [12] and TraNNsformer [13]. Nevertheless, those works are still subject to several weaknesses:

- Retraining is usually needed to recover the accuracy degradation caused by increasing sparsity. Such process is time-consuming and requires accessing to the training data, which is not always feasible in real-world scenarios due to users' privacy and security concerns.
- Complicated peripheral circuits with complex operations are introduced to utilize the unstructured sparsity, which will increase the overhead and reduce the area-efficiency.
- Too idealistic crossbar structure is considered, which ignored the maximum number of rows that can be executed at the same time.

In this paper, we propose an algorithm-hardware co-design framework using Single-Level-Cells (SLCs) ReRAM crossbar, called **Bit-Transformer**, to address these weaknesses. Our contribution could be summarized as:

- We propose a 3-dimension bit-wise mapping scheme and an accompanying bit-flip scheme to introduce crossbar-friendly bit-wise sparsity without retraining the model.
- We design a novel ReRAM-based architecture, which integrates the algorithm framework and only modifies the existing architecture with small overhead.
- We have conducted comprehensive experiments upon various datasets and neural network benchmarks. The results indicate that our Bit-Transformer outperforms prior state-of-the-art designs up to $13\times$, $35\times$, and $67\times$, in terms of energy-efficiency, area-efficiency, and throughput respectively.

The remainder of this paper is organized as follows: Section II introduces the background and motivation of the

*Corresponding author: Li Jiang.

proposed design. Section III describes the algorithm for Bit-Transformer. Section IV elaborates the proposed the architecture and circuit. Section V present the experimental methodology and experiment results. Finally, Section VI concludes this paper.

II. Background and Motivation

A. DNN Model Compression

Two main streams of DNN model compression techniques are weight quantization and weight pruning/sparsification. Especially, quantization is a necessary step to deploy the network on hardware and accelerate DNN inference.

Weight Quantization attempts to reduce the bit-width of operands in the calculation, which shrinks the model size for memory saving and simplify the operations for acceleration. Many of the prior works [14], [15] pursue quantizing the model with extremely low bit-width. They have to adopt an iterative quantization and retraining/fine-tuning method to learn the network characteristic from the degree of accuracy degradation during quantization steps. The power-of-two based quantization [16] quantizes weights into the sum of several power-of-twos (Eq. 1), which is a good candidate for quantization to facilitate hardware implementation. These methods quantize weights into the sum of multiple power-of-twos with negligible accuracy loss and the binary encodings of the quantized weights naturally match the SLC-based crossbar:

$$Q(weight) = \gamma \times (\sum_{i=1}^{n} 2^{p_i}), \quad p_i \in \mathbb{Z} \tag{1}$$

Where γ is a scaling coefficient to make weight into the range of $[0, 1]$, p_i is the exponent of power-of-two and n is the number of power-of-two terms for superposition.

Network Sparsification attempts to reduce the number of operands and operations in the calculation to leverage the redundancy in weights.

Existing works [12], [13], [17], [18] mainly focus on the structured compression (i.e., quantization and sparsification) for practical acceleration and the compression object is the numbers (i.e., weights). Besides, these works lead to user-unfriendliness because it normally need to search the optimal hyperparameters or require model retraining to mitigate the accuracy degradation, which is time-consuming. For example, PIM-Prune [12] partitions the weight matrix into blocks and further reorders the columns and rows in each block to cluster non-zero weights onto the same crossbar for coarse-grain pruning with more than 200 epochs fine-tuning.

B. ReRAM-Based DNN Acceleration

ReRAM crossbar is emerging as a promising solution to mitigate problems such as memory wall, owing to its high memory accessing bandwidth and high density [4], [6], [9], [10], [19]–[23]. The existing researches about ReRAM-based NN accelerators use a mass of crossbar arrays and treat the ReRAM crossbar as a low-precision, low-energy and high-speed dot-product engine.

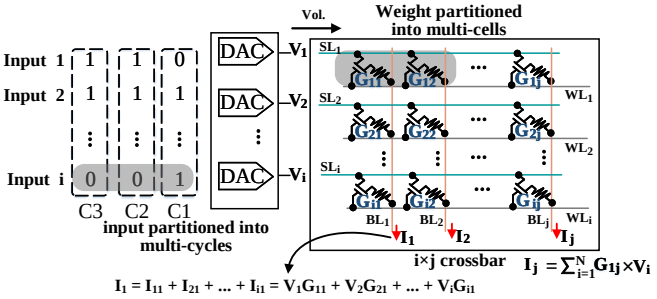

Fig. 1: In-situ computation based on the ReRAM crossbar.

Digital inputs are first converted into analog voltages using digital-to-analog converters (DACs). These voltages are applied on each word-line and the output current on each bit-line collects the current of all the cells in that column according to the Kirchhoff's law. Then, the gathered current is the weighted sum of inputs, which implemented the function of MACs. Lastly, analog-to-digital converters (ADCs) convert these current into digital value to go through further calculation (as shown in Fig. 1).

However, there are several problems that impede the practical application of ReRAM crossbar accelerators. Firstly, the cell arrangement is tightly coupled with the computation result, making it hard to utilize the sparsity of single cells. Even if a zero-leveled cell is detected and do not contribute to the output value in that bit-line, it can not be removed since it shares the input with other cells on the same row. Also, the capacity of ADCs and the non-linearity current-voltage characteristic limits that a single input is split into bits to be fed in several cycles and multi-level cells which store multiple bit in one ReRAM cell degrade the accuracy.

Most Importantly, the non-ideal effects limit the number of rows and columns that can be executed at the same time. For example, the IR drop caused by wire resistance lead to a huge voltage drop at the target cell and limits the number of columns that can be executed at the same time [8], [11], [24]. In addition, the maximum workload of the analog-digital converters that reads the gathered current in bit-lines limits the number of rows that can be executed at the same time, especially the maximum number of low-conduction state cells in a single column. Therefore, there are only 9 rows, and 8 columns of ReRAM cells on a 512×256 crossbar for concurrent execution of MACs in the macro of a state-of-the-art ReRAM-crossbar acceleration [25] in 65nm process.

C. Motivation

Because of the problems that current ReRAM-crossbar accelerators have, we propose our **Bit-Transformer** to improve the current design to meet the following properties:

1) The design use single-level cells (SLCs) to store the weights, which means that each cell will only save one bit, 0 or 1.
2) The design can utilize the bit-level sparsity of SLCs to increase the throughput of our accelerator without introducing complicated peripheral circuits. Specifically,

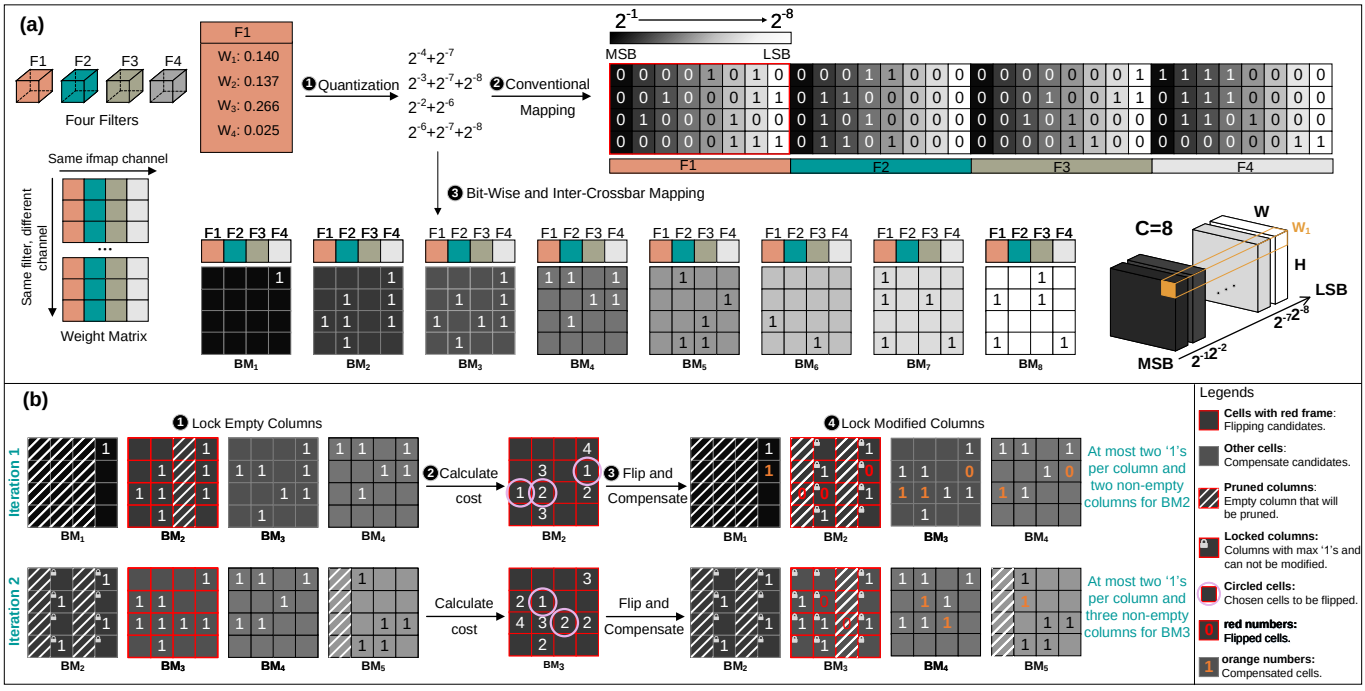

Fig. 2: The overview of Bit-Transformer algorithm. (a) We first quantize the weights into the sum of multiple power-of-twos. Then, we compare the traditional and our bit-wise and inter crossbar mapping. (b) We generate hardware-friendly bit-wise sparsity using our bit-flip and compensation scheme, while minimizing the value bias generated on weights.

each column will contain limited number of '1's, so that more number of rows can be executed concurrently.

3) The design doesn't require retrain to recover the accuracy brought by increasing the sparsity. In addition, it would be a bonus if finetuning could provide an increase in accuracy.

III. Approach

A. Overview

We propose our Bit-Transformer algorithm framework to exploit the bit-wise sparsity of weight matrix, i.e., the "0" bits in the codeword, with the attempt of enhancing the crossbar utilization and throughput. The key idea is to reorganize the bits of the weight matrix with the architecture support to generate the sparsity that is expensive or even impossible to use in conventional architectures.

As shown in Fig. 2, the algorithm consists of three parts: quantization (a) ❶, mapping (a) ❸ and bit-flip (b). Quantization turns weights into codewords that can be deployed on crossbars and provide preliminary sparsity. Mapping maps the codeword into multiple crossbars to utilize bit-wise sparsity and simplify peripheral circuits. Finally, the bit-flip scheme transforms the original bit-level sparsity into veritable performance gain.

B. Quantization and Mapping

The quantization and mapping are illustraed in Fig. 2(a). We first quantize the weights into the sum of limited number of power-of-twos [16], as shown in equation Eq. (1). In the example ❶, the maximum number of power-of-twos is set to be three. This quantization method limits the '1' bits in the codeword and thus increases the preliminary overall bit-level sparsity while retains the accuracy well.

Because of the non-linearity of ReRAM cells, we use single-level cells to store the weights, which means that each cell only have two states: high-conductance state (1) and low-conductance state (0). Then, a quantized weight can be stored in C single-level cells w_i:

$$Q(weight) = \gamma \times \sum_{i=1}^{C} w_i 2^{-i} \tag{2}$$

Then, the inner product of input and weight array can be calculated as the sum of shifted output currents

$$\begin{aligned} \mathbf{X} \cdot \mathbf{W} &= \gamma \times \mathbf{X} \cdot \sum_{i=1}^{C} \mathbf{w}_i 2^{-i} \\ &= \gamma \times \sum_{i=1}^{C} 2^{-i} (\mathbf{X} \cdot \mathbf{w}_i) = \gamma \times \sum_{i=1}^{C} 2^{-i} I_i \end{aligned} \tag{3}$$

In Fig. 2(a) ❷, conventional weight mapping methods unfold a $H \times W$ weight matrix of C-bit quantized weights to a large $H \times (W \times C)$ bit matrix, and map into $\lceil H/h \rceil \times \lceil W \times C/w \rceil$ crossbars of size $h \times w$. This method is called intra-crossbar mapping, which maps all the bits of a weight on a single crossbar. However it ignores the difference of bit-level sparsity and significance between different digits and also leads to complicated peripheral circuits when columns are pruned, since each column should be shifted by different values before they are added together, especially when several columns are pruned.

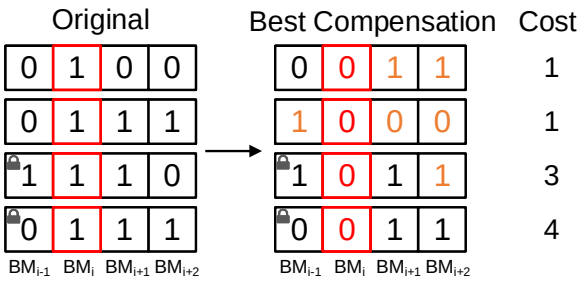

Fig. 3: The bit flip and compensation scheme. The cells with red frames are the cells to be flipped and other cells are compensate candidates. The locked cells cannot be modified and orange numbers are compensated values.

Therefore, we propose our bit-wise intra-crossbar mapping scheme ❸ that split the weight matrix bit-wisely into C bit matrices (denote as BM_i) of size $H \times W$ and each bit matrix is then divided into several crossbars. The bits of each weight are located in the same place of all bit matrices. Firstly, our method enables us to assign different pruning rates to different bit matrices, since the sparsity and significance of different digits differs greatly. The more significant digits are more sparse but their pruning rate should be smaller due to their higher significance. In addition, the output value in the same bit matrix will be shifted by a same amount when they are added up to the final result, which simplifies the peripheral circuit even when columns are pruned.

C. Bit-Flip and Compensation Scheme

Our bit-flip and compensation scheme aims at limiting the maximum number of '1's in each column to increase the number of row can be executed concurrently and prune unimportant columns to improve throughput. As shown in Fig. 2(b), the scheme is consisted of the following steps.

1) Lock the columns that are already empty in all bit matrices and execute step Item 2-4 for bit matrix 2 to C. The first bit matrix is skipped since it is usually sparse and very significant, which does not need further sparsification.
2) Calculate the cost of flipping each '1' into '0' in the given bit matrix. For each '1' bit, its one more significant digit and two less significant bits are take into consideration to compensate the value difference. The cost is determined as the smallest value difference by adjusting these cells that are has not been locked. Examples are given in Fig. 3.
3) Select the cells to flip and compensate the flips accordingly. We set two thresholds for the bit matrix: the maximum number of '1's in a column k and the number of columns left in the matrix n. Then, for the column whose number of '1' cells exceeds k, the cells that have the lowest flipping cost are flipped. Also, we calculate the total cost of all columns can flip the '1's in those columns with lowest total cost, until n columns are left. At the same time, we compensate the flips so that the minimum value bias are reached.

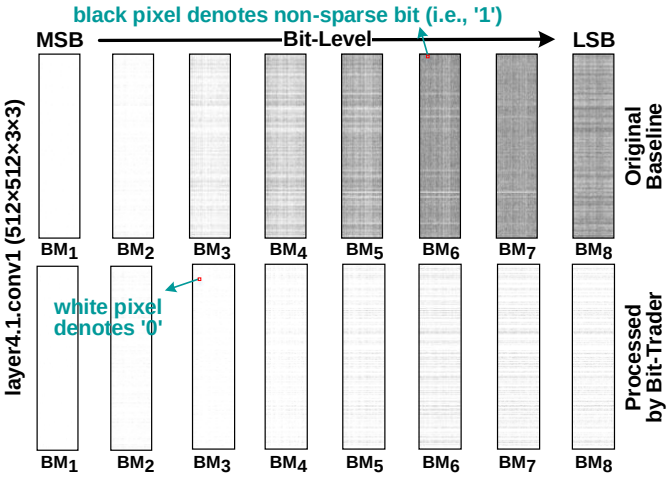

Fig. 4: The changes of bit-level sparsity in bit matrices of ResNet-18: original (up) and processed bit matrices by Bit-Transformer (down).

4) Lock the columns that have the maximum number of '1's and are newly pruned.

Thanks to our bit-wise inter-crossbar mapping scheme, the maximum number of '1's in a column and the number of columns left can be determined individually and adaptively for each bit matrix, and the pruned columns will introduce little extra overhead to the peripheral circuit.

For example, in Fig. 2 (b), for bit matrix 2, we first calculated the cost of flipping each '1' cell. After that we find that column 2 and 4 have more than two '1's, thus the cells that have the lowest column is flipped. Also, the first column has the lowest total cost and thus will be pruned. Finally, we flip the chosen '1's in bit matrix 2 and compensate their value bias, and lock the modified columns. We can see in Fig. 4 that the bit-wise sparsity is greatly increased after processing by our bit-flip and compensation scheme increases dramatically.

IV. Architecture

A. Architecture Overview

We present the overview of Bit-Transformer architecture aiming at inference in edge devices. As shown in Fig. 5, each bank consists of three parts, which are all connected to a shared bus: 1) a controller that decodes instructions

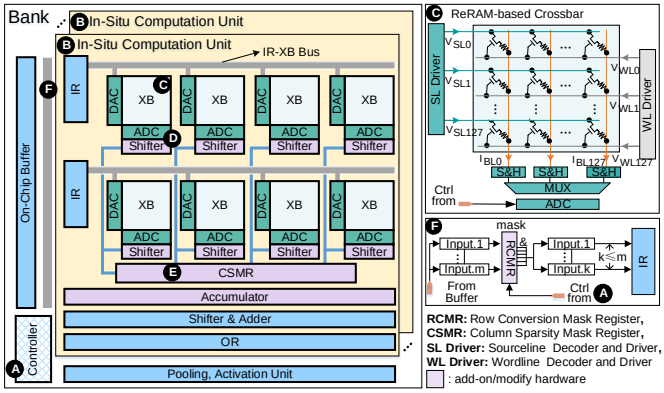

Fig. 5: Architecture Overview of Bit-Transformer.

and provides control signals to all the peripheral circuits; 2) an in-situ Computation Units (CU), which is the core computing and storage unit; 3) shared blocks that contain the sigmoid units, pooling units, and buffers for storing activations (i.e., intermediate computing results). Each CU consists of input/output registers, crossbars (XBs), ADCs, and shift-and-add units, which are drawn by blue box. These are consistent with the conventional ReRAM based architecture. Besides, the components marked with the orange line need to be added and optimized to exploit our techniques. Furthermore, to better integrate our techniques to achieve better performance, we customize the peripheral circuits.

B. Module and DataFlow

Controller. Controller in Fig. 5 Ⓐ provides control signals to all the peripheral circuits and drive the finite state machines that steer the inputs and outputs correctly after every cycle based on the technique configurations.

In-situ Computation Unit. Fig. 5 Ⓑ shows the CU which contains multiple crossbars and the peripheral circuit. The DACs receive the input digital signals from the input register and convert them into analog voltage signals as the input for ReRAM-based crossbar. After executing the MAC operation in the crossbar, the analog output current signal are then send to the ADCs to convert the analog voltage signal into digital output. Then, the shifters shift the obtained output according to the significance of the crossbar, $i.e.$, the index of bit matrix. The column sparsity mark registers (CSRM) then pad zeros to recover the pruned output into the original size and send the result to the accumulator to obtain the final result of the input cycle. Given that the input are divided into bits and are send in several cycles, the output of the accumulator is then send to the shift and adder module to get the full result. In addition, the weight matrix contains positive and negative weights, which stored in two separate crossbars.

ReRAM-based crossbar. Fig. 5 Ⓒ shows the ReRAM-based crossbar with 128×128 size, which perform parallel MAC operations. We adopt the single-level cell 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. And the peripheral circuits including the word-line driver, sample-and-hold blocks, multiplexer (MUX) and MUX controller. The output analog signals are transmitted to ADC via MUX to perform analog-to-digital conversion, which is controlled by MUX controller.

The circuit to support flexible control of ReRAM cells. Fig.5 Ⓓ indicates the circuit design for controlling ReRAM cells for flexible controlling. As shown in Fig. 6, each column and row can be switched independently. Sepcifically, SL_{Ctrl} wires are connected to the DAC and serves as the input of the crossbar; WL_{Ctrl} signals connect the control signals with the gate of the transistor to controll the connectivity of

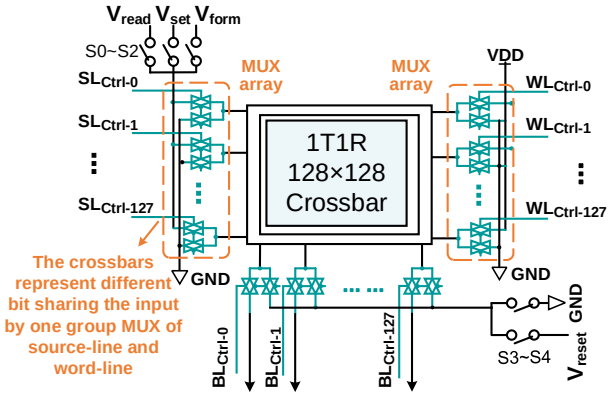

Fig. 6: The circuit design for flexible execution of MACs.

the cell; the BL_{Ctrl} signals control whether the BL current is connected to ADC. These signals enables us to perform flexible selection of the activated area of the crossbar that are executed concurrently.

The circuit to utilize bit-level row and column sparsity. Fig. 5 Ⓔ represents the circuit supports the index decoder. The cost of index is taken into account in the algorithm design, and the storage cost is further reduced via algorithm optimization. For example, in Fig. 7, the sparsity of each block is stored in the sparsity table. Each block has a size of 32*32 and '1' in the table means that the block is not pruned completely. We use the sparsity table, which is generated offline, to generate the CSM register (stored output mask). CSM register indicates whether all columns of bit matrix occupied by a block have been removed. In the given red frame of cells mapped on a single crossbar, the 1-th and 4-th cell is '0' in the CSM register, which means the columns occupied by the first and fourth blocks have no outputs.

In addition, Fig. 5 Ⓕ shows the communication between eDRAM Buffer and register. After bit-flipping, there still exists plenty of row sparsity in the pruned bit-matrix, as shown in Fig. 4. Therefore, skipping these empty rows are necessary to increase computation efficiency and exploit the performance. The RCM register is used for fetching inputs from buffer to the register and filter the unnecessary inputs

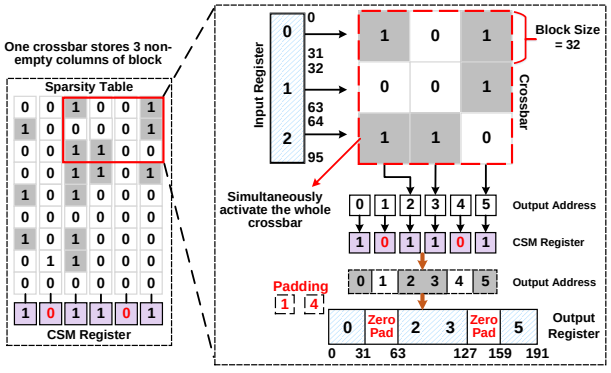

Fig. 7: Schematic diagram of index decoder. The column storage mask (CSM) register is a binary sequence of masking bits.

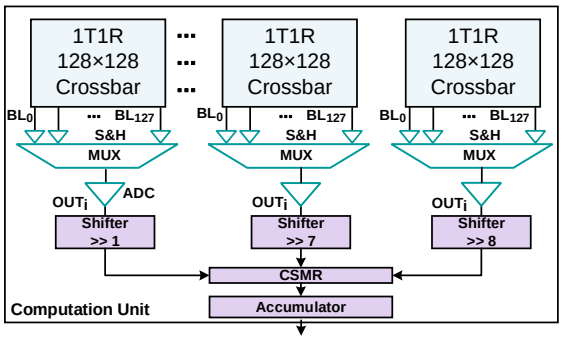

Fig. 8: The bit-wise and inter-crossbar mapping circuit.

(i.e., the corresponding row is empty). The RCM register consists of a binary sequence in which the '0' identifies the location of the empty and thus unnecessary row, whose corresponding input is not required to be entered into the crossbar for MAC operation.

The circuit to support bit-level inter-crossbar mapping. Fig. 8 illustrates the the digital circuit we designed to support the accumulation of bit-level inter-crossbar mapping results. The input voltages are first applied on each row of all the crossbars in the same in-situ CU to carry out MAC operation and get the generated current on each bitline. The currents then go through the sample-and-hold (S&H) to transfer into voltages. Then, the stable voltage signals of the S&H are passed to the ADC block to converts them into digital signals (i.e. OUT_i in Fig. 8) one-by-one through the control of the MUX-based data path. After that, the results are shifted by different value according to the index of the bit matrix. Finally, they are padded with zeros through CSM register and accumulated to get the final result of the current input cycle.

V. Experiments

A. Experimental Methodology

Experimental Setup The goal of Bit-Transformer is to improve the throughput and efficiency of the hardware accelerator. We have taken the following factors into consideration when selecting the benchmarks. They should be sufficiently representative and diverse enough to cover a range of hardware performance and overhead caused by compression effect, ranging from small scale dataset to big scale dataset. Besides, we implement our Bit-Transformer algorithm framework in the Pytorch framework to valid it.

We examine our work on classical image classfication tasks, using several representative DNNs (ResNet18, ResNet50 [2], VGG16 [26], AlexNet [27], MobileNet-v2 [28]) and two benchmark datasets (CIFAR-10 [29], ImageNet [30]). We compare the accuracy and crossbar footprint reduction with the PIM-Prune [12], SRE [11], SmartExchange [31], Pattern-based [32], and P-RM [33]. For comparison, we take the results reported in these original works. For results that are not available, we reproduce their experiments and report the results.

TABLE I: Configurations of GEM5 used in our simulation.

On-Chip Buffer (DRAM)	HBM_1000_4H_1×128 model; 16KB/Bank
Bank IO Bus	42.56MB/s
ReRAM-based Main Memory	20MB ReRAM; 100ns/cycle 8 crossbars/CU; 8 CUs/bank;

TABLE II: Comparison of NN accuracy and compression effect.

Network	Ori. Acc.	Method	Proc. Acc.§	Spar sity	XB FP	Sparsity Type	Re-RAM
CIFAR-10							
VGG-16	93.66	SmartExchange [31]	92.87	94.1%	N/A	Unstructured	✓
	93.70	PIM-Prune [12]	93.23	N/A	26.85×	Structured	✓
	93.70	P-RM [33]	93.2	98.0%	N/A	Unstructured	×
	93.70	Bit-Transformer-W	93.60	99.2%	11.34×	Structured	✓
	93.70	Bit-Transformer-X†	92.98	97.3%	32.17×	Structured	✓
ResNet-18	94.58	SmartExchange [31] ‡	94.54	91.3%	N/A	Unstructured	✓
	94.14	PIM-Prune [12]	93.84	N/A	24.85×	Structured	✓
	94.14	P-RM [33]	93.22	98.3%	N/A	Unstructured	×
	94.14	Bit-Transformer-W	93.97	99.2%	10.18×	Structured	✓
	94.14	Bit-Transformer-X	93.34	97.1%	31.98×	Structured	✓
ImageNet							
VGG-16	71.18	SmartExchange [31] ‡	70.97	87.7%	N/A	Unstructured	✓
	74.5	Pattern-Based [32]	73.6	92.3%	N/A	Structured	×
	71.59	SRE [11]	N/A	95.2%ι	N/A	Unstructured	✓
	71.59	Bit-Transformer-W	70.91	91.0%	3.77×	Structured	✓
	71.59	Bit-Transformer-X	70.87	89.4%	7.52 ×	Structured	✓
ResNet-18	69.9	Pattern-Based [32]	67.1	88.9%	N/A	Structured	×
	69.76	PIM-Prune	68.72	N/A	3.56×	Structured	✓
	69.76	Bit-Transformer-W	69.04	88.4%	3.18×	Structured	✓
	69.76	Bit-Transformer-X	68.97	86.3%	5.84 ×	Structured	✓
ResNet-50	76.13	SmartExchange [31]	75.31	70.1%	N/A	Unstructured	✓
	76.13	Pattern-Based [32]	75.6	85.3%	N/A	Structured	×
	76.13	SRE [11]	N/A	83.9%	N/A	Unstructured	✓
	76.13	Bit-Transformer-W	75.47	91.9%	3.87×	Structured	✓
	76.13	Bit-Transformer-X	75.18	89.2%	7.89 ×	Structured	✓
MobileNet-v2	72.19	SmartExchange [12]	70.16	86.79%	N/A	Structured	✓
	71.88	PIM-Prune [12]	70.11	N/A	1.91×	Structured	✓
	71.88	Bit-Transformer-W	71.23	87.17%	2.03×	Structured	✓
	71.88	Bit-Transformer-X	70.79	79.87%	3.96×	Structured	✓

§ Proc Acc. is the accuracy of the model be processed. XB FP indicates the crossbar footprint reduction
ι The sparsity ratio of SRE is based on the aggressive assumption that SRE can fully utilize the sparsity in the weights without considering the extra cost.
† The result of our Bit-Transformer-X without retraining or fine-tuning.
‡ SmartExchange uses VGG-19 and ResNet-20 network structure on the CIFAR-10 dataset, and uses VGG-11 and ResNet-50 network structure on the ImageNet dataset.

Simulation Setup We adapt the parameters similar to the analysis ISAAC with modified GEM5 [34] to build a simulator for ReRAM-based crossbar architecture, including shift-and-add module, ReRAM-based crossbar, the HyperTransport links and other parameters from ISAAC [9] and DaDianNao [35]. The configurations of DRAM, Bank and ReRAM main memory are shown in Table I. We implemented the entire digital circuit in SystemVerilog RTL, such as the controller, the multiplexer, and BL/WL Driver for the input vector. Moreover, the energy consumption and area overhead of memories, including eDRAM buffer, input/output register and registers stored mask (i.e., CSM and RCM Register), are calculated with CACTI [36] based on 32-nm CMOS process [37]. The memristors adapted SLC, which have a resistance range of $10K\Omega \sim 100K\Omega$ and crossbar size set as 128×128. For ADC and DAC, the model from [38] is used.

B. Experimental Result

Validation on DNN Accuracy In Table II, "FP" and "ReRAM" indicate the crossbar footprint reduction ratio and whether the sparsification method is based on the ReRAM structure, respectively. "FP" denotes the amount of crossbar saved to further saving energy and area, which is a key and intuitive indicator. We can observe that our Bit-Transformer can achieve nearly $8\times$ and $3\times$ crossbar footprint reduction

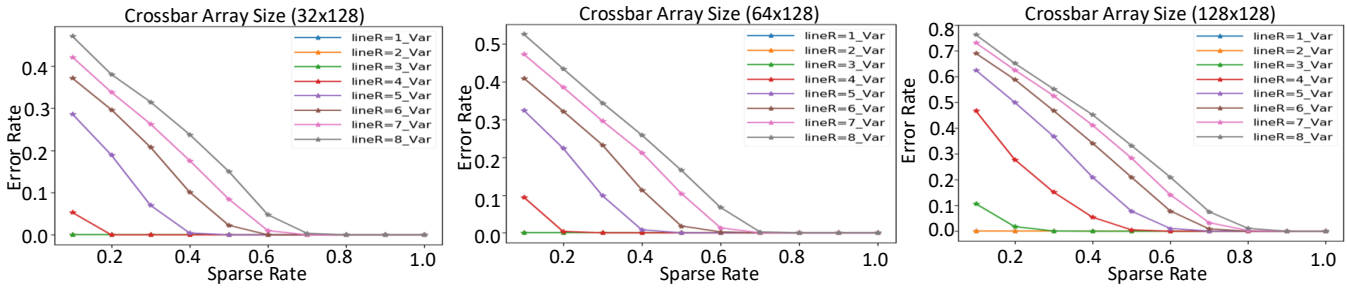

Fig. 9: Crossbar computing error rate versus crossbar sparse rate, using varying crossbar dimension.

with negligible accuracy loss compared to PIM-Prune on CIFAR-10 and ImageNet. For the more compact MobileNet-v2, the parameter size is much smaller, we also can reduce the crossbar footprint by $10\times$ on CIFAR-10. However, many solutions do not convert the reduced parameters into the reduction of the crossbar, we use the sparse rate as the metric. Specifically, for ResNet-50, Bit-Transformer achieve 91.9% sparse rate with 0.6 accuracy loss compared to SmartExchange (70.1% sparse rate with 0.8 loss) and SRE (83.9% sparse rate with unreported loss) on ImageNet.

Impact on the concurrent execution of MACs. Fig. 9 first shows the relationship between sparsity and error rate under the different number of wordlines and bitlines for concurrent execution of MACs in the crossbar. We can find the minimum sparsity requirements for concurrent execution of MACs of crossbar. We set the initial input voltages V_{read} as $1.2V$, LRS = $10K\Omega$, HRS = $100K\Omega$, and vary the sparsity ratio from 0% to 100%. We set wire resistance from 1Ω to 8Ω for simulating the IR drop. The error rate (y-axis) is determined on the output current of the sourceline. We observe that crossbar size = 32×128, crossbar size = 64×128 and crossbar size = 128×128 have the zero error under the 50% sparsity with the wire resistance as $1 \sim 5\Omega$. The error rate drop is attributed to the enhancement of sparsity accomplished by Bit-Transformer that enables more LRS (low-resistance state) ReRAM cell on both row or column of a crossbar,

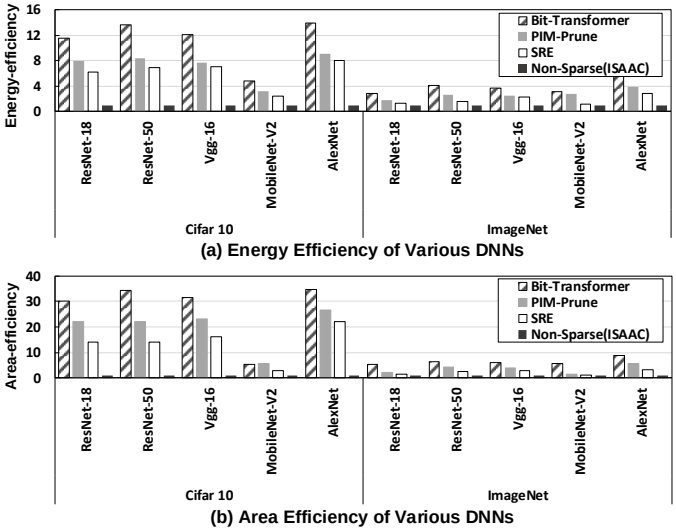

Fig. 10: Area- and energy-efficiency of between Bit-Transformer and baseline designs for various networks

which not only increases the information density and the overall throughput, but also decreases the dynamic power consumption. We take the minimum sparsity requirement of each column as the optimization objective of our bit-flip strategy (refer to Section Section III-C).

Besides, We made strict restrictions on the error rate. We concurrently execute the MACs of the whole array size and compare the current of each cell I_{cell} with reference current $I_{\text{ref}} = V_{\text{read}}/55K\Omega$. If the cell's state is HRS (stores '0') and $I_{\text{cell}} < I_{\text{ref}}$, output current generated by the cell is correct, and vice versa. And then use the number of cell that produces the error current by dividing the array size to get the error rate.

Energy, Area Consumption and Performance. Fig. 10 reports the area- and energy-efficiency for the five networks with 128×128 crossbar on the two datasets. We normalize the area- and energy-efficiency to that of the model without any sparsification, named Non-Sparse. Specifically, our bit-flip strategy improves the area-efficiency for more than $31\times$ on ResNet, VGG16, and AlexNet on CIFAR-10, compared to $20\times$ for PIM-Prune and $14\times$ for SRE (OU size is 4×4). The energy-efficiency has improved $11\times$ on ResNet, VGG16, and AlexNet for CIFAR-10, and the effect was much better than that of PIM-Prune and SRE. The energy and area reduction against PIM-Prune and SRE is mainly due to the reduced the crossbars overhead and the resolution of ADCs. For ImageNet, Our method is still superior to the existing methods, but the energy- and area-efficiency are limited (i.e., $3\times$ for energy-efficiency and $5\times$ for area-efficiency on average). To ensure the accuracy within the acceptable range, the model is difficult to further compress for large-scale datasets. While for the more compact MobileNet-v2 which has a much smaller parameter size, flipping bits makes our Bit-Transformer more effective for improving the crossbar utilization without accuracy loss.

Fig. 11 shows the speed up over the baseline accelerator (i.e., ISAAC), we compare the cycles of inference on various networks. For a fair comparison, we choose the solutions (SRE [11], PIM-Prune [12]) are based on ISAAC. We can see Bit-Transformer achieves the best performance under all the three DNN models on ImageNet. Taking the ResNet-50 as an example, compared to SRE, PIM-Prune and ISAAC, Bit-Transformer achieves performance improvement ranging from $31.3\times$ to $121.7\times$. This experiment verifies the effectiveness of the algorithm-hardware co-design, Bit-Transformer, and reduces the computation and improve the throughput. Since Bit-Transformer takes full advantage of bit-level sparsity of

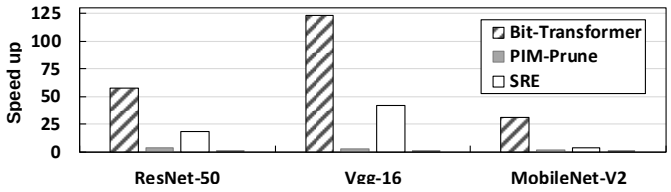

Fig. 11: Comparison of normalized speed up for various networks.

the weights, it not only reduces the crossbar footprint for deploying the model but also improves throughput by sparsity on the crossbar. Hence, it has a higher speedup than the baseline. Also, we map each bit of the weight to different crossbars. After we perform the bit-flip strategy, we realized an almost unstructured bit-level pruning without introducing much extra peripheral circuits, making the acceleration effect of quantization enhenced by such bit-level sparsity.

Indexing Overhead Analysis It is worth noting that the output mask we stored is much cheaper than the index requirement of SRE on the column. Besides, all masks we need are based on block, the size of the block determines the further reduces the overhead of masks. Fig.12 shows the storage overhead of different networks varies according to the model size and block size. Bit-Transformer achieves average of 75% and 53% output register overhead reduction compared to PIM-Prune and SRE with $height \times 4$ block size; achieves average of 96.8% and 94.1% output register overhead reduction compared to PIM-Prune and SRE with 32×32 block size and achieves average of 93.7% and 88.2% output register overhead reduction compared to PIM-Prune and SRE with 16×16 block size. The larger the network scale (e.g., ResNet-50 with massive convolution layers), the greater the storage overhead. However, we benefit from our algorithm framework, the padded output is continuous by our Bit-Transformer, and the storage overhead will be further reduced according to the block size. The block size $A \times B$ means the consecutive A rows shares a mask (1-bit), and consecutive B columns shares a mask (1-bit). Specifically, for ResNet-50, compared to SRE (required 778KB input index and 48KB output index), our Bit-Transformer consumes nearly 100% and 95.6% less storage of input index and output index with same block size (16×16) reduces. Because Bit-Transformer performs bit-wise inter-crossbar mapping, the processed model is deployed on the crossbar, and a group of crossbars represents the same bit of different weights. To save the overhead of input MUX, crossbars in the same group (i.e., store the same bit) share the input. In this way, we focus on reducing the columns. And

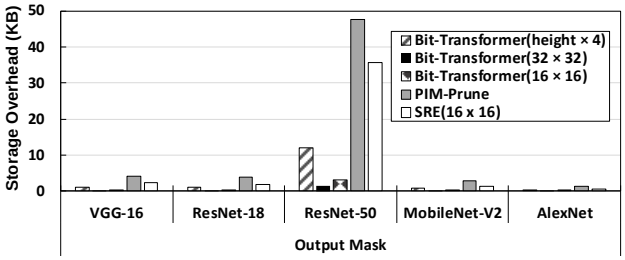

Fig. 12: Comparison of output register overhead.

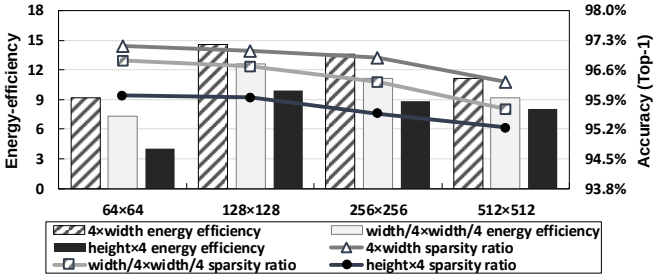

Fig. 13: Analyzation of the crossbar size and block size

while there is some sparsity on the rows, it can also be stored on the crossbar to avoid the additional hardware cost of inputs.

C. Design Space Exploration

In this section, we explore the influence of crossbar size and the block size on our Bit-Transformer.

Impact of crossbar size and block size. Fig. 13 shows the influence of different crossbar array on sparsification effect and overhead on ResNet-18 with the accuracy loss $<1\%$. We change the size of the crossbar array from 64 to 256. Our block settings are related to the crossbar size so that that block will change as the size of crossbar changes. For the weight matrix of ResNet-18, the broader block size is better, because the weight matrix of ResNet-18 is slender, but our method needs to split and map to the different crossbars according to bits, so we have more tolerance on columns. However, for MobileNet-v2, we can achieve great performance because its convolution layers' shape is to match our techniques.

Besides, conventional sparsification methods pursue that the smaller the granularity, the better the sparsification effect, while the hardware design desire that larger the crossbar size, the greater density and energy-efficiency. We can find that the sparsification effect is the highest when the crossbar size is 64, but the energy efficiency is the lowest. Therefore, to obtain the optimal performance, the influence of software and hardware should be considered in the design process.

VI. Conclusion

ReRAM has shown great prospects in neural network accelerator design. However, the non-ideal effects of devices and circuits leading to the limited throughput. In this paper, we propose Bit-Transformer, an algorithm-hardware co-design framework to trade higher bit-level sparsity in weights for lower-cost and higher-throughout computation, for boosting the energy- and area-efficiency of DNN inference on ReRAM. Besides, we design the architecture to efficiently support the sparsity generated by the proposed algorithm through the well-designed crossbars with the peripheral circuit. Our evaluation shows that the proposed Bit-Transformer outperforms other similar solutions in performance, energy, and accuracy.

Acknowledgment

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

REFERENCES

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[3] 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.

[4] 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.

[5] M. Hu, J. P. Strachan, Z. Li, E. M. Grafals, N. Davila, C. Graves, S. Lam, N. Ge, J. J. Yang, and R. S. Williams, "Dot-product engine for neuromorphic computing: Programming 1t1m crossbar to accelerate matrix-vector multiplication," in *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE, 2016, pp. 1–6.

[6] 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.

[7] Y. Ji, Y. Zhang, X. Xie, S. Li, P. Wang, X. Hu, Y. Zhang, and Y. Xie, "Fpsa: A full system stack solution for reconfigurable reram-based nn accelerator architecture," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019, pp. 733–747.

[8] Z. He, J. Lin, R. Ewetz, J.-S. Yuan, and D. Fan, "Noise injection adaption: End-to-end reram crossbar non-ideal effect adaption for neural network mapping," in *Proceedings of the 56th Annual Design Automation Conference 2019*, 2019, pp. 1–6.

[9] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. P. Strachan, M. Hu, R. S. Williams, and V. Srikumar, "Isaac: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 14–26, 2016.

[10] L. Song, X. Qian, H. Li, and Y. Chen, "Pipelayer: A pipelined reram-based accelerator for deep learning," in *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2017, pp. 541–552.

[11] 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, pp. 236–249.

[12] C. Chu, Y. Wang, Y. Zhao, X. Ma, S. Ye, Y. Hong, X. Liang, H. Yinhe, and J. Li, "Pim-prune: Fine-grain dcnn pruning for crossbar-based process-in-memory architecture," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2020, pp. 1–6.

[13] A. Ankit, T. Ibrayev, A. Sengupta, and K. Roy, "Trannsformer: Clustered pruning on crossbar-based architectures for energy efficient neural networks," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2019.

[14] C. Leng, Z. Dou, H. Li, S. Zhu, and R. Jin, "Extremely low bit neural network: Squeeze the last bit out with admm," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[15] S. K. Esser, J. L. McKinstry, D. Bablani, R. Appuswamy, and D. S. Modha, "Learned step size quantization," in *8th International Conference on Learning Representations (ICLR)*, 2020.

[16] Y. Li, X. Dong, and W. Wang, "Additive powers-of-two quantization: An efficient non-uniform discretization for neural networks," in *International Conference on Learning Representations*, 2019.

[17] L. Liang, L. Deng, Y. Zeng, X. Hu, Y. Ji, X. Ma, G. Li, and Y. Xie, "Crossbar-aware neural network pruning," *IEEE Access*, vol. 6, pp. 58 324–58 337, 2018.

[18] 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.

[19] A. Nag, R. Balasubramonian, V. Srikumar, R. Walker, A. Shafiee, J. P. Strachan, and N. Muralimanohar, "Newton: Gravitating towards the physical limits of crossbar acceleration," *IEEE Micro*, vol. 38, no. 5, pp. 41–49, 2018.

[20] A. Ankit, I. E. Hajj, S. R. Chalamalasetti, G. Ndu, M. Foltin, R. S. Williams, P. Faraboschi, W.-m. W. Hwu, J. P. Strachan, K. Roy *et al.*, "Puma: A programmable ultra-efficient memristor-based accelerator for machine learning inference," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019, pp. 715–731.

[21] W. Li, P. Xu, Y. Zhao, H. Li, Y. Xie, and Y. Lin, "Timely: Pushing data movements and interfaces in pim accelerators towards local and in time domain," *arXiv preprint arXiv:2005.01206*, 2020.

[22] 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.

[23] F. Liu, W. Zhao, Z. Wang, T. Yang, and L. Jiang, "Im3a: Boosting deep neural network efficiency via in-memory addressing-assisted acceleration," in *Proceedings of the 2021 on Great Lakes Symposium on VLSI*, 2021, pp. 253–258.

[24] B. Liu, H. Li, Y. Chen, X. Li, T. Huang, Q. Wu, and M. Barnell, "Reduction and ir-drop compensations techniques for reliable neuromorphic computing systems," in *2014 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2015, pp. 63–70.

[25] W.-H. Chen, K.-X. Li, W.-Y. Lin, K.-H. Hsu, P.-Y. Li, C.-H. Yang, C.-X. Xue, E.-Y. Yang, Y.-K. Chen, Y.-S. Chang *et al.*, "A 65nm 1mb nonvolatile computing-in-memory reram macro with sub-16ns multiply-and-accumulate for binary dnn ai edge processors," in *2018 IEEE International Solid-State Circuits Conference-(ISSCC)*. IEEE, 2018, pp. 494–496.

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

[27] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[28] 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.

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

[30] 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.

[31] 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," *arXiv preprint arXiv:2005.03403*, 2020.

[32] W. Niu, X. Ma, S. Lin, S. Wang, X. Qian, X. Lin, Y. Wang, and B. Ren, "Patdnn: Achieving real-time dnn execution on mobile devices with pattern-based weight pruning," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2020, pp. 907–922.

[33] X. Ma, G. Yuan, S. Lin, Z. Li, H. Sun, and Y. Wang, "Resnet can be pruned 60×: Introducing network purification and unused path removal (p-rm) after weight pruning," in *2019 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*. IEEE, 2019, pp. 1–2.

[34] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The gem5 simulator," *SIGARCH Comput. Archit. News*, vol. 39, no. 2, p. 1–7, Aug. 2011. [Online]. Available: https://doi.org/10.1145/2024716.2024718

[35] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun *et al.*, "Dadiannao: A machine-learning supercomputer," in *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE, 2014, pp. 609–622.

[36] N. Muralimanohar, R. Balasubramonian, and N. Jouppi, "Optimizing nuca organizations and wiring alternatives for large caches with cacti 6.0," in *40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2007)*. IEEE, 2007, pp. 3–14.

[37] Synopsys, [Online], https://www.synopsys.com/community/university-program/teaching-resources.html.

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