Hyb-Learn: A Framework for On-Device Self-Supervised Continual Learning with Hybrid RRAM/SRAM Memory

Fan Zhang*
fzhang62@jh.edu
Johns Hopkins University
Baltimore, Maryland, USA

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

Deliang Fan
dfan10@jh.edu
Johns Hopkins University
Baltimore, Maryland, USA

ABSTRACT

While RRAM crossbar-based In-Memory Computing (IMC) has proven highly effective in accelerating Deep Neural Networks (DNNs) inference, RRAM-based on-device training is less explored due to its high energy consumption of weight re-programming and cells' low endurance problem. Besides, emerging trends indicate a need for on-device continual learning which sequentially acquires knowledge from multiple tasks to enhance user's experiences and eliminate data privacy concerns. However, learning on each new task leads to forgetting prior learned knowledge on prior tasks, which is known as catastrophic forgetting. To address these challenges, we are the first to propose a novel training framework, *Hyb-Learn*, for enabling on-device continual learning with a hybrid RRAM/SRAM IMC architecture design. Specifically, when training each new arriving task, our approach first partitions the model into two groups based on the proposed task-correlated PE-wise correlation to freeze or re-training, and correspondingly mapping to RRAM and SRAM, respectively. In practice, the RRAM stores frozen weights with strong task correlation to prior tasks to eliminate the high cost of weight reprogramming issue of RRAM, while the SRAM stores the remaining weights that will be updated. Furthermore, to maximize the freezing ratio for improving training efficiency while maintaining accuracy and mitigating catastrophic forgetting, we incorporate self-supervised learning algorithms that are initialized from a pre-trained model for training each new task.

1 INTRODUCTION

Continual learning (CL) enables Deep Neural Network models (DNNs) to adapt and acquire new knowledge sequentially over time while retaining previously learned information. This characteristic is vital for AI systems to continually enhance their knowledge and skills throughout their operational lifespan. The main challenge in continual learning is to prevent catastrophic forgetting, a phenomenon where a model's accuracy on previously learned tasks significantly deteriorates when learning new tasks. To mitigate this issue, many continual learning techniques are developed to selectively reuse and consolidate knowledge from past tasks while accommodating new information[1].

From the hardware perspective, the traditional Von-Neumann architecture (e.g., CPU, GPU) suffers from high off-chip data communication energy consumption which is approximately two orders of magnitude higher than the actual data processing. This phenomenon, often referred as the "Memory Wall", poses a significant challenge in achieving efficient and energy-conscious DNN computations. Recently, the RRAM crossbar-based In-Memory Computing (IMC) [2–4] has proven highly effective in expediting the inference process of DNNs, owing to its high storage density, strong parallel computing capacity, and low energy consumption. For example, related to continual learning, [5] proposes an RRAM crossbar column-wise sparse training framework which performs sparse training for each task offline and then maps the well-trained weights to RRAM crossbar for online inference.

Different from online inference, on-device continual learning involves sequentially training the DNN model online for each task, which requires to update weights frequently for optimizing the loss function. Due to the fact that the RRAM write operation (i.e., weight reprogramming) suffers from high latency and energy consumption, performing weight updating in the RRAM-based IMC is quite expensive or even unacceptable. Therefore, how to avoid the expensive weights updating meanwhile taking advantage of its computing and memory capacity becomes the critical issue for on-device DNN training in the IMC. Furthermore, from algorithm perspective, since continually training DNN models on new tasks can result in the forgetting of previously learned knowledge (a.k.a catastrophic forgetting), on-device continual learning needs to improve the training efficiency without sacrificing the learning performance.

To tackle these crucial challenges for enabling on-device continual learning, we are the first to propose *Hyb-Learn*, a novel training framework with designing a hybrid RRAM/SRAM IMC system. Specifically, the architecture of the hybrid RRAM/SRAM IMC system includes RRAM-based IMC macros and SRAM macros with the corresponding auxiliary unit and control logic, where we treat both RRAM and SRAM-based macros as the *MAC* engine which uses processing element (PE) as the basic computing unit. Importantly, these two types of macro have different objectives: RRAM-based PE aims to maximize the data computing in parallel, while the SRAM PE is to modify/update the stored data with less power. Furthermore, to support continual learning with co-design of hardware and software, by taking advantage of the proposed Hybrid RRAM/SRAM IMC system, the proposed training framework incorporates two important strategies:

1) **Model weights freezing and partition via PE-wise task correlations.** Prior to train each new task, we propose a model weight freezing and partition mechanism that separates the weights into two distinct groups: trainable weights and frozen weights, where the trainable weights are mapped to SRAM, and the frozen

*Both authors contributed equally to this research.

DAC '24, June 23–27, 2024, San Francisco, CA, USA

ACM ISBN 979-8-4007-0601-1/24/06…$15.00
https://doi.org/10.1145/3649329.3657389

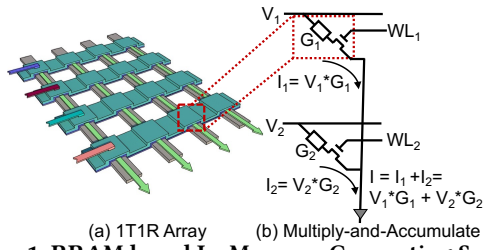

Figure 1: RRAM based In-Memory-Computing Systems

weights are mapped to RRAM. This freezing and partition is based on a defined *PE-wise task correlation* between the current task and previously trained tasks with the objective to only update the uncorrelated weights. By doing so, we can guarantee that the frozen weights will not be updated during training, avoiding high RRAM re-programming time and energy consumption, importantly also mitigating catastrophic forgetting in continual learning. Furthermore, we propose to freeze the weights in processing element wise (i.e., PE-wise), which is the basic parallel computing core in our hybrid RRAM/SRAM architecture. It helps to further improve the efficiency of the computation and data flow.

2) ***Continual learning via self-supervised learning.*** While weight freezing can enhance training efficiency, it generally leads to a degradation in DNN model accuracy especially large weight freezing ratio. To mitigate this issue, we employ a self-supervised learning algorithm [6] to train the model for each new task. It helps the model learning more general features that are proven to mitigate catastrophic forgetting and more robust when weight freezing is applied. Furthermore, we enhance the PE-wise weight freezing ratio to reduce the computation cost by initializing the weights using a pre-trained model.

2 RELATED WORKS

2.1 RRAM and Hybrid DNN accelerators

Attracted by high energy efficiency, storage density, and parallelism, existing RRAM-based IMC accelerators have primarily focused on DNN inference, employing pre-trained models for single-time deployment [3, 7–9]. RRAM-based IMC is an attractive solution for DNN inference due to its high parallelism and dense storage. Fig. 1 illustrates the basic architecture of the 1T1R crossbar array, which enables efficient vector-matrix multiplication (VMM) through parallel analog computation along the column. In the RRAM-based IMC accelerator, the weights of the pre-trained DNN model are mapped as the conductance values, denoted as G, within RRAM cells. The input vector is represented by analog voltage pulses, referred to as V_{in} [4], which are applied through the horizontal select line (SL). The output of the vector-matrix multiplication (VMM) is obtained by computing the product current between the incoming voltage V_{in} and the programmed conductance G along the bit line (BL).

To explore the on-device learning, XMA[8] have been developed for multi-task adaptation. It learns new tasks offline, training column-wise masks while keeping the backbone model static. By enabling columns for new tasks or shifting partial accumulated results, it bypasses the need for expensive reprogramming and thus enhances energy efficiency. In a different approach, [9] proposes a method that, instead of calculating accurate gradients, determines only the sign of the gradient. During the on-chip training phase, this method involves array-level weight updates, where each RRAM device's resistance is gradually adjusted based on the gradient sign, without verification. Although this technique has shown effectiveness on small-scale data, its simplicity may not guarantee convergence when applied to training larger models.

In addition, a few hybrid DNN accelerators are proposed with different objectives. [10] develops a hybrid RRAM/SRAM IMC design for robust DNN acceleration. To mitigate the post-mapping accuracy loss in DNNs, it trains models on RRAM and SRAM IMC memories respectively. Then it sums the outputs from both sides to create an ensemble model with bit-level compensation. [11] proposes to accelerate DNN on-device training with non-volatile and volatile memory-based hybrid precision synapses.

2.2 Continual learning

Continual learning aims to continually learn multiple tasks that arrive in a sequential manner without forgetting prior learned knowledge. Plentiful continual learning methods have been developed in supervised learning with the main objective of mitigating cartographic forgetting when learning new tasks [12–14]. and can be generally divided into three categories: 1) *Regularization-based methods* (e.g., [15]) preserve knowledge from old tasks by incorporating an additional regularization term in the loss function. 2) *Structure-based methods* (e.g., [12, 16]) adapt model parameters or architectures sequentially as new tasks are introduced. 3) *Memory-based methods* can be further divided into memory-replay methods and orthogonal-projection based methods. Memory-replay methods (e.g., [17]) store and replay data from old tasks when learning new tasks. On the other hand, orthogonal-projection based methods (e.g., [13, 14]) update the model for each new task in a direction orthogonal to the subspace spanned by the inputs of old tasks.

Different from supervised learning, self-supervised learning is an unsupervised learning scheme aimed at learning visual representations without the need for costly data labeling. Recent advances [6] have demonstrated comparable or even superior performance compared to supervised representation learning. As a representative work, Barlow Twins [6] aims to minimize redundancy between components of embedding vectors while preserving maximum information. This is achieved by minimizing the cross-correlation matrix, computed between the outputs of two identical networks, to make it closer to the identity matrix. One advantage of BarlowTwins is that has no need for large batch size and negative samples compared to prior works. Thus, in this work, we adopt it as base learning method for self-supervised continual learning. More recently, several works [1, 18] have emerged to address the problem of self-supervised continual learning. These studies demonstrate that self-supervised continual learning can mitigate catastrophic forgetting and learn more general representations compared to supervised continual learning. Specifically, CaSSLe [18] and PFR [1], employ a temporal projection module to ensure that newly learned feature spaces preserve information from previous tasks. LUMP [19] adapts the Mixup technique [20] to interpolate data between the current task and instances from previous tasks. This interpolation is achieved by combining the current task data with samples from a replay buffer, reducing catastrophic forgetting. To improve the training efficiency of the self-supervised continual learning, [21] proposes layer-wise freezing based on task correlations to freeze partial layers for training each task.

3 METHODOLOGY

3.1 Overview of Hyb-Learn framework.

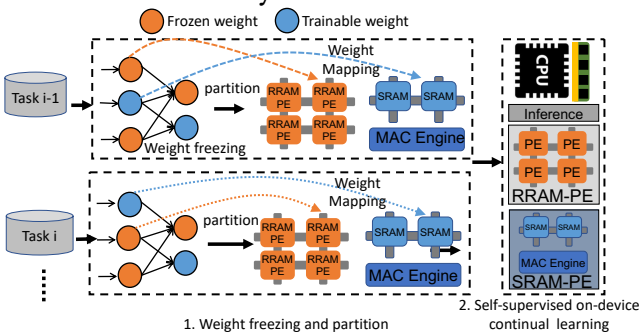

Figure 2: Overview of the framework.

In this work, we propose *Hyb-Learn*, a novel training framework to enable on-device continual learning with hybrid RRAM/SRAM IMC design. In practice, given new tasks ($\{\mathcal{T}_1, \mathcal{T}_2, ..., \mathcal{T}_N\}$) that arrive sequentially, *Hyb-Learn* aims to continually perform on-device training for each task in sequential order without forgetting prior learned knowledge. As shown in Fig. 2, the overflow of *Hyb-Learn* can be divided into two steps: 1) model freezing and partition via PE-wise task-correlation and 2) self-supervised learning where only trainable weights in SRAM are updated. In the following, we will dive into these two key steps of the proposed Hyb-Learn.

Model freezing and freezing via PE-wise task correlation. Given a new arriving task t_i, we first divide the DNN model weights into two groups in processing elements (PEs) granularity: trainable weights and frozen weights. The PE serves as the fundamental computing core in hybrid RRAM/SRAM architectures. To take advantage of the hybrid IMC design and avoid high-cost write operations in RRAM, we partition the frozen weights into RRAM PEs and the trainable weights into SRAM PEs. Furthermore, to determine the mapping partition to RRAM and SRAM, we propose *PE-wise task correlation* to partition weights in PE-wise. Specifically, the weights that are partitioned into RRAM (frozen) are highly correlated to the current task, while the low-correlated weights will be mapped into SRAM that needs to be updated for learning new task.

Continual learning via self-supervised learning. After partitioning certain weights into RRAM and SRAM PEs, we proceed with self-supervised learning to train the DNN model for each new task. Specifically, we employ the BarlowTwins [6] algorithm as the backbone self-supervised learning method to train the model. It is important to note that the weights located in the RRAM PEs are frozen and used only for forward propagation, while the weights in the SRAM PEs are updated during the training process which largely reduces the computing cost. In the following, we will introduce the detailed techniques of each component respectively.

3.2 Model freezing and patition via PE-wise task correlation

To maximize training efficiency with hardware and software co-design meanwhile mitigating catastrophic forgetting in continual learning, we propose **task-correlated PE-wise weight freezing** to freeze the large portion of weights that are "highly correlated" with prior currents and only fine-tune a small number of uncorrelated weights. The intuition of such design is that freezing "highly correlated" weights can preserve the prior learned knowledge, leading to mitigating catastrophic forgetting, meanwhile only updating the uncorrelated weights enable better learning capacity for current task. Specifically, we formally define the PE-wise correlation ratio between the current and prior tasks as:

$$r_i = \frac{\|\text{Proj}_{S_t^i}(\nabla \mathcal{L}_t(\boldsymbol{w}_{t-1}^i))\|_2}{\|\nabla \mathcal{L}_t(\boldsymbol{w}_{t-1}^i))\|_2} \tag{1}$$

where $\text{Proj}_{S_t^i}$ denotes the projection on the input subspace S_t^i of prior tasks $(1, 2, ..., t-1)$ on i^{th} PE, and $\boldsymbol{w}_{t-1}^l$ represents the i^{th} PE-wise weight in the model before learning task t. Here $\text{Proj}_S(A) = AB(B)'$ for some matrix A and B is the bases for S. Due to the fact that the gradient lies in the span of the input [22], if the task correlation ratio $r_l \in (0, 1)$ has a large value, it implies that the current task t and prior tasks may have sufficient common bases in i^{th} PE-wise weight between their input subspaces and hence are strongly correlated. Based on this, the proposed task-correlated PE-wise weight freezing method freezes partial PE-wise weights with the highest correlation ratios during new task learning to enhance the training computation and memory efficiency. After that, we apply a $TopK$ function according to the gradient projection norm magnitude in PE-wise to select the PEs to freeze.

$$F_t = \{i | P_t^i \in \text{TopK}(P_t, k)\} \tag{2}$$

where P_t denotes a set of global projection norms across all layers for current task t and P_t^i is the simplified denotation of

$$\|\text{Proj}_{S_t^i}(\nabla \mathcal{L}_t(\boldsymbol{w}_{t-1}^i))\|_2 \tag{3}$$

for i^{th} PE-wise weight. k is a pre-defined PE-wise freezing ratio (e.g., 0.5, 0.6, etc..) which denotes the global ratio of the number of frozen PE-wise weights. In practice, we use a fixed ratio that is shared by all tasks.

The proposed PE-wise freezing approach enables efficient training for each new task. However, as shown in Eq. (1), due to the varying task correlation between the current task and prior tasks, the indexes of the frozen PEs differ for different tasks. This means that after training the prior task, some of the PE-wise weights in RRAM become trainable weights for the current new task t_i, and vice versa for SRAM. To address this issue, we introduce a technique called *weight swapping* to partition weight in an efficient way. In practice, after performing the task-correlated weight correlation analysis as discussed above, we swap the newly determined trainable PE-wise weights in RRAM, for current task, with the frozen weights in SRAM. This swapping ensures that the weights identified as frozen in both prior task and current task remain in RRAM without incurring any additional reprogramming costs. By applying the weight swapping technique, we can minimize the cost of one-time weight re-partition before new task learning.

From the hardware perspective, the proposed PE-wise weight freezing naturally aligns with the hybrid RRAM/SRAM IMC design. For each new task data, the weights are partitioned into two groups: trainable weights and frozen weights, which can be efficiently mapped in the corresponding PEs. In practice, the frozen weights are allocated to RRAM PEs, while the trainable weights are stored in SRAM. During the training of each task, the weights in RRAM PEs are frozen, meaning they do not contribute to the weight

updates for backpropagation. This maximizes the computational capacity of the system and, importantly, eliminates the need for extensive and costly RRAM cell re-programming.

3.3 Self-supervised continual learning with pre-trained model

Although beneficial for training efficiency, weight freezing can often result in an accuracy degradation of learning new task. Motivated by the recent self-supervised continual learning works [1, 18, 21, 23], which shows that the self-supervised learning learns more general features that own higher correlations between tasks leading to less catastrophic forgetting. It implies that self-supervised continual learning has the potential to freeze more weights compared to supervised learning. Thus inspired, we employ the self-supervised learning algorithm to train the model for each task. In general, self-supervised continual learning does not require data labels during training. The objective is to learn a general representation invariant to augmentations on all tasks, which can be formulated as:

$$\min_{\boldsymbol{w}_f} \sum_{t=1}^{T} \sum_{i=1}^{N_t} \mathcal{L}_{SSL}(f(\boldsymbol{x}_{t,i}^1, \boldsymbol{x}_{t,i}^2)) \tag{4}$$

where $\boldsymbol{x}_{t,i}^1$ and $\boldsymbol{x}_{t,i}^2$ are augmented images generated from $\boldsymbol{x}_{t,i}$. More specifically, we utilize the BarlowTwins algorithm [6] which minimizes the redundancy between the embedding vector components while preserving the maximum information. It achieves this by minimizing a specific loss function that involves the cross-correlation matrix computed between the outputs of two identical networks.

Moreover, to maximize the freezing ratio of weight freezing, we initialize the weights using a pre-trained model. By doing so, we can enhance the PE-wise weight freezing ratio to reduce the computing cost meanwhile achieving better overall accuracy.

4 HYBRID RRAM/SRAM HARDWARE

Fig. 3 represents the overall architecture of the hybrid RRAM/SRAM system. Both RRAM-based PEs and SRAM-based PEs could process the required forward or backward computation along with input buffer to temporarily store the intermediate data.

Fig. 3(b) shows the RRAM-based IMC macro to perform the *MAC* (multiplication and accumulation) operation where the weight is stored in the RRAM devices as the conductance located at the intersection of horizontal worldlines (WLs) and vertical bitlines (BLs). The inputs (images or activation from the last layer) are fed to the SL in a bit-serial manner. Then the MAC operation is done based on Ohm's law at the analog domain where the 1-bit multiplication is done at the intersections based on $I = V \times G$. Next the BL accumulates all the currents together to realize the *ADD* operation. The ADC converts the analog current result back to the digital domain. Finally, the shift-and-add circuit works together with the bit-serial input scheme to generate the proper *MAC* result. The *MAC* result is temporally stored in the macro buffer and is waiting for post-process (concatenate/add with the result from other PEs to be able to send to the next module).

Fig. 3(c) shows the adopted SRAM PE design. For better reliability and to reduce the design complexity, we did not adopt the new rising in-SRAM computing circuits. Instead, we choose the typical design which has the SRAM array to store the correlated weights, and the digital PE units are placed close by to maximize the throughput and utilize the memory bandwidth. Each PE has the multiplier and adder to perform the *MAC* operation. Thus the PEs associated with the same SRAM array are able to run independently and in a parallel fashion. Note, the adopted SRAM PE is a general PE since it supports both multiplication and addition with arbitrary data from the SRAM array and input buffer. It could be used to replace RRAM PE for convolution and backpropagation, or used as extra computing resources to compensate the computing error due to RRAM PE[10, 11]. Here we treat both RRAM and SRAM-based PEs as the *MAC* engine, but with different properties: RRAM-based PE leverages the Non-Volatile Memory's zero leakage and the analog domain computation maximize the parallelism by turning all the devices on at the same time while the SRAM PE is more general and requires much less power to modify/update the stored data. To make data partition more efficient between SRAM PE and RRAM PE, we design both PEs hold the same amount of data. For the RRAM PE, each PE holds a 64×64 RRAM array where each RRAM cell can represent 4 different resistance levels. Such a 64×64 RRAM array is equivalent to 8Kb. Thus, we choose the SRAM array is the same as 8Kb, a SRAM PE can hold all data from a RRAM PE and process the same computation. If they both hold the same weight and fed the same input, the computation result is identical.

4.1 Training dataflow

Fig. 4 shows the dataflow during the backpropagation of the training process. Although only partial weights need to be updated, it still needs to backpropagate the loss with the transposed weight matrix for all layers. For the SRAM PE, transpose the weight can be easily done with a pre-defined access order. However, performing the *MAC* operation with the transposed weight in RRAM PE is not an easy job. Usually, extra accessing transistor, transposed WL/BL/SL decoders, ADC, and other peripherial circuits are necessary to design the tranposable RRAM PE. Not mentioning the extra effort for the routing. To simplify the design, we choose to use extra RRAM PE instead of designing the tranposable RRAM PE. For each 64×64 weights, we use two RRAM PEs to store the weight. One RRAM PE stores the weight as the matrix while the other RRAM PE stores the transposed weight matrix. Thus RRAM PEs will be used in forward and backward pass separately. Again, the weight stored in RRAM PEs are only involved in the activation/loss propagation. Those weights are frozen in RRAM arrays and will not be updated. Only the SRAM PE will be participated in the weight gradients calculation and weight update.

5 EXPERIMENT AND EVALUATION

5.1 Algorithm Experiment

We evaluate the algorithm performance of the proposed framework using ResNet-18 [24] architecture on split CIFAR-100. We follow the training and evaluation setup of [19] and learned models are evaluated with KNN classifier [25]. In addition, we initialize the model with the pre-trained weight by using ImageNet-1K dataset. Two metrics are used to evaluate the performance: *Accuracy*, the average final accuracy over all tasks, and *Forgetting*, which measures the forgetting of each task between its maximum accuracy and accuracy at the completion of training. Accuracy and Forgetting

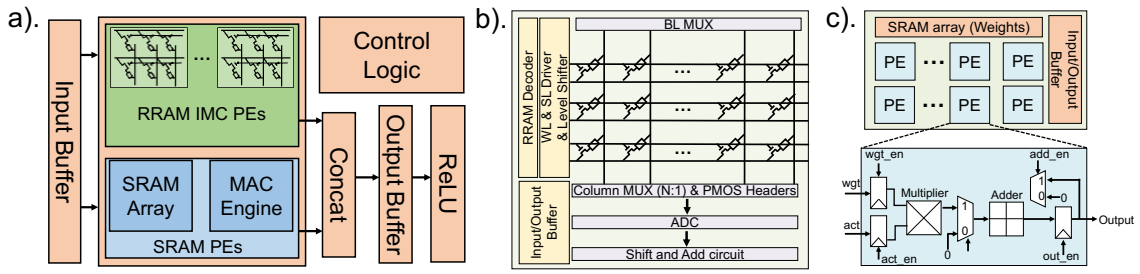

Figure 3: Overall architecture of the Hybrid RRAM and SRAM system: a) the hybrid RRAM/SRAM architecture; 2) RRAM-based IMC macro; 3) the architecture of the SRAM PE design.

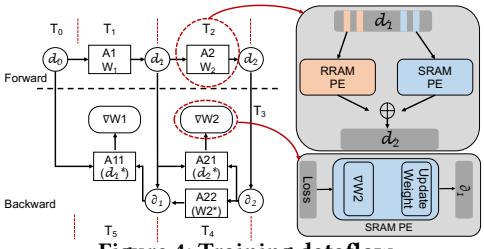

Figure 4: Training dataflow

are defined as:

$$Accuracy = \frac{1}{T}\sum_{i=1}^{T} A_{T,i} \tag{5}$$

$$Forgetting = \frac{1}{T-1}\sum_{i=1}^{T-1} \max_t (A_{T,i} - A_{i,i}) \tag{6}$$

where T is the number of tasks, $A_{T,i}$ is the accuracy of the model on i-th task after learning the T-th task sequentially.

Table 1 present the experimental results for accuracy and forgetting under different configurations. The "weight update" column indicates the ratio of non-frozen PE-wise weights, where "100% weight update" means re-training the **full weights** for each task. The terms "FP32" and "INT8" refer to the model weights represented by 32-bit floating points and 8-bit integers, respectively. First, the INT8 achieves the same accuracy as FP32 for all different weight update ratios. To demonstrate the effectiveness of using a pre-trained model for initialization, we compare the results w/wo a pre-trained backbone model for the case of 100% weight update. As shown in Table 1, utilizing a pre-trained model significantly increases the accuracy by 1.1%. Moreover, even updating only 50% of the weights, the accuracy of the pre-trained model surpasses that of the non pre-trained model with 100% weight update. When only updating 10% of weight, we can still obtain 76.79% accuracy which demonstrates that the prosed PE-wise freezing preserves the "important" weight for each task. Furthermore, we conduct an ablation study to evaluate the impact of the weight update ratio under 5 configurations (i.e., 100%, 60%, 50%, 20%, and 10%). It can be observed that as the weight update ratio decreases, the accuracy gradually decreases. However, the forgetting also decreases, indicating a reduction in catastrophic forgetting. This is attributed to a larger proportion of fixed weights when using a smaller weight update ratio, which helps mitigate the occurrence of catastrophic forgetting.

5.2 Hardware Evaluation

There are three different layers in the ResNet-18 based backbone models: 3×3 Conv2D, 1×1 Conv2D as residual connection, and Linear layers. All those layers' in/out channels are the power of 2 (32, 64, 128, 256, etc). To maximally utilize the internal memory bandwidth, layer weights are partitioned alone the in/out channel-wise. Both residual layers and linear layers could be easily mapped on the 64 × 64 RRAM array by dividing the in/out channels by 64. For the 3×3 Conv2D kernel, a 64 × 64 RRAM array can map a 3×3×7×64 portion which equals to 63×64 and leave one row spare for defect rescuing[27]. The hardware performance of different algorithms is evaluated based on the circuit-level simulator NeuroSim [28]. The quantized targeted DNNs are characterized by the 2-bit per cell HfO_2 1T1R RRAM devices, the RRAM array characteristics are adopted from[8]. Each RRAM 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. 5(a) shows the normalized energy cost of weight update during on-device learning. The RRAM only means the conventional RRAM-based IMC scheme that all the weights are stored and updated in the RRAM array during training. Other bars show the normalized weight update energy saving for our hyb-learn framework with different percentages of SRAM PEs for on-device learning. It could be seen that the smaller ratio of SRAM PE indicates larger portion of weights mapped to RRAM PEs are frozen, thus more energy saving during training. For example, with 10% SRAM PE ratio, 90% of weights are frozen during training, achieving around 13.6× energy cost reduction. Fig. 5(b) shows the total area usage while 50% weights are stored in SRAM PEs and 50% weights are stored in RRAM PEs.

Table 1: Accuracy/forgetting of the learned model on Split CIFAR-100

Pre-train?	N/A	With Pre-trained backbone				
Weight Update	100%	100%	60%	50%	20%	10%
FP32	79.03/2.29	81.4/2.52	81.19/2.17	80.64/1.8	77.69/1.26	76.79/1.54
INT8	79.0/2.23	81.38/2.54	81.13/2.15	80.63/1.8	77.6/1.41	76.65/1.68

Table 2: Comparison with other IMC Platforms

Reference	Ours	TIME[26]	PipeLayer[3]	Science[9]
Device	RRAM+SRAM	RRAM	RRAM	RRAM
RRAM Structure	1T1R	1T1R	1T1R	2T2R + 1T1R
Learning Capability	Y	Y	Y	Y
Weight update	backpropagation	backpropagation	backpropagation	signed based weight update
Learning Scheme	Only Update SRAM Weights	Re-Program RRAM	Re-Program RRAM	Re-Program RRAM
RRAM write freq. per iter.	0	3	3	1
Normalized Training Eng.	1 (10% SRAM) 5 (50% SRAM)	40.8	40.8	13.6
Normalized Inference Eng.	1.3 (10% SRAM) 2.5 (50% SRAM)	1	1	1
Normalized Area	1.2 (10% SRAM) 2 (50% SRAM)	1	1	1.5

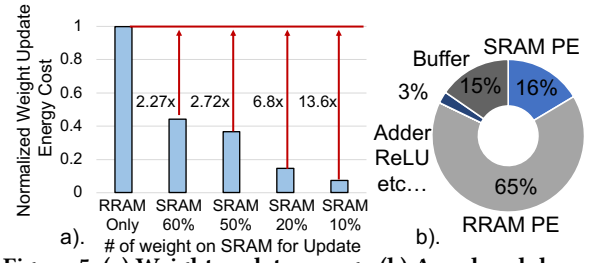

Figure 5: (a) Weight update energy (b) Area breakdown.

Table 2 presents a comparison of our proposed hybrid RRAM- and SRAM- IMC design with the state-of-the-art IMC designs that facilitate on-device learning. The training schemes employed in these state-of-the-art IMC designs require extensive reprogramming of RRAM devices, posing significant challenges in terms of both the endurance of RRAM devices and the high energy consumption associated with training.

The TIME[26] and PipeLayer[3] projects suggest conducting both forward and backward operations within a 1T1R crossbar array. However, this approach requires three RRAM write operations for each weight update due to the necessary transpose operation. These operations include writing transposed weights for error/loss propagation, writing transposed error/loss to calculate the gradient, and updating the weight. As RRAM devices typically consume much more energy during writing compared to SRAM, this RRAM-based on-device learning approach tends to be the most energy-intensive for training. A recent advancement is highlighted in a Science paper[9], which introduced the novel STELLAR learning scheme. This method diverges from traditional backpropagation-based learning by using the sign of the input, output, and error to determine the update direction for each RRAM device. A specially designed CMOS weight updating circuit facilitates this calculation. By eliminating the need to transpose weights and errors, this scheme reduces RRAM write operations to once per iteration, significantly lowering power consumption during training. However, while this modified learning scheme has shown promise in smaller datasets like MNIST, its effectiveness and convergence for training larger models and datasets have not yet been conclusively established. Finally, our proposed One-time re-map and learning on SRAM method, not only mitigate the endurance demand of RRAM but also leverages the frequent weight update on the low cost SRAM makes the on-device learning easier to implement.

6 CONCLUSION

In this work, we present Hyb-learn, an innovative framework developed for on-device continual learning, utilizing a hybrid RRAM and SRAM IMC architecture. With the arrival of sequential tasks, Hyb-learn features a unique hybrid design that smoothly integrates model partitioning and self-supervised learning initialization, realized through a strategic co-design of both software and hardware. Our experimental results highlight that the Hyb-learn framework not only achieves high accuracy but also effectively minimizes forgetting, even with 50% of the weights being frozen. Additionally, we offer a detailed analysis of the framework's energy efficiency and a comprehensive breakdown of the hardware area, underscoring the practicality and effectiveness of Hyb-learn in on-device learning applications.

ACKNOWLEDGMENT

This work is supported in part by the National Science Foundation under Grant No.2314591, No.2414603, No.2349802, No.2342726, and No.2348376.

REFERENCES

[1] A. Gomez-Villa et al. Continually learning self-supervised representations with projected functional regularization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 3867–3877, 2022.
[2] A. Shafiee et al. Isaac: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars. In *ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, pp. 14–26.
[3] L. Song et al. Pipelayer: A pipelined reram-based accelerator for deep learning. In *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 541–552, 2017.
[4] 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)*, pp. 1–6, 2016.
[5] F. Zhang et al. Xst: A crossbar column-wise sparse training for efficient continual learning. In *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 48–51. IEEE, 2022.
[6] J. Zbontar et al. Barlow twins: Self-supervised learning via redundancy reduction. In *International Conference on Machine Learning*, pp. 12310–12320. PMLR, 2021.
[7] S. Mittal. A survey of reram-based architectures for processing-in-memory and neural networks. *Machine Learning and Knowledge Extraction*, 1(1):75–114, 2019.
[8] F. Zhang et al. Xma: A crossbar-aware multi-task adaption framework via shift-based mask learning method. In *2022 DAC*, 2022.
[9] W. Zhang et al. Edge learning using a fully integrated neuro-inspired memristor chip. *Science*, 381(6663):1205–1211, 2023.
[10] G. Krishnan et al. Hybrid rram/sram in-memory computing for robust dnn acceleration. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 41(11):4241–4252, 2022.
[11] Y. Luo et al. Accelerating deep neural network in-situ training with non-volatile and volatile memory based hybrid precision synapses. *IEEE Transactions on Computers*, 69(8):1113–1127, 2020.
[12] L. Yang et al. Grown: Grow only when necessary for continual learning. *arXiv preprint arXiv:2110.00908*, 2021.
[13] S. Lin et al. Trgp: Trust region gradient projection for continual learning. *arXiv preprint arXiv:2202.02931*, 2022.
[14] S. Lin et al. Beyond not-forgetting: Continual learning with backward knowledge transfer. *arXiv preprint arXiv:2211.00789*, 2022.
[15] J. Kirkpatrick et al. Overcoming catastrophic forgetting in neural networks. volume 114, pp. 3521–3526. National Acad Sciences, 2017.
[16] J. Yoon et al. Lifelong learning with dynamically expandable networks. *arXiv preprint arXiv:1708.01547*, 2017.
[17] A. Chaudhry et al. Efficient lifelong learning with a-gem. *arXiv preprint arXiv:1812.00420*, 2018.
[18] E. Fini et al. Self-supervised models are continual learners. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 9621–9630.
[19] D. Madaan et al. Representational continuity for unsupervised continual learning. In *International Conference on Learning Representations*, 2021.
[20] H. Zhang et al. mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412*, 2017.
[21] L. Yang et al. Efficient self-supervised continual learning with progressive task-correlated layer freezing. *arXiv preprint arXiv:2303.07477*, 2023.
[22] G. Saha et al. Gradient projection memory for continual learning. *arXiv preprint arXiv:2103.09762*, 2021.
[23] D. Hu et al. How well does self-supervised pre-training perform with streaming data? In *International Conference on Learning Representations*, 2022.
[24] K. He et al. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.
[25] Z. Wu et al. Unsupervised feature learning via non-parametric instance discrimination. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3733–3742, 2018.
[26] M. Cheng et al. Time: A training-in-memory architecture for rram-based deep neural networks. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 38(5):834–847, 2019.
[27] F. Zhang et al. Defects mitigation in resistive crossbars for analog vector matrix multiplication. In *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 187–192, 2020.
[28] 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)*, pp. 32.5.1–32.5.4, 2019.