XMA: A Crossbar-aware Multi-task Adaption Framework via Shift-based Mask Learning Method

Fan Zhang, Li Yang, Jian Meng, Jae-sun Seo, Yu (Kevin) Cao, Deliang Fan
School of Electrical, Computer and Energy Engineering, Arizona State University
{fzhang95,lyang166,jmeng15,jseo28,ycao17,dfan}@asu.edu

ABSTRACT

ReRAM crossbar array as a high-parallel fast and energy-efficient structure attracts much attention, especially on the acceleration of Deep Neural Network (DNN) inference on one specific task. However, due to the high energy consumption of weight re-programming and the ReRAM cells' low endurance problem, adapting the crossbar array for multiple tasks has not been well explored. In this paper, we propose *XMA*, a novel crossbar-aware shift-based mask learning method for multiple task adaption in the ReRAM crossbar DNN accelerator for the first time. XMA leverages the popular mask-based learning algorithm's benefit to mitigate catastrophic forgetting and learn a task-specific, crossbar column-wise, and shift-based multi-level mask, rather than the most commonly used element-wise binary mask, for each new task based on a frozen backbone model. With our crossbar-aware design innovation, the required masking operation to adapt for a new task could be implemented in an existing crossbar-based convolution engine with minimal hardware/memory overhead and, more importantly, no need for power-hungry cell re-programming, unlike prior works. The extensive experimental results show that, compared with state-of-the-art multiple task adaption Piggyback method [1], XMA achieves 3.19% higher accuracy on average, while saving 96.6% memory overhead. Moreover, by eliminating cell re-programming, XMA achieves ~4.3× higher energy efficiency than Piggyback.

KEYWORDS

In-Memory Computing, Neural Networks, Multi-task Learning

1 INTRODUCTION

Although deep neural networks (DNNs) show superior performance in many applications, the high degree of specialization to a single task limits its potential development. Inspired by this, researchers started developing algorithms that could sequentially adapt the DNN model to multiple tasks while still performing well on past tasks. This process of gradually adapting the DNN model to learn from various tasks is known as *multi-task adaption* [1, 2]. Fine-tuning [3] is an intuitive way to adopt the knowledge from the current model (i.e., backbone model) to a new task. Although it shows good accuracy on new learnt task, updating the weights of the backbone model could result in the forgetting of old knowledge upon earlier tasks, thus greatly degrading the performance. Such phenomenon is known as *catastrophic forgetting*, which widely exists in multi-task adaption.

On the hardware side, DNNs involve a vast number of multiply and accumulate (MAC) operations and data movement. In traditional von Neumann architecture (e.g., CPU, GPU), such a large amount of data movement may consume ~100× higher energy than a floating-point operation which is also known as "memory wall" [4]. Recently, in-memory computing (IMC) has attracted growing attention due to its capability to compute MAC directly within the memory array. Such ability significantly relieves the "memory wall" issue [5, 6]. Among different volatile/non-volatile IMC designs, ReRAM crossbar-based design is a promising candidate for the next generation DNN accelerator for deploying inference due to its simple structure, high on/off ratio, high density, multi-bit per cell storage, and fabrication compatibility with CMOS [4, 7].

Although many ReRAM crossbar-based designs [4, 5] have been proposed to support DNN inference as area and energy efficient computing engines, they are mostly designed for deploying on **a single specialized task, which cannot be adapted to different tasks (i.e., *multi-task adaption*)**. Such limitation greatly impedes its practical usability in the real-world. Intuitively, to adapt a specialized DNN model deployed in the ReRAM crossbar for a new task, a common practice is to fine-tune the weight parameters (i.e., cell conductance) based on the new knowledge. However, this scheme will require updating the conductance of almost all cells to reflect the new set of fine-tuned weight parameters. This is inefficient and impractical in real-world multi-task learning due to limitations in both the ReRAM device (e.g., high re-programming energy, limited endurance, etc.) and algorithm (e.g., catastrophic forgetting for large scale multi-task learning). Accordingly, differentiating from the traditional one-time mapping for inference on a single specialized task, ReRAM crossbar-based multi-task adaption requires software and hardware co-design to solve these issues.

Recently, mask-based learning [1, 2] method has been proposed to perform multi-task adaption. Piggyback [1], as a representative work, learns a task-specific binary mask $\in \{0, 1\}$ w.r.t all weights in an element-wise manner for each new task while freezing the backbone model. This method is a possible candidate for ReRAM crossbar-based accelerator design as it has no re-programming issue for the weights parameters. However, there are still two drawbacks from software and hardware perspectives, which impedes its practical usability: 1) the adaption capacity of Piggyback is limited, as it only removes the irrelevant weights by applying a binary mask for each new task, without involving new task-specific knowledge. This is why Piggyback cannot reach similar accuracy as the fine-tuning method; 2) the element-wise pattern of the learnable mask is not hardware-friendly, mainly due to the following two

DAC '22, July 10–14, 2022, San Francisco, CA, USA

ACM ISBN 978-1-4503-9142-9/22/07…$15.00
https://doi.org/10.1145/3489517.3530458

reasons: first, applying element-wise mask on crossbar against the crossbar's intrinsic property that crossbar works in the row/column-wise parallel fashion. Although it is easy to control/read the entire row/column, manipulating each cell individually is impractical; second, the element-wise pattern requires a considerable memory overhead for the new learnt mask. For example, Piggyback's element-wise mask will cause memory overhead of 1/8 of the total model size in an 8-bit DNN model for each new task.

To tackle these crucial and practical issues, we are the first to propose a novel crossbar-friendly multi-task adaption framework, called XMA, which learns a *crossbar column-wise* and *shift-based* mask while freezing the backbone model, taking great advantages on both software algorithm for offline training and hardware deployment for online inference. Such novel mask design is distinguished from prior works in the following two aspects:

IMC-friendly 'Shift-based Mask'. To improve the adaption capacity with negligible hardware overhead, rather than element-wise binary or soft mask, we propose to learn a crossbar column-wise shift-based mask, where all the mask values are zeros or power-of-two $\in [0, 1]$ (i.e., 0,1/2,1/4,1/8,1). By doing so, the compute/memory-hungry multiplication operation between mask and column-wise MAC output can be simplified as a shift operation. Moreover, such shift operation can be implemented by reusing the existing shift adder in most ReRAM based IMC platforms without increasing hardware overhead. The experiment shows that the proposed shift-based mask learning method reaches 2.26% higher accuracy on average than the fine-tuning method and 3.19% higher accuracy than the element-wise binary mask Piggyback [1]. In addition, selecting the number of shift levels 'N' in shift-based mask is flexible that could be adjusted to achieve different trade-offs between accuracy and mask overhead. For example, if $N = 3$, it supports maximally 3 different shift levels and 2 non-shift levels, i.e., 0, 1/8, 1/4, 1/2 and 1, with the best accuracy. Note, mask value '1' means no shift, and mask value '0' means turning off current column. If $N = 0$, the shift-based mask is equivalent to the binary mask with the smallest mask memory overhead.

IMC-friendly crossbar 'column-wise mask' pattern. To reduce the peripheral circuit overhead for implementing the masking function in hardware and avoid power-hungry re-programming of ReRAM cells for multi-task adaption, we design the shift-based mask in crossbar column-wise pattern, where each learned mask value controls the operations (i.e., on/off, shift) of entire crossbar column for the new task inference, instead of each element in Piggyback, which saves 96.6% memory overhead in our experiments.

In summary, considering software-hardware co-design, XMA is simple but efficient and powerful. It improves the adaption capacity of the existing binary mask-based algorithm in multi-task learning. Also, it could be easily implemented in existing crossbar-based DNN accelerator hardware with minimum peripheral circuits, mask memory overhead, and, more importantly, there is no need to re-program ReRAM cell conductance for learning a new task.

2 RELATED WORK

2.1 ReRAM based NN accelerator

As a new technology that has attracted much attention, many ReRAM-based accelerator designs have been proposed, especially for DNN inference [4, 5]. Fig. 1 shows the basic structure of the 1T1R crossbar array, which can efficiently perform the vector-matrix multiplication (VMM) operation. The 1T1R cell is a passive three-terminal device where the ReRAM cell applies the metal-insulator-metal sandwich-like structure. The middle insulator layer is usually made by HfO_2 [8], TiO_x [9], etc. Those provide the ReRAM the resistive switching characteristics when applying different voltages.

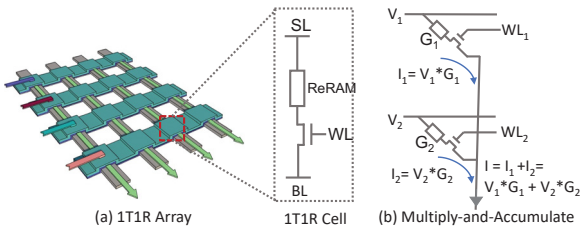

Figure 1: ReRAM 1T1R crossbar array.

When using a 1T1R array for the VMM operation, the weight matrix is stored at the cross-point ReRAM cells as the conductance G while the input vector is fed through the horizontal SL as analog voltage V_{in} [7, 10]. According to Kirchhoff's Current Law (KCL), $I_{BL} = G \cdot V_{in}$, the bit-line current is the VMM result. A $m \times n$ sized crossbar array can perform the VMM operation in one step, reducing the time complexity from O(mn) to O(1). Due to the 2D convolution can be transferred to VMM either by Toeplitz matrix or unrolling the convolution kernel. Recently a lot of ReRAM crossbar array-based neural network accelerator designs have been proposed to leverage the IMC's energy efficiency and high throughput [4, 5].

Existing ReRAM crossbar designs focus on improving energy efficiency for the off-line trained fixed model. If the dataset or task changes, re-programming the entire crossbar array is necessary. Although Fouda et al. [11] proposed a mask-based method for crossbar array, the mask is used only during off-line training. Moreover, this method is employed to alleviate the sneak path problem, not for multi-task adaption. Different from the above works, we explore the design space of the ReRAM crossbar for multi-task adaption and propose the crossbar-aware mask learning method to reuse the crossbar for new tasks efficiently.

2.2 Multi-Task Adaption

Multi-task adaption [12, 13] aims to build a model, which can adapt a task into multiple visual tasks/domains without forgetting previous knowledge, and meanwhile using as few parameters as possible. [13] proposes to recombine the weights of the backbone model via controller modules in channel-wise. [14] proposes domain-specific attention modules for the backbone model. One of the most relevant works is Piggyback [1], which solves the issue by learning task-specific binary masks for each task while freezing the backbone model except final classifier (known as *multi-head*). They achieve this by generating the real-value masks which own the same size with weights, passing through a binarization function to obtain binary masks, which are then applied to existing weights. We denote the real-value mask and binary mask as m^r and m^b respectively. Then, the binarization function is given by:

$$\text{Forward}: \quad \boldsymbol{m}^{\mathrm{b}} = \begin{cases} 1 & \text{if } \boldsymbol{m}^{\mathrm{r}} \geq \tau \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

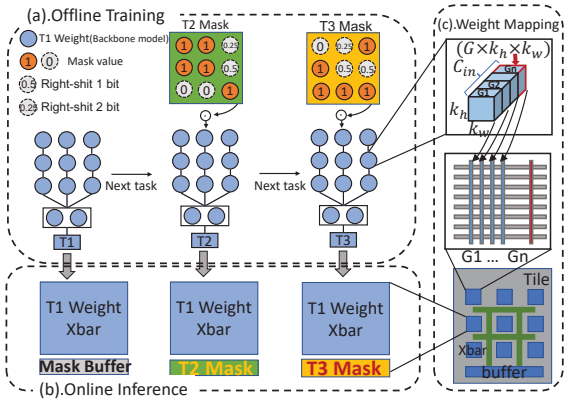

Figure 2: Overview of the crossbar-aware multi-task adaption framework, including offline shift-based masking learning (a) , the ReRAM crossbar deployment for online inference (b), and the weight mapping (c).

$$\text{Backward}: \quad \nabla \boldsymbol{m}^{\mathrm{b}} = \nabla \boldsymbol{m}^{\mathrm{r}} \tag{2}$$

Where τ is a constant threshold value. However, the gradient of binarization is non-differential during back-propagation. They use the straight-through estimator (STE) [15] to solve this problem, which estimates the gradient of real-value mask by the gradient of binary mask. Furthermore, [2, 16] combine the binary mask with additional floating-point scaling values to improve the adaption capacity, but suffer from even more computation and memory cost during the training procedure.

2.3 Neural Network Quantization

Neural network quantization has been widely studied to compress the data precision (e.g., weight, activation) while maintaining accuracy [17–19]. The stringent resource constraint of the hardware accelerator necessitates efficient quantization algorithms. Early research works [17] demonstrate the feasibility of discretizing the full precision weights between the fixed boundaries $[-1, 1]$. However, the deterministic quantization range failed to adaptively fit the layer-wise distributions, leading to the sub-optimal model performance. To minimize quantization error during training, various studies introduce layer-wise learnable clipping parameters. Under this context, PACT dynamically clips the activation based on the trainable quantization boundary. However, PACT [18] only utilizes the gradient inside the truncation range, leading to insufficient learning. To avoid this issue, we adopt the quantization algorithm from PROFIT [19] to train the DNN model.

3 METHODOLOGY

3.1 Overview

The overall framework of the proposed XMA is presented in Fig. 2, including the overflow of both offline training (Fig. 2(a)) and ReRAM crossbar array deployment for online inference (Fig. 2(b)). During the offline training, to adapt the preserved model for previous tasks to current task, XMA learns a task-specific and crossbar column-wise shift mask that is applied on the fixed weights to perform task adaption. Then for the online inference after the offline training of each task, only the mask buffer needs to be updated with the new values, with no need to re-program the preserved backbone model,

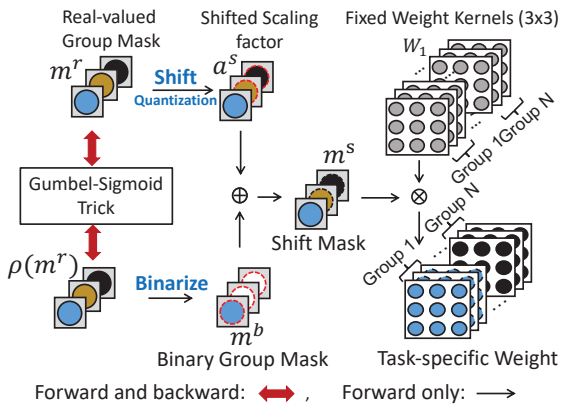

Figure 3: Overflow of the shift-based mask learning scheme

as shown in Fig. 2(b). Furthermore, XMA designs the learnable shift-based mask in crossbar column-wise, where each value controls the operation of the entire crossbar column, enabling hardware friendly crossbar mapping. The detailed shift-based mask offline learning and online inference procedure will be presented in the following subsections.

3.2 Offline Shift-based Mask Learning

3.2.1 The overflow of the shift-based mask learning. As in Piggyback method [1], the adopted binary mask is generated by binarizing trainable real-valued masks m^r, as presented in Eq. (1). We conjecture that the magnitude of these masks have the in-nature property to represent the importance levels w.r.t the corresponding weights of the backbone model. Inspired by that, we aim to utilize the real-valued mask to improve the adaption capacity. However, multiplying a real value (i.e., 32 bits floating number) for every partial sum is a massive overhead in both latency and energy. To address this issue, as illustrated in Fig. 3, we propose a *learnable shift-based mask* m^s that keeps the '1' in the binary mask, but introduces an additional *shift factors* a^s as a replacement of the zero elements in the binary mask counterpart, so as to improve the adaption capacity with negligible additional overhead. The shift-based mask can be expressed as:

$$m^s = [m^b_{m^b=1}, a^s_{m^b=0}] \tag{3}$$

Where $m^b_{m^b=1}$ means the shift-based mask only uses the '1' in the binary mask and $a^s_{m^b=0}$ denotes that the '0' in the binary mask is replaced by the proposed shift factor. It can be understood as we fix the important kernels ('1' in binary mask) and scale the unimportant kernels ('0' in binary mask) to be different shift levels for the new task.

Learn the shift factor a^s. In order to learn the shift factor, we apply a *shift equivalent quantization* method. In practice, we first normalize the real-valued mask under the range $[0, 1]$, serving as a scaling factor to represent the weight importance for multi-task adaption. Then, the normalized real-valued mask is quantized to the nearest power-of-two values (i.e., 1/2, 1/4, 1/8) or zero. Based on this, the shift-based mask m^s could maximally include 3 different shift levels (i.e., 1/8, 1/4, 1/2) and 2 non-shift level (i.e., 0, 1). By doing so, the computing/memory-hungry multiplication operation

between real-valued mask and fixed weight can be replaced by the shift operation, resulting in computing and energy reduction. Moreover, such shift operation can be implemented by reusing the existing shift adder in most ReRAM based IMC platforms without increasing hardware overhead. In addition, selecting the number of shift levels 'N' in shift-based mask is flexible that could be adjusted to achieve different trade-offs between accuracy and mask overhead. For example, if $N = 3$, it supports maximally 3 different shift levels and 2 non-shift levels, i.e., 0, 1/8, 1/4, 1/2 and 1, achieving the best accuracy. Note, mask value '1' means no shift, and mask value '0' means turning off current column. If $N = 0$, the shift-based mask is equivalent to the binary mask with the smallest mask memory overhead.

Learn the binary mask m^b. To learn the binary mask, we leverage the Gumbel-Sigmoid trick, inspired by Gumbel-Softmax [20] that performs a differential sampling to approximate a categorical random variable. Since Sigmoid function $\sigma(\dot{})$ can be viewed as a special two-class case of softmax, it can be defined as:

$$p(\boldsymbol{m}^r) = \frac{1}{1 + \exp(-(\log\pi_0 + g_0 - g_1)/T)}, \tag{4}$$

where π_0 represents $\sigma(\boldsymbol{m}^r)$. g_0 and g_1 are samples from Gumbel distribution. The temperature T is a hyper-parameter to adjust the range of input values. Benefiting from the differential property of Eq. (4), the real-value mask $\boldsymbol{m}^r$ can be embedded with existing gradient based back-propagation training. To represent $p(\boldsymbol{m}^r)$ as binary format $\boldsymbol{m}^b$, we use a hard threshold (i.e., 0.5) during forward-propagation of training. Because most values in the distribution of $p(\boldsymbol{m}^r)$ will move towards either 0 or 1 during training, generating the binary mask by $p(\boldsymbol{m}^r)$ (instead of the real-value mask $\boldsymbol{m}^r$ directly) could have more accurate decision, resulting in better accuracy.

3.2.2 Column-wise mask. In the 1T1R crossbar, the entire row and column shares the same input, and the transistors' gates are connected together either horizontally or vertically. It is challenging to apply an element-wise mask by controlling every weight stored at the crossbar intersections. However, such row/column-wise parallelism provides the opportunity of the row/col wise controlling for the existing crossbar design. In the conventional convolution kernel mapping method, the kernel is divided by output feature map dimension. For example, a $C_{out} \times C_{in} \times kh \times hw$ kernel will be reshaped to a $(C_{in} \times kh \times kw, C_{out})$ sized 2D matrix. With the development of deep learning in recent years, DNNs grow into more complex and more extensive structures, the size of one filter $C_{in} \times kh \times kw$ usually is too large to fit into a single crossbar column. A general solution is to further partition and then deploy one filter into multiple columns.

To leverage the row/column-wise parallelism, we define the mask size as $G \times kh \times kw$ to make it consistent with the size of a crossbar column, namely column-wise mask, where the group $G \in \{1, C_{\text{in}}\}$. So that, a single mask value can control the entire column of a crossbar array, which improves the computation efficiency significantly compared to the element-wise mask. In our design, the size of the crossbar column is set as 72×1. Equivalently, we define the group size of the kernel-wise mask as $8 \times 3 \times 3$ with the group $G = 8$ in the algorithm.

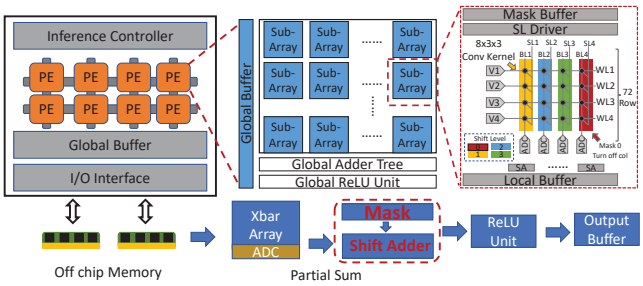

Figure 4: Hardware structure

3.3 Online Crossbar Array Inference

3.3.1 Hardware Architecture. Fig. 4 shows the ReRAM crossbar-based NN accelerator design to support inference with the proposed column-wise mask. It consists of an I/O interface for data exchange, multiple processing element (PE) for computing, and the interface controller to decode the instruction. Each PE has multiple ReRAM crossbar sub-arrays for the convolution operation; global ReLU and adder tree are used to post-process the partial sum from the sub-arrays. Inside the ReRAM sub-array, convolution kernels are mapped on ReRAM cells. According to the ReRAM device property, it may need multiple ReRAM cells to represent one convolution kernel. For example, the convolution kernel is quantized to 4-bit where each ReRAM cell can only represent a 2-bit number by four different resistance levels. Then, each convolution kernel requires two adjacent columns to map its higher bits and lower bits. Since the weight has been divided into two columns, each column only carries a partial accumulation which ADC reads as 2-bit activation. The shift-adder (SA) manipulates the partial activation to obtain the 4-bit activation result. The mask buffer stores the column-wise mask and controls how to shift the activation. The processed activation is then sent to the global adder tree and ReLU, which subsequently is conveyed to the next layer as the input.

3.3.2 Weight Mapping. Recently many weight mapping methods have been proposed for better data reuse and ReRAM array utilization [21–24]. We adopt the mapping method from [21] which is widely used and works properly with our proposed method. We assume the 4D kernel is organized as $C_{out} \times C_{in} \times kh \times hw$ dimensions. When performing the convolution, kernels that belong to the same layer but on different C_{out} dimensions share the same input. This is similar to the crossbar, where the ReRAM cells on the same row but different columns share the same input voltage. Therefore, we unroll the convolution kernel along the C_{out} dimension and map them to the ReRAM crossbar sub-array. In our design, we use the 72×72 crossbar array to make the group size the same as mentioned earlier. Also, we chose the 2-bit ReRAM cell and 4-bit quantized model in our design. Since we need two columns to represent one 4-bit weight, each crossbar array can map a 36×8×3×3 convolution kernel. Any convolution kernel larger than 36×8×3×3 will be partitioned into multiple arrays. In that case, each array will generate a partial sum instead of the activation.

4 EXPERIMENT

We use the following five popular datasets in multi-task learning domain to evaluate our XMA scheme: CUBS [25], Stanford Cars [26],

Table 1: Multi-task adaption accuracy (%)

Mask	No Mask		Ele-wise	Col-wise
	4-bit Quantization backbone model			
Dataset	Finetune Floating Number	Finetune Quantized Model	Piggyback[1]	Shift Mask 3 levels
CUBS	82.83	78.5	74.47	**80.07**
Stanford_cars	91.83	85.1	86.85	**88.32**
flowers	96.56	93.8	91.09	**95.59**
Wikiart	75.60	71.2	68.97	**72.6**
Sketches	80.78	76.3	78.88	**79.62**
Mask overhead	0%	0%	25%	0.87%

Table 2: The impact of different shift levels

	Column-wise Mask			
Dataset	Shift Mask	Shift Mask	Shift Mask	Binary Mask
Shift Levels	3	2	1	0
Mask Levels	$[0,\frac{1}{8},\frac{1}{4},\frac{1}{2},1]$	$[0,\frac{1}{4},\frac{1}{2},1]$	$[0,\frac{1}{2},1]$	[0,1]
CUBS	**80.07**	79.67	79.38	77.86
Stanford_cars	**88.32**	88.12	88.02	87.48
flowers	**95.59**	95.14	95.04	95.02
Wikiart	**72.6**	72.51	2.56	71.18
Sketches	79.62	**79.92**	79.92	78.8
Mask overhead	0.87%	0.69%	0.52%	0.35%

Flowers [27], Wikiart [28], and Sketch [29]. We choose the ResNet-50 [30] as our backbone model, which is pre-trained on ImageNet dataset [31].

4.1 Algorithm Evaluation

Table 1 shows the inference accuracy on different datasets. Here we use the backbone ResNet-50 trained on the ImageNet dataset with 4-bit weight and 4-bit activation quantization. The quantization method is adopted from PROFIT [19]. We choose the group size $G = 8$ in the experiment.

For floating-point precision, fine-tuning has the highest flexibility to change any weight to any level. Thus, fine-tuning the backbone model with floating-point precision achieves the best accuracy in most datasets. To simulate the crossbar inference behavior, we quantize the backbone model to 4-bit precision. Compared to floating-point weights, the 4-bit quantized weight sacrifices the representability, and fine-tuning the quantized backbone model shows some accuracy degradation as expected. The Piggyback scheme adopts the binary element-wise mask, where the binary precision of the mask further limits the flexibility of the quantized model. Thus Piggyback shows slightly worse accuracy than fine-tuning on the quantized backbone model.

Benefiting from the more representation levels, fewer training parameters, and gumbel sigmoid trick, our shift-based mask learning method shows consistently higher accuracy on all datasets than Piggyback and fine-tuning on quantized backbone model (Table 1). Although our proposed method has the group concept, it contaminates the accuracy further. Our shift-based mask provides more representation levels than the simple on/off scheme in the binary mask. The group concept also helps to cut down the training parameters, which boosts the training convergence speed. Moreover, sharing the mask value among the entire column significantly saves the memory overhead for mask storage.

Table 3: Hardware specification

RRAM Sub-Array		
Components	**Area** (μm^2)	**Energy** (pJ)
Memory Array (72 × 72)	84.93	
Switch Matrix (WL and SL)	457.3	1.1
SAR ADC (5-bit)	8,409.3	8.3
Shift-Add-Input	1,412.9	6.8
Shift-Add-Weight (2 col use 1)	825.8	1.0
Mask Buffer (72 × 1)	190.4	0.003/bit/access
Total	11,380.2	17.2
Peripheral Circuits		
1 stage AdderTree (128 units)	2,510.3	4.4
2 stage AdderTree (128 units)	7,740.1	13.7
3 stage AdderTree (128 units)	18,408.8	32.6
Global Buffer (64 × 112 × 112 × 4)	8,490,034	0.003/bit/access
ReLU (128 units)	939.5	0.9

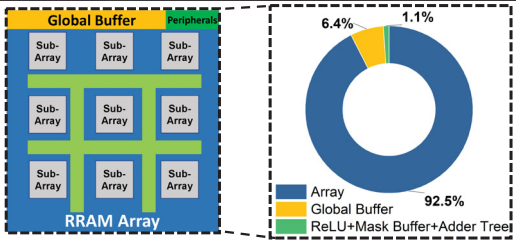

Figure 5: Area breakdown of 4-bit ResNet-50 backbone model hardware deployment. The peripheral circuits including ReLU module, adder tree, and mask buffer

Different shift levels not only determine the mask storage overhead but also affect the accuracy. Table 2 shows the accuracy and mask overhead for different shift levels. More shift levels show better accuracy in the cost of more mask overhead, where the mask overhead is defined as the complete storage required by the mask over the storage required by all the weights in the backbone model. As the shift level goes down, one extreme example is when there is no shift level available between the range of [0,1], which means the mask only has binary value. In that case, our shift-based mask method is equivalent to the column-wise binary mask. Due to the group mask sharing, binary group mask size is only $\frac{1}{72}$ of Piggyback. For the ResNet-50 backbone model, Piggyback's element-wise binary mask requires $23M/8 = 2.88MB$, while the binary mask only consumes around 40KB. Although the binary mask claims the least mask overhead, it achieves the worst accuracy than other shift-based methods. To achieve the best accuracy, three shift levels only require less than 100KB storage for the mask, which is only 3.4% of that in Piggyback (i.e., 96.6% reduction). Despite this reduction, on average, the accuracy is 3.2% higher than Piggyback.

4.2 Hardware Evaluation

We implement the proposed algorithm on hardware as shown in Fig. 4. The hardware performance of different algorithms is evaluated based on the circuit level simulator NeuroSim [32]. The 4-bit quantized targeted DNNs are characterized by the 2-bit per cell HfO_2 1T1R ReRAM devices, characterized from [33] and projected to 32nm CMOS node. The ReRAM array characteristics and the total area usage are summarized in Table 3 and Fig. 5. Each ReRAM

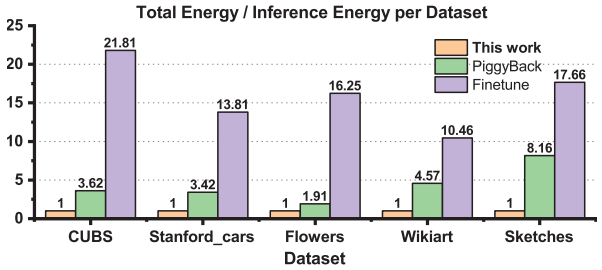

Figure 6: Total inference energy for different methods

column is connected to a 5-bit successive approximation register (SAR) analog-to-digital converter (ADC). To avoid frequent off-chip memory access, we choose the global buffer as the same size as the largest feature map during the inference process.

Fig. 6 shows the overhead caused by the re-programming process of different sub-tasks. The total inference energy includes the energy consumption per dataset and the re-programming energy per sub-task. Given the amount of low-precision weight updates, the re-programming energy can be computed based on the write voltage, write pulses, and conductance level changes [33, 34]. Fine-tuning the model entirely for each individual task requires universal re-programming or even second-time deployment. The largely biased energy consumption between inference and re-programming cumbers pragmatic implementation of continual learning. Piggyback [1] partially programs the weights to zero for different tasks. Compared to the inference energy itself, the additional programming process increases the total energy by up to 8×. Furthermore, re-programming each individual cell requires additional element-wise sparse indexes, leading to intricate array-level manipulation.

Different from the naive fine-tuning or Piggyback [1] learning, the proposed algorithm update the model in a structured manner. Activating RRAM columns for different sub-tasks simplifies the continuous model updating without any re-programming or fine-grained indexes. Compared to the fine-tuning and Piggyback [1], the proposed algorithm reduces the total energy consumption up to 21.81× and 8.16× respectively, as shown in Fig. 6. Such significant energy reduction of the proposed XMA algorithm fully unleashes the practical merit of continual learning.

5 CONCLUSION

In summary, we proposed XMA, a shift-based crossbar mask, to efficiently deploy the multi-task adaption to crossbar-based DNN accelerator while considering hardware costs. The main contribution of XMA is that it does not need to change the neural network structure or re-program any ReRAM cell for new task learning. Moreover, the XMA reuses the existing shift-adder to apply the shift-based mask onto fixed weight and minimize the hardware overhead. Compared with other mask-based methods, XMA significantly saves inference energy and reduces the mask size to less than 1% while maintaining similar accuracy.

ACKNOWLEDGMENTS

This work is supported in part by the National Science Foundation under Grant No.2003749, No.1931871, No. 2144751

REFERENCES

[1] A. Mallya *et al.*, "Piggyback: Adapting a single network to multiple tasks by learning to mask weights," in *European Conference on Computer Vision (ECCV)*.

[2] L. Yang *et al.*, "Ksm: Fast multiple task adaption via kernel-wise soft mask learning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 13 845–13 853.

[3] S. Kornblith, J. Shlens, and Q. V. Le, "Do better imagenet models transfer better?" in *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019.

[4] S. Mittal, "A survey of reram-based architectures for processing-in-memory and neural networks," *Machine Learning and Knowledge Extraction*, vol. 1, no. 1, pp. 75–114, 2019.

[5] C. Eckert *et al.*, "Neural cache: Bit-serial in-cache acceleration of deep neural networks," in *Proceedings of the 45th Annual International Symposium on Computer Architecture*, ser. ISCA '18. IEEE Press, 2018, p. 383–396.

[6] D. Fan *et al.*, "Energy efficient in-memory binary deep neural network accelerator with dual-mode sot-mram," in *2017 IEEE International Conference on Computer Design (ICCD)*, 2017, pp. 609–612.

[7] M. Hu *et al.*, "Dot-product engine for neuromorphic computing: Programming 1t1m crossbar to accelerate matrix-vector multiplication," in *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2016, pp. 1–6.

[8] J. Woo *et al.*, "Optimized programming scheme enabling linear potentiation in filamentary hfo2 rram synapse for neuromorphic systems," *IEEE Transactions on Electron Devices*, vol. 63, no. 12, pp. 5064–5067, 2016.

[9] L.-E. Yu *et al.*, "Structure effects on resistive switching of Al/TiO_x/Al devices for rram applications," *IEEE Electron Device Letters*, vol. 29, no. 4, pp. 331–333, 2008.

[10] F. Zhang *et al.*, "Cccs: Customized spice-level crossbar-array circuit simulator for in-memory computing," in *Proceedings of the 39th International Conference on Computer-Aided Design*, ser. ICCAD '20, 2020.

[11] M. E. Fouda *et al.*, "Mask technique for fast and efficient training of binary resistive crossbar arrays," *IEEE Transactions on Nanotechnology*, vol. 18, pp. 704–716, 2019.

[12] S.-A. Rebuffi *et al.*, "Learning multiple visual domains with residual adapters," in *Advances in Neural Information Processing Systems*, 2017, pp. 506–516.

[13] A. Rosenfeld *et al.*, "Incremental learning through deep adaptation," *IEEE transactions on pattern analysis and machine intelligence*, 2018.

[14] S. Liu *et al.*, "End-to-end multi-task learning with attention," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 1871–1880.

[15] I. Hubara *et al.*, "Binarized neural networks," in *Advances in neural information processing systems*, 2016, pp. 4107–4115.

[16] M. Mancini *et al.*, "Adding new tasks to a single network with weight transformations using binary masks," in *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*, 2018, pp. 0–0.

[17] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou, "Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients," 2016.

[18] J. Choi, S. Venkataramani, V. Srinivasan, K. Gopalakrishnan, Z. Wang, and P. Chuang, "Accurate and efficient 2-bit quantized neural networks." in *MLSys*.

[19] E. Park *et al.*, "Profit: A novel training method for sub-4-bit mobilenet models."

[20] E. Jang, S. Gu, and B. Poole, "Categorical reparameterization with gumbel-softmax," *arXiv preprint arXiv:1611.01144*, 2016.

[21] F. Zhang *et al.*, "Mitigate parasitic resistance in resistive crossbar-based convolutional neural networks," *J. Emerg. Technol. Comput. Syst.*, vol. 16, no. 3, 2020.

[22] X. Peng *et al.*, "Optimizing weight mapping and data flow for convolutional neural networks on rram based processing-in-memory architecture," in *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2019.

[23] Z. Zhu *et al.*, "Mixed size crossbar based rram cnn accelerator with overlapped mapping method," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2018.

[24] W. Wan *et al.*, "A voltage-mode sensing scheme with differential-row weight mapping for energy-efficient rram-based in-memory computing," in *2020 IEEE Symposium on VLSI Technology*, 2020, pp. 1–2.

[25] C. Wah *et al.*, "The Caltech-UCSD Birds-200-2011 Dataset," California Institute of Technology, Tech. Rep. CNS-TR-2011-001, 2011.

[26] J. Krause *et al.*, "3d object representations for fine-grained categorization," in *2013 IEEE International Conference on Computer Vision Workshops*, 2013, pp. 554–561.

[27] M.-E. Nilsback *et al.*, "Automated flower classification over a large number of classes," in *2008 Sixth Indian Conference on Computer Vision, Graphics Image Processing*, 2008, pp. 722–729.

[28] B. Saleh *et al.*, "Large-scale classification of fine-art paintings: Learning the right metric on the right feature," *CoRR*, vol. abs/1505.00855, 2015.

[29] M. Eitz *et al.*, "How do humans sketch objects?" *ACM Trans. Graph. (Proc. SIGGRAPH)*, vol. 31, no. 4, pp. 44:1–44:10, 2012.

[30] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition."

[31] O. Russakovsky *et al.*, "Imagenet large scale visual recognition challenge," 2015.

[32] X. Peng *et al.*, "DNN+NeuroSim: An end-to-end benchmarking framework for compute-in-memory accelerators with versatile device technologies," in *IEEE International Electron Devices Meeting (IEDM)*, 2019, pp. 32.5.1–32.5.4.

[33] W. Wu *et al.*, "A methodology to improve linearity of analog RRAM for neuromorphic computing," in *IEEE Symposium on VLSI Technology*, 2018, pp. 103–104.

[34] P.-Y. Chen *et al.*, "Neurosim: A circuit-level macro model for benchmarking neuro-inspired architectures in online learning," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 12, pp. 3067–3080, 2018.