PRISM: A Programming-Free On-Device Multi-task Adaptation Framework for ReRAM-based Computing-in-Memory Accelerator

Yihang ZUO
Arizona State University
Phoenix, USA
yhzuo@asu.edu

Wanhao Yu
University of North Carolina at Charlotte
Charlotte, USA
wyu6@charlotte.edu

Asmer Ali
Arizona State University
Phoenix, USA
aali115@asu.edu

Li Yang
University of North Carolina at Charlotte
Charlotte, USA
lyang50@charlotte.edu

Deliang Fan
Arizona State University
Phoenix, USA
dfan@asu.edu

Abstract

Resistive random-access memory (ReRAM) crossbar arrays, known for high parallelism and energy efficiency, have been widely adopted to accelerate neural network (NN) inference. However, enabling on-device training on ReRAM-based accelerators remains challenging due to their high programming energy, voltage, and limited endurance. In this paper, we propose PRISM, a novel ReRAM programming-free on-device multi-task adaptation framework, enabling task adaptation via novel lightweight task-specific attribute prompt generation and step mask learning, with ultra-lightweight hardware and memory overhead, and eliminating energy-intensive ReRAM cell reprogramming. Specifically, PRISM first builds a task-specific attribute library using the frozen backbone model deployed in ReRAM by clustering representative features from a subset of the target new task dataset. Then, it computes the correlation between each input and the attribute library to generate a prefix prompt embedding. In addition, PRISM learns a novel crossbar column-wise and hardware-friendly step mask for each new task while keeping the backbone fixed. Extensive experiments show that the total training energy of PRISM is only 0.03% of that of all-parameter fine-tuning, with only 3.9% area overhead. Moreover, compared with the state-of-the-art multi-task adaptation method, PRISM improves accuracy by an average of 3.6% in the DeiT model for standard multi-task adaptation datasets.

CCS Concepts

• **Hardware** → *Memory and dense storage.*

Keywords

Computing-in-Memory, On-device Training, Multi-task Adaptation

1 Introduction

Recent advances in pre-trained models (PTMs), built on large-scale datasets and transformer-based architectures [1, 6, 30], have significantly accelerated the development of artificial general intelligence, driving substantial societal and technological progress. These models are typically pre-trained on massive datasets and then fine-tuned for specific downstream tasks [12, 24]. As a result, the number of parameters that need to be trained and updated grows rapidly with the diversity of downstream tasks, and even a full model may need to be trained for each task. Inspired by this, researchers started developing algorithms that could incrementally adapt a shared pre-trained model to a sequence of downstream tasks while preserving performance on previously learned tasks [41]. This process of gradually adapting the model to learn from various tasks is known as multi-task adaptation [2, 3]. Although Fine-tuning [17] shows good accuracy on newly learned tasks, overwriting 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 adaptation [22, 41].

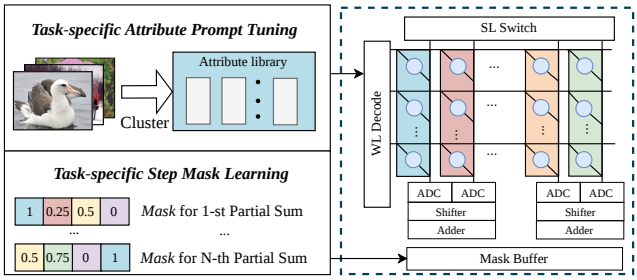


Figure 1: (a) PRISM's ReRAM programming-free task adaptation mechanism, including attribute prompt generation (for activation distribution) and step-mask learning (for weight modulation). (b) hardware mapping onto a ReRAM crossbar array, where the step mask is applied to partial sums via shift operation.

On the hardware side, ReRAM-based computing-in-memory (CIM) architectures have shown great potential for accelerating neural network inference due to their high parallelism and energy efficiency [4, 23]. However, enabling on-device training on ReRAM remains fundamentally challenging. Unlike inference, training requires frequent updates of weights, which in ReRAM are implemented through conductance programming. Such programming operations are energy-intensive, slow, and severely limited by device endurance, making repeated weight updates impractical. As a result, weight programming becomes the primary bottleneck for

realizing multi-task adaptation on ReRAM-based accelerators. However, in ReRAM CIM accelerators, this demand acts as an amplifier of hardware challenges: each new task typically or switching to old tasks, introducing additional weight updates. The sequential learning process of multi-task adaptation significantly magnifies the reprogramming overhead, including energy consumption, endurance, and reliability. As the number of tasks grows, the burden of repeated conductance programming accumulates, making direct on-device adaptation increasingly inefficient and impractical. Accordingly, unlike traditional one-time mapping for inference on a single specialized task, ReRAM-based multi-task adaptation requires coordinated algorithm-mapping co-design to minimize or avoid weight reprogramming.

Previous multi-task adaptation methods can be broadly categorized into two main types: mask-based and parameter-efficient finetuning-based methods. 1) Mask-based learning approaches, such as Piggyback [21] and MEAT [37], freeze the backbone model while learning task-specific binary masks (with values in 0, 1) at the element-wise level. While per-cell manipulation is theoretically feasible, it incurs prohibitive overhead in terms of programming latency, energy consumption, and device endurance in practical ReRAM systems. XMA [40] proposes a column-wise mask pattern for CNNs, where subnetworks are obtained by modulating the weights via masking, following the intuition of the lottery ticket hypothesis [8, 38]. However, these methods rely solely on fixed backbone weights and binary (or uniform) masks, which provide limited learnable degrees of freedom, thereby restricting the model's adaptation capacity. 2) Adapter-based methods [2, 11] require learning many additional parameters, which leads to frequent writes. These can significantly degrade ReRAM device endurance, ultimately leading to a loss of accuracy over time. Prompt-based approaches [15, 43] would learn a prompt pool during training and query it to construct prefix prompts that modify the distribution of activations. In some cases, the prompt pool to be learned can exceed the size of the adapters, which could not be stored in the SRAM buffer. Overall, these methods fail to strike a balance between adaptation capability and hardware efficiency, making them unsuitable for on-device multi-task adaptation with ReRAM.

To tackle these crucial and practical issues, we propose PRISM, a ReRAM-friendly **programming-free** on-device multi-task adaptation paradigm that adapts tasks without modifying the stored backbone model weights in ReRAM arrays. Instead of updating ReRAM conductance values, our approach leverages a lightweight attribute prompt library and column-wise step-mask mechanisms to modulate model behavior on new task data while keeping the backbone frozen and storing all parameters in the SRAM buffer. As shown in Fig. 1(a), our PRISM framework is distinguished from prior works in the following two major aspects:

ReRAM-friendly Attribute Prompt Tuning. PRISM first constructs a pre-calculated prompt library by extracting representative features from the target new task training dataset, reducing trainable parameters, and writing energy during prompt training. During learning, PRISM computes the correlation between the input vector and the library, then dynamically generates prompt embeddings using similarity-based attention scores in a frozen-backbone setting. Unlike previous prompt-tuning methods, PRISM inserts a prompt before the feature of the next attention block, enabling higher parallelism.

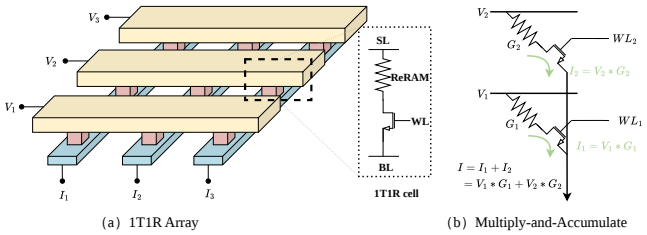
ReRAM-friendly Shift-based Step Mask and Resource-aware Loss Function. PRISM learns a column-wise *step mask* that quantizes values uniformly into discrete levels. Compared with previous binary or shift mask-based methods, the representative ability of the step mask is more uniform and stronger. Furthermore, to maximize hardware utilization, we propose a novel resource-aware loss function for new-task learning that regularizes the step-mask distribution based on target ReRAM hardware configurations.

Based on the above contributions, PRISM enables efficient multi-task learning without reprogramming the backbone model once deployed in a ReRAM crossbar, resulting in significant performance improvements and reduced energy consumption.

2 Background

2.1 ReRAM-based NN accelerator

To mitigate the memory wall bottleneck, researchers have proposed ReRAM-based CIM architectures that perform in-situ matrix-vector multiplications (MVMs) [23, 34]. In analog CIM architectures, vector and matrix data are represented using voltage, current, and conductance values. The MVM operations are executed directly in the analog domain, leveraging Kirchhoff's Current Law and Ohm's Law to achieve efficient and precise computations [7], as shown in Fig. 2.



(a) 1T1R Array (b) Multiply-and-Accumulate

Figure 2: (a) Structure of a ReRAM 1T1R crossbar array, (b) the working principle of ReRAM crossbar.

When using a 1T1R array for the MVM 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}. As illustrated in Fig. 2(b), each ReRAM cell follows Ohm's law and converts the applied row voltage and stored conductance into current, i.e., $I_{ij} = V_i G_{ij}$ (e.g., $I_1 = V_1 G_1$ and $I_2 = V_2 G_2$). The column current is then accumulated by Kirchhoff's Current Law as $I_j^{\text{col}} = \sum_i V_i G_{ij}$, which naturally implements analog multiply–accumulate operations. An $m \times n$ crossbar array can perform the MVM operation in one step, reducing the time complexity from $O(mn)$ to $O(1)$. Therefore, by encoding the input vector as voltages and the weight matrix as conductance, the crossbar directly performs in-situ matrix-vector multiplication.

2.2 Multi-task Adaptation

Prompt-based method. Prompt-based multi-task adaptation methods freeze the pre-trained backbone and adapt the model to new tasks by learning task-specific prompt embeddings [15, 27, 32]. These prompts are concatenated into the backbone model's input

token sequence or intermediate hidden states, allowing the model to adapt to specific tasks without directly updating the backbone weights. Conventional prompt-tuning methods [15, 27, 32] often maintain a prompt pool and use an input-dependent query function, such as cosine similarity or L2 distance, to retrieve prompts for each input sample. The retrieved prompts are then prepended to the input sequence or intermediate hidden input features of each attention block. Although this design is more parameter-efficient than full fine-tuning, the learnable prompt pool can still be prohibitively large for ReRAM-based on-device adaptation, resulting in substantial memory writes and high energy cost. For example, CODA-Prompt [27] needs a $100 \times L \times d$ prompt pool for each 12 attention blocks, where 100 refers to the pool size, L is the prompt length, and d refers to the model embedding length. To reduce the number of trainable parameters, recent studies propose constructing the prompt pool from representative visual attributes, such as color, shape, texture, and local object parts, instead of optimizing a large prompt pool entirely from scratch [16, 28].

Mask-based method. Mask-based methods achieve task adaptation by learning task-specific masks over a frozen pretrained backbone, thereby deriving a dedicated subnetwork for each task without directly updating the original weights. Depending on the method, the learned masks can be binary [21] or multi-level [37, 38], and can operate at different granularities, ranging from individual weights to structured groups [40]. In general, these methods optimize real-valued mask parameters during training and then quantize them into discrete masks for deployment. Because the discretization operation is non-differentiable, the straight-through estimator [13] is commonly used to approximate gradients by passing them through the non-differentiable function during backpropagation.

3 Proposed PRISM Framework

3.1 Overview

We propose PRISM, a programming-free on-device multi-task adaptation framework for ReRAM-based accelerators through joint algorithm-hardware co-design. As shown in Fig. 3, the key idea of PRISM is to enable efficient adaptation to new tasks without modifying the pretrained backbone weights stored in ReRAM, thereby avoiding costly and unreliable weight reprogramming. To achieve this, PRISM co-designs lightweight task-specific learning algorithms together with hardware-aware execution and mapping strategies, so that both adaptation capability and on-device training efficiency can be improved simultaneously.

On the algorithm side, PRISM employs two complementary task-specific adaptation methods: **attribute prompt tuning** and **step-mask learning** with a resource-aware loss function. The attribute prompt tuning method adapts the pretrained backbone by introducing task-specific prompts derived from a pre-calculated attribute library, enabling input-aware adaptation under a frozen-backbone setting. The step-mask learning method further improves adaptation capacity by learning uniform step-wise masks that control how pretrained weights are used, thereby forming a task-specific subnetwork without modifying the original weights.

On the hardware mapping side, PRISM improves on-device training efficiency through two co-designed techniques. First, PRISM introduces a **parallel feedforward dataflow** to increase computation parallelism and reduce latency. By decoupling prompt query generation from backbone computation and executing them in parallel, this dataflow removes unnecessary sequential dependency and improves hardware utilization. Second, PRISM adopts a **resource-aware constraint and mapping** scheme to support efficient hardware implementation of the learned step-wise masks. By constraining the learned masks to match shift-friendly hardware operations, this scheme replaces costly multiplications with lightweight shift-based computation and enables efficient mapping onto the target accelerator. To ensure compatibility between the learned model and hardware constraints, PRISM further incorporates a **violation-fix and prompt-retraining procedure**. Specifically, after mask learning under resource-aware constraints, a violation-fix step is applied to correct mapping inconsistencies, followed by prompt retraining to recover potential accuracy loss. This forms a closed algorithm–mapping co-design loop that jointly enforces hardware efficiency and model performance.

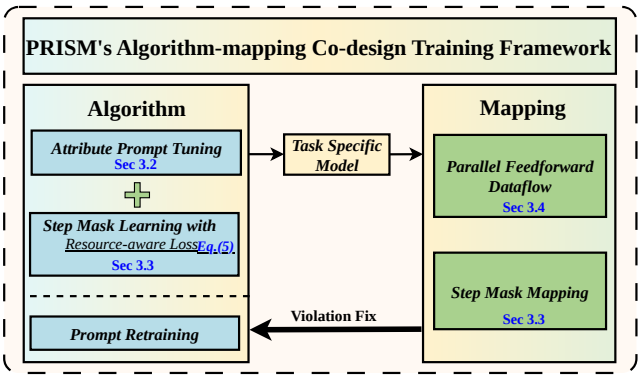


Figure 3: Overview of PRISM's algorithm-mapping co-design.

3.2 Task-specific Attribute Prompt Tuning

Pre-calculated Attribute Library Construction. Inspired by recent work about attribute prompts [16, 28, 43], we propose a novel and efficient prompt-tuning method based on a pre-calculated attribute library. To build these priors, we first sample a representative subset D_t from the task t's training set. Let D_t go through the original backbone and record the classification token (CLS token, the first token) before the classifier for all samples in D_t, as shown in Fig. 4. Then, we apply K-means clustering [20] to partition the feature space into K groups (default $K = 300$), where each cluster captures a dominant semantic pattern shared by visually similar samples. For the i-th cluster, we compute the centroid feature as an attribute prototype a_i, and collect all prototypes into an offline attribute library $\mathcal{P} = \{p_1, p_2, ..., p_K\}$, where $p_i \in R^{d*1}$ and $\mathcal{P} \in R^{d*K}$. This latent attribute library summarizes the high-dimensional feature space into a compact set of task-relevant semantic prototypes, reducing online training cost while preserving discriminative attribute information for subsequent prompt generation.

Dynamic Prompt Embedding Generation. During forward propagation, PRISM retrieves attributes to construct a dynamic attribute prompt for each selected layer, as illustrated in Fig. 4. For each sample, PRISM collects the CLS token from the attention score

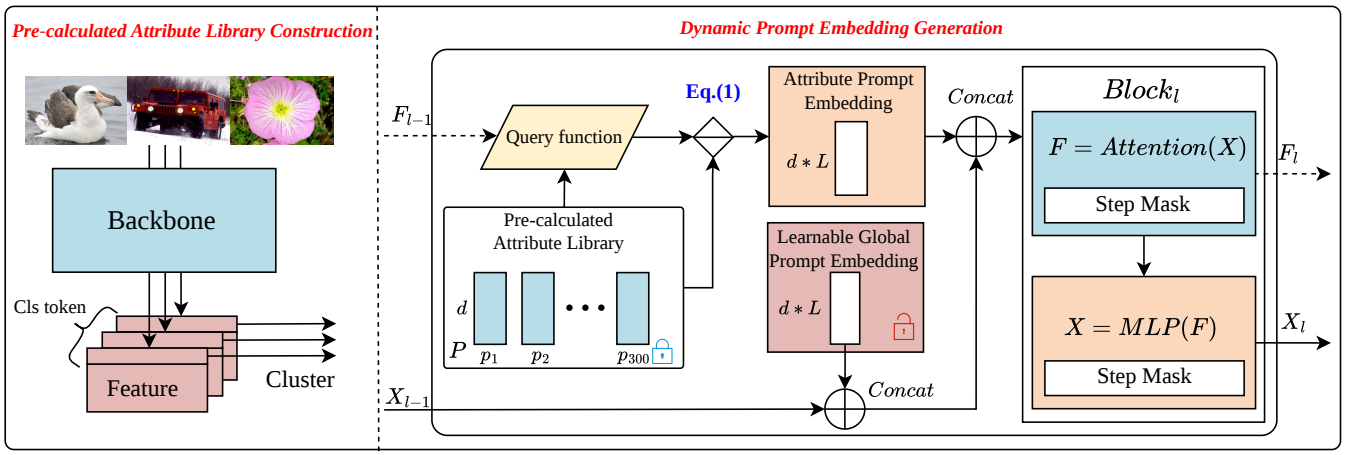


Figure 4: Workflow of the attribute prompt tuning.

map of each attention block. Based on the CLS token value, we select the Top-L token from the input X of block l and record it as $F_l \in R^{L*d}$, where d is the embedding length of the model. Then, PRISM takes the current sample-dependent query F_l and computes its affinity to each prototype like Eq. 1. Second, weighted aggregation over prototypes produces an instance-conditioned attribute component, combined with a set of trainable parameters, that form the attribute prompt embedding. Let $L_l \in \mathbb{R}^{L\times d}$ be a layer-wise learnable prompt template. The total prompt is defined as:

$$\mathbf{P}_a^{(l)} = (1-\lambda_a)L_l + \lambda_a \,\mathrm{Softmax}(F_l\mathcal{P})\mathcal{P}^T, \tag{1}$$

where $\mathrm{Softmax}(F_l\mathcal{P})$ gives affinity-based attention weights over prototypes, and λ_a controls the trade-off between learnable prior prompts and retrieved sample-dependent attributes. In practice, L is set to 30, providing sufficient adaptation capacity with minimal additional tokens. This design improves transfer across tasks while preserving the frozen backbone and reducing the total trainable prompt parameters to $13 \times L \times d$.

Learning with Prompt. During inference, PRISM generates prompts on-the-fly and prepends them to the token sequence before the following transformer blocks. In addition to the attribute prompt, we maintain a short learnable global prompt P_g to provide stable task-level context. The prompt-augmented input is

$$X_l' = \mathrm{concat}(P_g,\ \mathbf{P}_a^{(l)},\ X_l), \tag{2}$$

where X_l is the original token sequence at layer l. The updated sequence X_l' is then processed by both attention and MLP branches under the trainable step masks, as shown in Fig. 4. PRISM also uses decoupled attention [43] to reduce the additional computational cost introduced by sequence length. Since the large prompt library is locked during training, PRISM avoids frequent ReRAM weight reprogramming, thereby preserving deployment efficiency and endurance.

3.3 Step Mask Learning with Resource-aware Loss

3.3.1 *Step Mask Learning.*

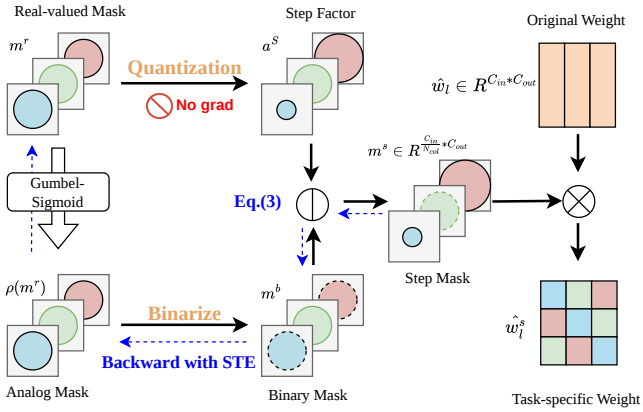


Figure 5: The learning and gradient flow of the step mask learning scheme.

Motivation. To enable efficient task adaptation on ReRAM-based accelerators, the masking mechanism must be carefully designed to align with hardware constraints. Conventional mask-based methods (e.g., MEAT [37]) generate binary masks by binarizing a learnable real-valued mask, where each element indicates whether the corresponding weight is activated or suppressed. While such fine-grained masking improves flexibility, it relies on element-wise scaling operations that are inefficient for ReRAM crossbar implementations, as they require writing each ReRAM cell. To address this limitation, we propose a shift-based column-wise stepping mask for the partial sums computed within each ReRAM array. By combining with our resource-aware loss function, violation fix, and prompt retraining, PRISM could reuse the shifter in the ReRAM array to replace expensive multiplications with lightweight shift-and-add operations, as described in Section 3.3.2.

Overview. As illustrated in Fig. 5, starting from a learnable real-valued mask, we first apply the Gumbel-Sigmoid function to obtain a binary mask, which is then binarized during the forward pass. Based on this binary mask, the step mask is constructed by preserving entries with value 1 and replacing zero entries with learnable

step factors as defined in Eq. 3.

$$m^s = (1 - m^b)a^s + m^b, \text{where } m^b = 0 \text{ or } 1. \tag{3}$$

When $m^b = 1$, the step mask keeps the binary value unchanged. When $m^b = 0$, the zero entry is replaced by the learned step factor. The resulting column-wise specific weights, and during backpropagation, gradients are propagated through the binarization process using the straight-through estimator (STE [39]). In the transformer architecture, there are many large fully connected layers that exceed the column and row capacity of the ReRAM array. Therefore, we need to partition and map them to different ReRAM arrays that share the same inputs. When the input dimension of the weight matrix $C_{in}\times C_{out}$ exceeds the column capacity of a crossbar $N_{row}\times N_{col}$, the weight matrix is partitioned along the dimension and distributed across multiple crossbar columns. The partial sums from different partitions are then accumulated in the adder tree to form the final result.

Learn the binary mask m^b**.** Following KSM [38] and MEAT [37], the binary mask is obtained by binarizing a trainable real-valued mask m^r. To learn the binary mask, we leverage the Gumbel-Sigmoid trick, inspired by Gumbel-Softmax [14] that performs a differential sampling to approximate a categorical random variable. Since the Sigmoid function $\rho()$ can be viewed as a special two-class case of softmax, it can be defined as:

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

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

Learning the step factor a^s**.** To learn the step mask, we adopt a shift-equivalent quantization scheme. In practice, we first normalize the real-valued mask to $[0, 1]$, which serves as an importance factor for weights in multi-task adaptation. The normalized mask is then quantized to the nearest step value (e.g., 3/4, 1/2, 1/4) or to zero. Under this setting, the step mask m^s contains some step levels (3/4, 1/2, 1/4) and two non-step levels (0 and 1). Moreover, the base step size $\frac{1}{2^N}$ is configurable, enabling flexible trade-offs between accuracy and mask overhead. For example, when $N = 1$, the mask supports up to one step level plus two non-step levels: 0, 1/2, and 1. When $N = 0$, the step mask reduces to a binary mask, with minimal mask memory overhead. Note that a mask value of 1 indicates no change, whereas a mask value of 0 disables the current column.

3.3.2 *Resource-aware Constraint Loss and Mapping.*

While the column-wise step mask already improves hardware compatibility by avoiding fine-grained operations, its direct implementation still involves multiplication between partial sums and mask values, which incurs non-negligible latency and energy overhead on ReRAM-based accelerators. Therefore, we reuse the shifter in ReRAM arrays to eliminate it. Specifically, shifters can perform right-shift operations on partial sums efficiently, enabling efficient scaling by factors such as 1/2, 1/4, and 1/8. Step-mask values that cannot be generated by a single shift can be synthesized by combining multiple shifted outputs. For example, the value 3/4 is implemented as the sum of 1/2 and 1/4. This strategy introduces additional partial sums, which may increase the number of stages in the adder tree. However, in practice, we observe that the adder tree has some redundancy, as shown in Fig. 6. This sparsity arises from two factors: (1) zero entries in the mask, and (2) zero-padding required when the number of partial sums is not a power of two.

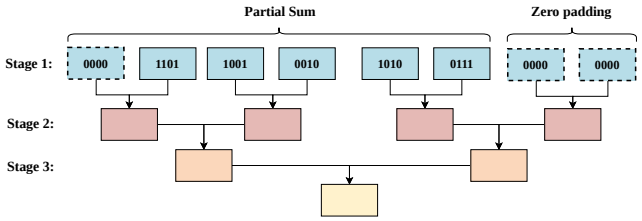


Figure 6: The architecture of a 3-stage adder tree.

This observation motivates us to design a resource-aware loss function and constrain the distribution of step masks to avoid increasing the number of adder-tree stages as in Eq. 5.

$$L_{\text{m}} = \sum_l \sum_j \left[\text{ReLU}\left(c_{3/4}^{(l,j)} - c_0^{(l,j)} - \frac{1}{3} \right) \right]^2 \tag{5}$$

Here, $c_m^{(l,j)}$ denotes the proportion of mask value m in the j-th column of layer l. The magic number $\frac{1}{3}$ corresponds to the zero-padding ratio in the partial sum. For example, the number of DeiT-Small's partial sums is $C_{in}/N_{row} = 384/64 = 6$, as shown in Fig. 6. So, a 3-stage adder tree with 8 inputs is required, introducing $8-6 = 2$ zero-padded inputs. Hence, the allowable proportion gap between mask values $c_{3/4}$ and c_0 is bounded by $2/6 = 1/3$. The final loss $\mathcal{L}$ is a combination of the mask learning loss L_m and the classification loss L_c, where L_m is weighted by a coefficient λ_m.

$$\mathcal{L} = L_c + \lambda_m L_{\text{m}} \tag{6}$$

After applying the loss constraint, the violation rate would be very low as shown in Table 5. Then, we apply a mandatory violation fix step to correct remaining mask violations by pushing $\frac{3}{4}$ entries toward 1. Finally, we freeze the mask and retrain the prompt for several epochs to avoid accuracy loss compared with the uncorrected setting.

With this design, the memory-intensive multiplication between step masks and fixed weights is replaced by shift operations, reducing both computation and energy consumption. Moreover, these shift operations can reuse existing shift-add logic in most ReRAM-based IMC platforms, incurring negligible additional hardware overhead.

3.4 Parallel Feedforward Dataflow

As shown in Fig. 7(a), we optimize the original prompt-tuning dataflow by reorganizing computation across transformer layers to improve efficiency and eliminate redundant operations. In the dataflow of vanilla prompt-tuning methods, as in [33, 42], the

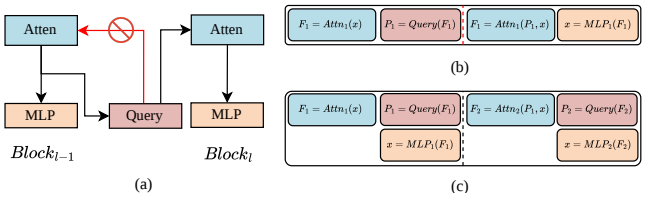


Figure 7: (a) The process of conventional prompt-tuning (red line) and the proposed PRISM. (b) Vanilla prompt-tuning dataflow with sequential dependency. (c) Proposed parallel feedforward dataflow, where query generation and MLP computation are decoupled to improve hardware utilization and reduce latency.

prompt from the query is generated from the attention output and immediately reused within the same stage, introducing unnecessary sequential dependencies between the attention and MLP computations. In our optimized design, we decouple these steps and pipeline operations across consecutive layers. Specifically, the query generated from the attention output of one layer is reused by the attention module of the next layer, while the MLP computation proceeds in parallel with the query process. This restructuring shortens the critical path, increases computational overlap, and improves hardware utilization without changing the model's functional behavior. As a result, the optimized dataflow reduces latency and improves the efficiency of prompt tuning in transformer architectures, as shown in Fig. 7(b) and (c).

4 Hardware

Fig. 8(a) shows the ReRAM-based crossbar ViT accelerator with the proposed column-wise mask. The architecture consists of a global buffer for storing activations, multiple ReRAM crossbars for matrix computation, and a peripheral circuit for other processing (e.g., Softmax, accumulator, and GeLU [10]). Each ReRAM crossbar executes the dominant matrix operations in Transformer blocks (e.g., Q/K/V projections, output projections, and MLP linear layers). A multi-stage adder tree is used to post-process partial sums from the ReRAM arrays. Inside each ReRAM sub-array, quantized linear-layer weights are mapped onto ReRAM cells. Due to device-level multi-bit constraints, one logical weight may span multiple cells/columns. For example, when weights are quantized to 4 bits and each ReRAM cell stores 2-bit conductance states, each weight is split across two adjacent columns for the high and low bits. Since weights are decomposed across columns, each column produces only a partial sum, which the ADC digitizes as low-bit activations. The shift-adder (SA) in Fig. 8(b) then aligns and accumulates these partial activations to recover the full-precision output. The mask buffer stores the column-wise step mask and controls shift/scaling behavior for each column as shown in Fig. 8(c) and (d). Finally, outputs are aggregated by the global adder tree, passed through nonlinear/post-processing units, and forwarded as token features to the next Transformer layer.

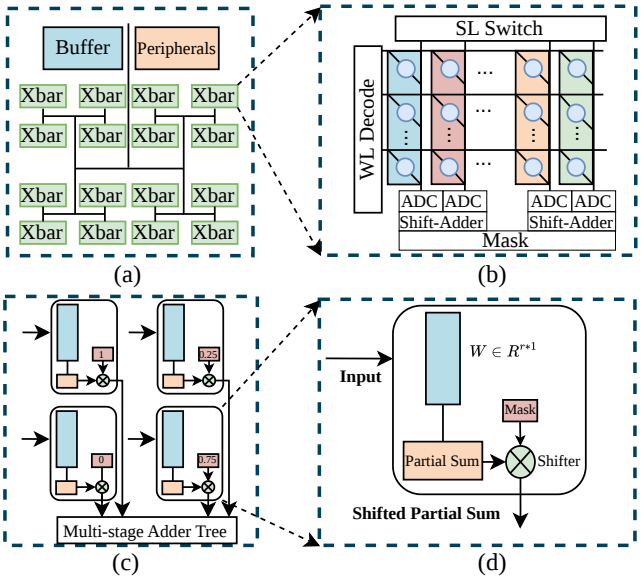


Figure 8: Hardware architecture of the PRISM's ReRAM-based ViT accelerator. (a) System-level organization with the global buffer, ReRAM crossbar arrays, and peripheral modules. (b) ReRAM crossbar with column-wise step masks. (c) Shift-adder datapath that aligns and accumulates partial sums from multi-bit weight decomposition. (d) Processing flow of the proposed resource-aware column-wise step mask.

5 Experiment

5.1 Experimental Setup

Baseline Methods. As shown in Table 1 and Table 2, we compare PRISM with six traditional and SOTA methods listed: Classifier, single task fine-tuning(FT), mask-based methods (HAT [26], MEAT [37], and XMA [40]), and parameter-efficient fine-tuning methods (Adaptor-B [11], SHIP [43]). The classifier is a simple multi-task adaptation strategy that finetunes only the classifier layer for the new task. Among these baselines, FT denotes that an independent ViT is trained for each task, whose performance can serve as the upper bound of multi-task adaptation. We extend XMA in the DeiT architecture and adopt the official implementations and the hyperparameter settings.

Datasets. ImageNet [5] is set as the initial task. We evaluate sequential adaptation on three downstream datasets: CUBS [31], Stanford Cars [18], and CIFAR-100 [19] sequentially. After all adaptations, we would evaluate the accuracy on the original ImageNet dataset to get their difference (called **forgetting**).

Hardware platform. We implement the PRISM on hardware as shown in Fig. 8. The hardware performance of different algorithms is evaluated based on the circuit-level simulator NeuroSim V2.1 [9, 25, 44]. The 4-bit quantized targeted DeiT is implemented using 2-bit-per-cell HfO2 1T1R ReRAM devices, as reported in [35] and projected to the 32nm CMOS node.

Experiment Details. For the initial ImageNet task, we directly utilize the official open-source pre-trained weights [29] and APHQ-ViT's released PTQ INT4 version [36]. We use two transformer backbones shown in Table 1 and Table 2: DeiT-Tiny and DeiT-Small.

We operate our experiments on a server equipped with an NVIDIA Ada5000 GPU and train all experiments for 50 epochs with a batch size of 48. PRISM would retrain prompts for 10 epochs after the violation fix. Based on the trainable parameter ratio and position of baseline methods, we select the ReRAM array and the SRAM buffer to store and update them.

5.2 Algorithm Performance

We evaluate PRISM on both DeiT-Small and DeiT-Tiny under INT4 and FP32 precision, as shown in Tables 1 and 2. Since only the INT4 version can be deployed on the ReRAM array, and some baseline methods do not support INT4 results, we report FP32 accuracy for broader algorithmic comparison. It can be observed that the only classifier training has the lowest accuracy and writing energy overhead, indicating the lower bound for all algorithms. While fine-tuning is ideal from an accuracy perspective, it increases the model's parameters by about 3 times, as 3 additional independent models are trained for the new tasks, which violates the multi-task learning setting and leads to forgetting.

Accuracy. Adaptor-B, which inserts lightweight adapters, and SHIP, which relies on prompt-based semantic adaptation, both deliver noticeably lower accuracy, suggesting that previous methods are insufficient for strong multi-task adaptation. For FP32 precision, PRISM achieves 82.77% on DeiT-Small and 74.98% on DeiT-Tiny, which is close to full fine-tuning (85.49% and 76.27%, respectively) while outperforming the other parameter-efficient baselines. Across both backbones, PRISM also achieves the best INT4 accuracy (79.21% on DeiT-Small and 68.45% on DeiT-Tiny), surpassing MEAT (75.49% and 64.30%) and XMA (76.88% and 63.60%). These improvements mainly come from the complementary design of attribute prompt tuning and step-mask modulation: the prompts provide sample-aware task adaptation, while the masks selectively reuse pretrained knowledge in the frozen quantized backbone, leading to stronger transfer under low-precision constraints.

Forgetting. From the forgetting metrics, FT shows the largest forgetting (30.06% on DeiT-Small and 41.38% on DeiT-Tiny) because it updates the entire backbone, leading to substantial drift in previously learned representations. Adaptor-B is N/A because it was not evaluated under a multi-task adaptation setting, and thus does not report forgetting. In contrast, most parameter-efficient methods, including PRISM, report zero forgetting in these tables, indicating better retention of prior knowledge by keeping backbone weights fixed and learning only lightweight task-specific modules. Compared with methods that also exhibit zero forgetting, PRISM further provides higher accuracy, demonstrating that it not only mitigates catastrophic forgetting but also preserves strong task adaptability, yielding a more favorable stability-plasticity balance for continual adaptation.

5.3 Hardware Evaluation

Writing Energy. As shown in the Table 1 and Table 2, FT, HAT, and MEAT directly update conductance states in crossbar cells, which incurs expensive in-situ programming overhead. In contrast, methods using *SRAM Buffer* writing (Adaptor-B, SHIP, XMA, and PRISM) keep the quantized backbone fixed in ReRAM and only write lightweight task-specific parameters in buffers, avoiding frequent ReRAM reprogramming and significantly reducing energy. PRISM follows this buffer-based principle and achieves low writing energy (0.03% of full fine-tuning on DeiT-Tiny and 0.02% on DeiT-Small) while maintaining the best accuracy. Compared to SHIP and XMA, PRISM's writing energy increased by only approximately 0.02% while its accuracy was significantly better, demonstrating a favorable efficiency-accuracy trade-off for multi-task adaptation.

Area Breakdown & Hardware Overhead. The ReRAM array characteristics and the total area usage are summarized in Table 3. Each ReRAM column is connected to a 5-bit successive approximation register (SAR) analog-to-digital converter (ADC). As shown in Table 3, the total hardware area is dominated by the ReRAM compute macros, with the SAR ADC accounting for 74.2% of the area. The remaining components, the global buffer and the SIMD + adder tree, occupy relatively moderate portions (5.4% and 0.6%). Notably, the proposed mask buffer and the pre-calculated prompt library introduce only about 3.9% total area overhead (1.5% from the mask buffer and 2.4% from the ReRAM crossbar for the prompt library), which remains negligible compared to the core compute modules.

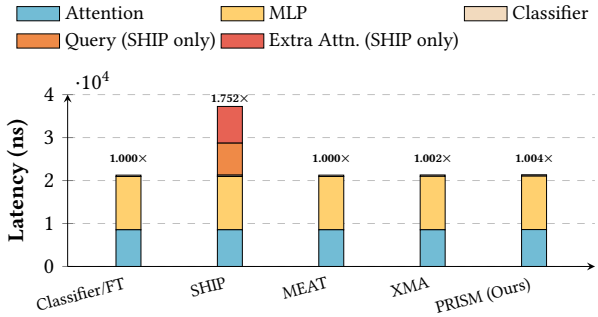


Figure 9: Inference latency breakdown of different methods on the DeiT-Small INT4 backbone. Ratios are normalized to the vanilla Transformer inference path.

Latency. Fig. 9 presents the inference latency breakdown of different methods on the DeiT-Small INT4 backbone, normalized to the vanilla Transformer inference path. Prompt-based methods such as SHIP [43] require an additional query stage and a second attention pass to select prompts from the prompt pool, thereby increasing the runtime to 1.75× that of the original inference path. In contrast, PRISM removes the extra attention pass and overlaps the attribute-prompt branch with the MLP computation through the proposed parallel feedforward dataflow, so the query-generation latency is hidden behind the MLP latency, and the exposed latency of the two parallel branches is determined by their maximum rather than their sum. The only remaining overhead comes from the step mask operation, which takes one additional cycle to perform the shift after the ADC readout, increasing the total latency by merely 0.4%.

5.4 Ablation Study

Effectiveness of Each Component. In Table 4, we demonstrate how our method benefits from each designed component. Five

Table 1: Hardware and algorithm performance comparison on DeiT-Small (22M parameters). ↑ means higher is the better, and ↓ means lower is the better. (Note that only the INT4 version could run on the ReRAM array; we report the FP32 version's accuracy for wider comparison.)

	Methods	Hardware Performance		Algorithm Performance			
		Writing method	Energy (↓)	Avg. Accuracy (↑)		Trainable parameter	Forgetting (↓)
				INT4	FP32		
DeiT-small	Classifier	SRAM Buffer	<0.01%	54.40	57.84	0.36%	0
	FT	ReRAM Array	100%	–	**85.49**	100%	30.06
	Adaptor-B [11]	SRAM Buffer	0.04%	–	76.14	5.54%	0
	SHIP [43]	SRAM Buffer	<0.01%	65.97	71.29	0.56%	0
	HAT [26]	ReRAM Array	66.8%	–	77.47	66.79%	N/A
	MEAT [37]	ReRAM Array	66.8%	75.49	81.55	66.79%	0
	XMA [40]	SRAM Buffer	0.01%	76.88	82.09	1.04%	0
	PRISM (Ours)	SRAM Buffer	0.02%	**79.21**	82.77	2.38%	0

Table 2: Hardware and algorithm performance comparison on DeiT-Tiny (5M parameters). ↑ means higher is the better, and ↓ means lower is the better. (Note that only the INT4 version could run on the ReRAM array; we report the FP32 version's accuracy for wider comparison.)

	Methods	Hardware Performance		Algorithm Performance			
		Writing method	Energy (↓)	Avg. Accuracy (↑)		Trainable parameter	Forgetting (↓)
				INT4	FP32		
DeiT-Tiny	Classifier	SRAM Buffer	<0.01%	42.71	51.08	0.72%	0
	FT	ReRAM Array	100%	–	**76.27**	100%	41.38
	Adaptor-B [11]	SRAM Buffer	0.08%	–	62.25	11.75%	0
	SHIP [43]	SRAM Buffer	<0.01%	49.32	65.88	1.12%	0
	HAT [26]	ReRAM Array	66.7%	–	63.21	66.67%	N/A
	MEAT [37]	ReRAM Array	66.7%	64.30	67.60	66.67%	0
	XMA [40]	SRAM Buffer	0.01%	63.6	73.99	1.55%	0
	PRISM (Ours)	SRAM Buffer	0.03%	**68.45**	74.98	4.75%	0

Table 3: Hardware specification.

Single RRAM Sub-Array		
Components	**Area (μm^2)**	**Energy (pJ)**
Memory Array (64 × 64)	64.11	7124.7
Switch Matrix (WL and SL)	407	1.1
SAR ADC (5-bit)	7484.3	8.3
Shift-Add-Input	1257.5	6.8
Shift-Add-Weight (2 col use 1)	734.96	1.0
Mask Buffer (64 × 1)	169.24	0.003/bit/access
Sum	**10117**	**7142.9**
Total Chip		
ReRAM Arrays (5304 units)	60,140,200	37,028,794
SIMD+AdderTree (1536 units)	355,184.6	619.2
Global Buffer (197 × 1536 × 4)	3,200,213	0.003/bit/access

model variants are listed in each backbone to denote different proposed modules or baselines used for comparison. Concretely, **(1)** the Classifier baseline; **(2)** transformers with attribute prompt tuning; **(3)** transformers with step masks, resource-aware loss function, and violation fix, without prompt retraining; **(4)** transformers with step masks and retraining; **(5)** the proposed PRISM. Our attribute prompt tuning alters the distribution of activations within the self-attention mechanism, leading to a significant performance improvement (about 10%) over only training the classifier. The step mask significantly improves results by dynamically activating or scaling pretrained weights, customizing a subnetwork of the complete transformer for each new task. In conclusion, each adopted component of PRISM consistently promotes multi-task adaptation performance.

Effectiveness of Resource-aware Loss Function. We ablate three key hyperparameters on CUBS, as summarized in Table 5: the regularization coefficient λ_m, the attribute library size, and the prompt length. The violation number denotes the number of additional adder-tree stages introduced by the extra partial sums generated by the shift-based step mask. For λ_m, performance peaks at 1e−1 with 78.82%, while both weaker regularization (1e−2 and

Table 4: Ablation study for CUB dataset on the DeiT-Tiny and DeiT-Small backbones.

Classifier	Mask	Retraining	Prompt	Tiny	Small
✔				56.35	64.60
✔			✔	67.71	78.16
✔	✔			68.14	78.04
✔	✔	✔		68.14	78.11
✔	✔	✔	✔	**69.76**	**79.17**

1e−3) and overly strong regularization reduce accuracy. This trade-off reflects the balance between adaptation flexibility and structural constraint. When λ_m is too large, the constraint becomes overly strong and restricts task adaptation, thereby degrading accuracy. When λ_m is too small, more violations remain to be corrected after training, exceeding the compensation capability of prompt retraining. Notably, the violation number decreases sharply as λ_m increases, confirming that this term effectively controls mask violations.

Determination of Hyperparameters. For library size, accuracy improves from 78.02% ($K = 100$) to a maximum of 79.17% ($K = 300$), then slightly drops at larger sizes, suggesting that a moderate prototype set provides sufficient diversity without introducing redundant attributes. For prompt length, the best result is also 79.17% at length 30; shorter prompts underfit task-specific cues, whereas longer prompts (e.g., 50) introduce redundancy and hurt discrimination. Based on this trend, we use λ_m = 1e−1, library size $K = 300$, and prompt length 30 in the main setting.

Table 5: Hyperparameters comparison for INT4 DeiT-Small on the CUB dataset.

λ_m	Violation Number	Accuracy	Library Size	Accuracy	Prompt Length	Accuracy
1	6	78.59	100	78.02	10	78.42
1e-1	22	**79.17**	200	78.82	20	77.48
1e-2	87	78.73	300	**79.17**	30	**79.17**
1e-3	1453	78.02	400	78.73	40	79.10
0	7112	78.04	500	78.45	50	77.53

6 Conclusion

We propose PRISM, a programming-free on-device multi-task adaptation framework for ReRAM-based IMC accelerators. By combining attribute-prompt tuning with a resource-aware shift-based step mask, PRISM adapts new tasks while freezing backbone weights and avoiding costly ReRAM reprogramming. Results show that PRISM achieves strong improvements in accuracy and efficiency, including up to 3.2% average accuracy over XMA and 99% lower writing energy overhead than fine-tuning.

Acknowledgments

This work is supported in part by the National Science Foundation under Grant No.2314591, No.2505326 and No.2528767

References

[1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774* (2023).

[2] Ahmed Agiza, Marina Neseem, and Sherief Reda. 2024. Mtlora: Low-rank adaptation approach for efficient multi-task learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 16196–16205.

[3] Deblina Bhattacharjee, Sabine Süsstrunk, and Mathieu Salzmann. 2023. Vision transformer adapters for generalizable multitask learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 19015–19026.

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

[5] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 248–255.

[6] Alexey Dosovitskiy. 2020. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929* (2020).

[7] Donato Francesco Falcone, Victoria Clerico, Wooseok Choi, Tommaso Stecconi, Folkert Horst, Laura Bégon-Lours, Matteo Galetta, Antonio La Porta, Nikhil Garg, Fabien Alibart, et al. 2025. All-in-One Analog AI Hardware: On-Chip Training and Inference with Conductive-Metal-Oxide/HfOx ReRAM Devices. *Advanced Functional Materials* (2025), 2504688.

[8] Jonathan Frankle and Michael Carbin. 2018. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635* (2018).

[9] Zexin Fu, Yihang Zuo, Yuzhe Ma, and Jiayi Huang. 2025. Optimizing Heterogeneous Compute-in-Memory with Hybrid Dataflow and In-Network Reduction for Vision Transformer. In *2025 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*. IEEE, 1–7.

[10] Dan Hendrycks and Kevin Gimpel. 2016. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415* (2016).

[11] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for NLP. In *International conference on machine learning*. PMLR, 2790–2799.

[12] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. 2022. Lora: Low-rank adaptation of large language models. *ICLR* 1, 2 (2022), 3.

[13] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2016. Binarized neural networks. *Advances in neural information processing systems* 29 (2016).

[14] Eric Jang, Shixiang Gu, and Ben Poole. 2016. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144* (2016).

[15] Menglin Jia, Luming Tang, Bor-Chun Chen, Claire Cardie, Serge Belongie, Bharath Hariharan, and Ser-Nam Lim. 2022. Visual prompt tuning. In *European conference on computer vision*. Springer, 709–727.

[16] Gahyeon Kim, Sohee Kim, and Seokju Lee. 2024. Aapl: Adding attributes to prompt learning for vision-language models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 1572–1582.

[17] Simon Kornblith, Jonathon Shlens, and Quoc V Le. 2019. Do better imagenet models transfer better?. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2661–2671.

[18] Jonathan Krause, Michael Stark, Jia Deng, and Li Fei-Fei. 2013. 3d object representations for fine-grained categorization. In *Proceedings of the IEEE international conference on computer vision workshops*. 554–561.

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

[20] J MacQueen. 1967. Multivariate observations. In *Proceedings ofthe 5th Berkeley symposium on mathematical statisticsand probability*, Vol. 1. University of California press Oakland, CA, USA, 281–297.

[21] Arun Mallya, Dillon Davis, and Svetlana Lazebnik. 2018. Piggyback: Adapting a single network to multiple tasks by learning to mask weights. In *Proceedings of the European conference on computer vision (ECCV)*. 67–82.

[22] Michael McCloskey and Neal J Cohen. 1989. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*. Vol. 24. Elsevier, 109–165.

[23] Sparsh Mittal. 2018. A survey of ReRAM-based architectures for processing-in-memory and neural networks. *Machine learning and knowledge extraction* 1, 1 (2018), 75–114.

[24] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. Training language models to follow instructions with human feedback. *Advances in neural information processing systems* 35 (2022), 27730–27744.

[25] Xiaochen Peng, Shanshi Huang, Hongwu Jiang, Anni Lu, and Shimeng Yu. 2020. DNN+ NeuroSim V2. 0: An end-to-end benchmarking framework for compute-in-memory accelerators for on-chip training. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 40, 11 (2020), 2306–2319.

[26] Joan Serra, Didac Suris, Marius Miron, and Alexandros Karatzoglou. 2018. Overcoming catastrophic forgetting with hard attention to the task. In *International conference on machine learning*. PMLR, 4548–4557.

[27] James Seale Smith, Leonid Karlinsky, Vyshnavi Gutta, Paola Cascante-Bonilla, Donghyun Kim, Assaf Arbelle, Rameswar Panda, Rogerio Feris, and Zsolt Kira. 2023. Coda-prompt: Continual decomposed attention-based prompting for rehearsal-free continual learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 11909–11919.

[28] Xinyu Tian, Shu Zou, Zhaoyuan Yang, and Jing Zhang. 2024. Argue: Attribute-guided prompt tuning for vision-language models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 28578–28587.

[29] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. 2021. Training data-efficient image transformers & distillation through attention. In *International conference on machine learning*. PMLR, 10347–10357.

[30] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).

[31] Catherine Wah, Steve Branson, Peter Welinder, Pietro Perona, and Serge Belongie. 2011. The caltech-ucsd birds-200-2011 dataset. (2011).

[32] Zhen Wang, Rameswar Panda, Leonid Karlinsky, Rogerio Feris, Huan Sun, and Yoon Kim. 2023. Multitask prompt tuning enables parameter-efficient transfer learning. *arXiv preprint arXiv:2303.02861* (2023).

[33] Zifeng Wang, Zizhao Zhang, Chen-Yu Lee, Han Zhang, Ruoxi Sun, Xiaoqi Ren, Guolong Su, Vincent Perot, Jennifer Dy, and Tomas Pfister. 2022. Learning to prompt for continual learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 139–149.

[34] H-S Philip Wong, Heng-Yuan Lee, Shimeng Yu, Yu-Sheng Chen, Yi Wu, Pang-Shiu Chen, Byoungil Lee, Frederick T Chen, and Ming-Jinn Tsai. 2012. Metal–oxide RRAM. *Proc. IEEE* 100, 6 (2012), 1951–1970.

[35] Wei Wu, Huaqiang Wu, Bin Gao, Peng Yao, Xiang Zhang, Xiaochen Peng, Shimeng Yu, and He Qian. 2018. A methodology to improve linearity of analog RRAM for neuromorphic computing. In *2018 IEEE symposium on VLSI technology*. IEEE, 103–104.

[36] Zhuguanyu Wu, Jiayi Zhang, Jiaxin Chen, Jinyang Guo, Di Huang, and Yunhong Wang. 2025. Aphq-vit: Post-training quantization with average perturbation hessian based reconstruction for vision transformers. In *Proceedings of the Computer Vision and Pattern Recognition Conference*. 9686–9695.

[37] Mengqi Xue, Haofei Zhang, Jie Song, and Mingli Song. 2022. Meta-attention for vit-backed continual learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 150–159.

[38] Li Yang, Zhezhi He, Junshan Zhang, and Deliang Fan. 2021. Ksm: Fast multiple task adaption via kernel-wise soft mask learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 13845–13853.

[39] Penghang Yin, Jiancheng Lyu, Shuai Zhang, Stanley Osher, Yingyong Qi, and Jack Xin. 2019. Understanding straight-through estimator in training activation quantized neural nets. *arXiv preprint arXiv:1903.05662* (2019).

[40] Fan Zhang, Li Yang, Jian Meng, Jae-sun Seo, Yu Cao, and Deliang Fan. 2022. Xma: A crossbar-aware multi-task adaption framework via shift-based mask learning method. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*. 271–276.

[41] Yu Zhang and Qiang Yang. 2021. A survey on multi-task learning. *IEEE transactions on knowledge and data engineering* 34, 12 (2021), 5586–5609.

[42] Kaiyang Zhou, Jingkang Yang, Chen Change Loy, and Ziwei Liu. 2022. Conditional prompt learning for vision-language models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 16816–16825.

[43] Haowei Zhu, Fangyuan Zhang, Rui Qin, Tianxiang Pan, Junhai Yong, and Bin Wang. 2024. Semantic hierarchical prompt tuning for parameter-efficient fine-tuning. *arXiv preprint arXiv:2412.16956* (2024).

[44] Yihang Zuo, Zexin Fu, Cong Wang, Yuchao Wu, Jiayi Huang, and Yuzhe Ma. 2026. Harmony: A Hardware-Mapping Co-Exploration Framework for Hybrid CIM-based Vision Transformer Accelerator. In *2026 27th International Symposium on Quality Electronic Design (ISQED)*. IEEE, 1–8.