Efficient Memory Integration: MRAM-SRAM Hybrid Accelerator for Sparse On-Device Learning

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

Amitesh Sridharan
asridha7@jh.edu
Johns Hopkins University
Baltimore, Maryland, USA

Wilman Tsai
wtsaia@stanford.edu
Stanford University
Stanford, California, USA

Yiran Chen
yiran.chen@duke.edu
Duke University
Durham, North Carolina, USA

Shan X. Wang
sxwang@stanford.edu
Stanford University
Stanford, California, USA

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

ABSTRACT

With the prosperous development of Deep Neural Network (DNNs), numerous Process-In-Memory (PIM) designs have emerged to accelerate DNN models with exceptional throughput and energy-efficiency. PIM accelerators based on Non-Volatile Memory (NVM) or volatile memory offer distinct advantages for computational efficiency and performance. NVM based PIM accelerators, demonstrated success in DNN inference, face limitations in on-device learning due to high write energy, latency, and instability. Conversely, fast volatile memories, like SRAM, offer rapid read/write operations for DNN training, but suffer from significant leakage currents and large memory footprints. In this paper, for the first time, we present a fully-digital sparse processing in hybrid NVM-SRAM design, synergistically combines the strengths of NVM and SRAM, tailored for on-device continual learning. Our designed NVM and SRAM based PIM circuit macros could support both storage and processing of N:M structured sparsity pattern, significantly improving the storage and computing efficiency. Exhaustive experiments demonstrate that our hybrid system effectively reduces area and power consumption while maintaining high accuracy, offering a scalable and versatile solution for on-device continual learning.

1 INTRODUCTION

The rapid advancement of DNNs in various domains has necessitated the development of specialized hardware accelerators to manage their substantial computational demands. Among these, NVM based PIM accelerators have gained prominence, especially for DNN inference. The energy efficiency and high-density storage of NVM like Resistive Random Access Memory (RRAM)[1–4] and Spin-Transfer Torque Magnetic RAM (STT-MRAM)[5–10] have enabled the creation of compact, power-efficient memory devices. However, leveraging these technologies for DNN training encounters significant challenges due to their inherent characteristics.

The core challenges with NVM in DNN training is its high write energy, latency, and unstability[4, 11, 12]. Training DNNs involves frequent updates to the weight memory, which are highly write-intensive operations. NVMs, optimized for read-intensive tasks, incur higher energy consumption and slower write speed compared to fast volatile memories, which becomes a bottleneck in training where speed and efficiency of weight updates are crucial. Additionally, DNN training requires the support of transpose matrix operations during backpropagation. Given the memory's write limitations, designing NVM-based accelerators to efficiently manage these operations also brings higher architectural complexity. Moreover, the endurance of certain types of NVMs, like RRAM, where each cell can sustain a finite number of write operations, becomes a critical concern due to the frequent weight updates in the training process. In contrast, volatile memory technologies, like SRAM, excel in scenarios requiring rapid operation and frequent data rewriting. Their low latency and fast write capabilities align well with the iterative nature of DNN training algorithms. However, these benefits are offset by issues such as significant leakage currents leading to higher power consumption, and larger memory footprint, which poses challenges in scalability and integration[13–15].

To address these challenges, in this work, for the first time, we present a fully-digital hybrid sparse PIM architecture that combines the strengths of both NVM and SRAM technologies, tailored for both DNN sparse inference and training for on-device continual learning. To further improve efficiency, we design both the NVM and SRAM based PIM circuit macros to support storage and processing with popular N:M structured sparse weight encoded with compressed sparse column (CSC) format. Sparsity in neural networks, where a significant portion of the weights are zeros, offers a great reduction in computational and storage requirements.

Our proposed hybrid architecture is effective and demonstrated for the state-of-the-art on-device continual learning application, where we exploit the high density and low leakage benefits of NVM based PIM processing element (PE) for storing and processing the offline-learned backbone network with fundamental features, where such basic model weight parameters are frozen during new downstream task learning. While, the SRAM based PIM PEs are used to implement small amount of residual adaptors for learning the new data features, leveraging its fast and efficient write capabilities. To facilitate smoother integration and enhance scalability, both the SRAM and NVM-based PEs have been implemented in the digital domain. Systematic and exhaustive experiments are conducted

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

ACM ISBN 979-8-4007-0601-1/24/06…$15.00
https://doi.org/10.1145/3649329.3657390

using a MRAM-SRAM prototype design. The results clearly demonstrate that our novel hybrid design, combined with the strategic mapping method and N:M sparse processing, significantly reduces both area and power consumption for on-device continual learning.

2 BACKGROUND & RELATED WORKS

2.1 Processing-In-Memory

2.1.1 Digital SRAM-based PIM. SRAM based PIM has been lauded as a promising approach to accelerate small scale compute intensive applications. SRAM being mature CMOS technology has popularized digital In-Memory compute and has shown to scale well in smaller nodes [14, 15]. There are analog counter parts to SRAM based compute but they incur a significant accuracy drop due to Analoge to Digital Converter (ADC) noise and huge power from the high-precision ADCs. Digital on the other hand can support high precision using off-the-shelf adder trees, but adder trees still dominate the area when compared to the bit-cell area. To reduce the adder tree cost per bit-cell, other works tend to time-multiplex the memory with the compute hardware to amortize the cost. But this just ends up scaling upon memory w.r.t. compute hardware. Since sparsity is a key component in modern DNNs, we take a different approach of processing the sparse-encoded weights in memory to improve storage density and time-multiplex sparsity, i.e. time-share the compute hardware w.r.t. to compressed weights similar to [13, 16]. But unlike supporting a proprietary compression format, we design to use a popular compression format called *Compressed sparse column (CSC)* for high adaptability.

2.1.2 MRAM-based PIM. Magnetoresistive random-access memory (MRAM) is an emerging NVM, where the basic memory cell device is Magnetic Tunnel Junction (MTJ). MTJ typically consists of multiple layers where two ferromagnetic layers sandwich a thin insulating layer. One of the two ferromagnetic layers is a permanent magnet with a particular polarity called 'fixed layer'. Another ferromagnetic layer's magnetization can be changed by external stimulus called 'free layer'. When the free layer's magnetization toward opposite to the fixed layer, this MTJ is in Anti-Parallel State (AP), making it have high electrical resistance. On the contrary, when the free layer's magnetization has the same polarity as the fixed layer. This MTJ is in Parallel State (P), with a relatively low electrical resistance.

Unlike other popular NVMs, like RRAM based PIM designs those are often analog, leveraging RRAM's capability to handle multiple resistance levels[1–4], MRAM-based PIM designs are predominantly digital due to MRAM's inherent binary nature (AP/P status). In this work, we mainly use MRAM as the major NVM technology for the purpose of all digital PIM design. Existing MRAM-based PIM accelerators primarily focus on DNN inference using one time deployment of pre-trained models [5–10]. Computation predominantly occurs in or near memory array, employing modified sense amplifiers for executing bit-wise operations essential to neural network processing[5, 6]. Additionally, more complex calculations are handled by dedicated circuitry such as adder trees, shift registers, and accumulators, located close to MRAM arrays to leverage the benefits of in/near-memory processing[7–10]. This architecture aims to minimize the energy and latency costs associated with data movement in traditional computing systems, making it a promising approach for efficient neural network hardware.

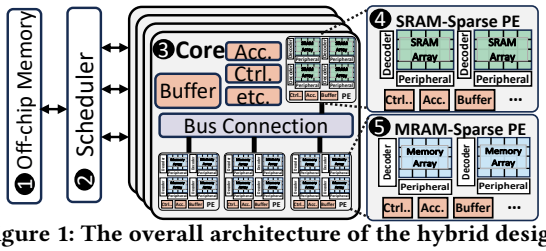

Figure 1: The overall architecture of the hybrid design.

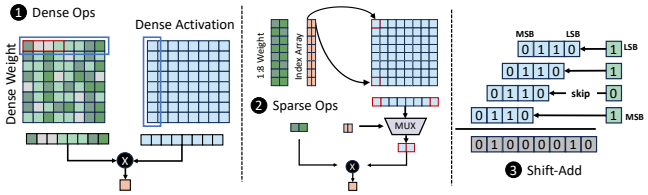

Figure 2: N:M Sparse Matrix Multiplication

2.2 Hybrid PIM design

[17] proposes a hybrid design to accelerate DNN on-device training with non-volatile and volatile memory-based hybrid precision synapses, where the RRAM and capacitor are combined to leverage the non-volatility and large dynamic range of RRAM, together with the symmetric charging/discharging behavior of capacitor. Another hybrid RRAM/SRAM approach is to partition weights bit precision where the RRAM is for MSBs and the SRAM/capacitor is for LSBs. [18] develops a hybrid RRAM/SRAM PIM design for robust DNN acceleration. To mitigate the post-mapping accuracy loss in DNNs, it trains models on RRAM and SRAM PIM memories respectively, and then sums the outputs from both sides to create an ensemble model with bit-level compensation. Both works focus on using hybrid RRAM/SRAM to partition weights bit precision where the RRAM is for MSBs, while SRAM/capacitor are for LSBs. Differentiating from prior hybrid works, our approach has novel circuit designs for all-digital NVM and SRAM based sparse-PIM circuit macros to directly store and process sparse-encoded weight parameters and further supporting on-device continual learning.

2.3 N:M Structured Sparse Neural Network

Pruning weights from a neural network is one of the most popular strategies to improve both memory and compute efficiency. Unlike unstructured pruning, structured pruning is favored for its efficient memory access and simplicity in hardware implementation. A recent concept of N:M structured sparsity [19] is proposed, where at most N out of every M (contiguous, aligned) elements are non-zero, promising enhanced computational efficiency with minimal accuracy loss. As exemplified by NVIDIA, its Ampere GPU architecture introduces a 2:4 sparsity pattern in its Tensor Cores[20], effectively reducing the required multiply-and-add operations by half and potentially doubling performance over equivalent dense matrix multiplications. Recognizing the current absence of PIM hardware designed to capitalize on N:M structured sparsity, our work aims to bridge this gap through designing sparse PIM circuits.

3 MRAM-SRAM HYBRID PIM DESIGN

Figure 1 illustrates our hybrid MRAM-SRAM based sparse PIM architecture. It integrates off-chip memory, a scheduler, and a cluster of hybrid MRAM/SRAM cores interconnected by buses. The off-chip

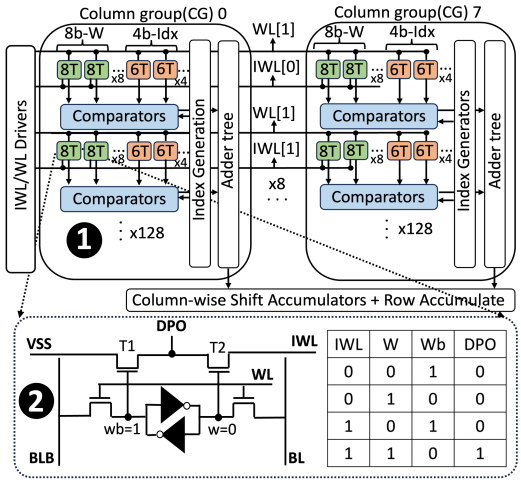

Figure 3: SRAM based sparse PE design

memory, marked as 1, is responsible for storing local data and parameters. The scheduler, denoted by 2, manages data distribution and orchestrates execution in a Single-Instruction-Multiple-Thread (SIMT) manner, maximizing hardware parallelism. Each core, labeled 3, is equipped with a data buffer, control units (Ctrl.), shared accumulators (Acc.), and multiple processing engines (PE) connected via a bus, specialized for sparse matrix operations. The data buffer facilitates pipelined execution in a row-stationary approach [21], temporarily buffering input and output activations. Note that, the computation is fully digital and completely supported by the compute logic. As discussed earlier, this hybrid architecture could be adapted to different NVM technologies, like MRAM or RRAM. Here in this work, we use MRAM as a digital NVM case study to evaluate its performance and conduct comprehensive analysis.

To harness the advantages of different memory technologies, two types of sparse PEs are designed: 4 the SRAM sparse PE and 5 the MRAM sparse PE. Both are capable of storing and processing N:M sparse matrix multiplication within PE. As shown in Fig. 2 2, during sparse multiplication, both the sparse non-zero weight and its corresponding index are involved to only process non-zero operands, rather than whole matrix multiplication as in traditional dense approach (Fig. 2 1), to reduce complexity. The follow up shift-add operations are the same as demonstrated in Fig. 2 3. The detailed circuit designs SRAM and MRAM sparse PEs are explained respectively below:

3.1 Sparse Matrix Multiplications in SRAM PE

The architecture and circuits of a fully digital bit-serial SRAM based sparse PE is shown in Fig. 3 1. Each SRAM PE is sized to be 128x96. Out of which, 128x8x8 is for weight storage and 128x8x4 is for index storage, to support 8bit (i.e., INT8) weight resolution and 4 bit index range for up to N:16 structured sparsity pattern as will be explained in the mapping section. Each 128x12 column group consists of an index generator and an adder tree, whereas each 12b Weight index pair is padded with a comparator for decoding the compressed weights. All the column groups in the PE deposit onto a shift accumulator for input precision compensation and a row-wise accumulator for edge cases of uneven sparsity. Fig. 3 2 shows the bit-cell circuit schematic. The source terminal of T1 is always grounded and the source of T2 is the row-wise shared input word line (IWL). The drain is shared across T1 and T2. Depending on the weight stored in the bit-cell, either the T1 or T2 is active. Thus, they perform a pass-gate based static AND operation against weight and the IWL, serving as the 1-bit in-memory partial product compute for digital multiplication.

Prior PIMs rely on the structure of the matrix to perform matrix multiplications in-memory [14, 15, 22], which does not work for sparse processing since compression breaks the matrix structure. Compressed Sparse Column (CSC) compresses a matrix along the column direction, there by preserving the column structure (multiplications) but breaking the row structure (accumulations). Similarly, Compressed Sparse Row (CSR) works the other way around. Digital PIMs usually perform multiplications through a fixed shared row-wise word lines and accumulations through column-wise adder trees. So prior to mapping, we need to determine the ideal compression structure to first map to the memory array. CSR as mentioned before breaks multiplications, column-wise matrix indices are used to encode and compress CSR. But breaking multiplications will require input reordering and an additional buffer to accumulate and write-back every cycle. Whereas CSC breaks accumulations only, in this work, we choose to gate accumulations of particular indices to enable sparse matrix multiplication using CSC. Fig. 4 describes how a sparse weight matrix is CSC compressed and mapped to the PIM design. As the term CSC suggests the sparse weight matrix is compressed in the column-wise direction. For sparse processing, we get two weight matrices namely the compressed weight matrix and the corresponding index matrix pair.

The steps to perform CSC sparse matrix multiplication as follows:

- Step 1: Parallel in-memory dot-products.
- Step 2: Index generation and compare.
- Step 3: Accumulate selected indices and shit accumulate for input precision compensation.

First, the inputs (i.e. activations) are streamed and applied on the input word lines of every row in bit-serial. The 8T bit-cells perform parallel AND operations as described earlier with the shared input. Next, the column-wise index generators generate column specific indices in each cycle and send it to the comparators. The comparators compare this index against the index stored in the 6T bitcells adjacent to the 8T bit-cells. If the comparison is successful, then the results from step-1 are sent to the adder-trees for accumulation. These comparisons happen in all 128 rows of a column group for 1 index in parallel, and also across 8 columns for 8 unique indices also in parallel. In step-3, the selected indices are sent to the adder tree for accumulation followed by the shift accumulator to compensate the cycle-wise bit-serial input precision. A corner case to CSC is that one column might be more sparse than another column. In that case, when mapping to PIM, one index might span several columns rather than being confined to a single column. So we need a row-wise accumulator as shown in Fig. 3 to satisfy this scenario.

3.2 Sparse Matrix Multiplications in MRAM PE

Figure 5 details the architecture and circuits of the MRAM sparse PE. In this design, it mainly leverages the concept of near-memory processing that MRAM array serves mainly as NVM to store the

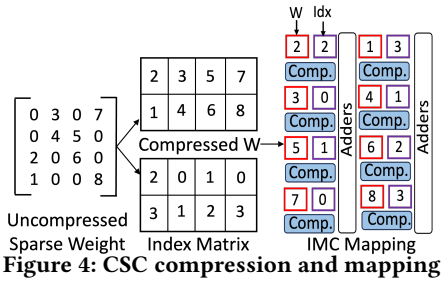

Figure 4: CSC compression and mapping

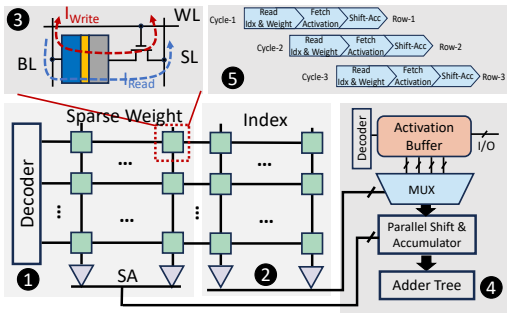

Figure 5: MRAM based sparse PE design.

sparse encoded weight and its indexes and all the required MAC operations are implemented through its peripheral digital circuits. As illustrated in Fig. 5 ❷, the MRAM array is divided into weight section and index section similar as our previous SRAM array structure. Upon receiving addresses from the control unit, the decoder activates the correspond memory row to retrieve weight values and sends to the parallel shift-and-accumulator for processing as the compressed matrix operand. Simultaneously, the corresponding index values are read and sent to the multiplexer (MUX). As shown in Fig. 5 ❹, this MUX interfaces with the activation buffer to select the appropriate activation data, which are then paired with the sparse weights in the parallel shift-and-accumulator to only process the stored non-zero weights. To maximize throughput, the PE operates in a pipeline fashion, as depicted in Fig. 5 ❺. In this setup, fetching the weight and index, selecting the corresponding activation, and performing parallel shift-and-accumulation are designed in three pipleline stages. Once accumulation is complete, the element-wise multiplication results are aggregated by the adder. The PE output will be transmitted to other PEs via a shared bus, facilitating systolic-array-like dataflow.

4 ALGORITHM DESIGN AND DATA MAPPING

In this work, we mainly focus on on-device multi-task continual learning setup, where assuming a pre-trained backbone model is available to be mapped to MRAM PE and a new task adaptor mapped to SRAM PE will be learned on-chip for new downstream task data. Figure 6 ❶ presents a state-of-the-art (SOTA) efficient multi-task continual learning structure, Rep-Net [23], which we will implement for this work. It includes a fixed main branch (i.e., the backbone model) and a tiny, parallel *reprogramming network* (Rep-Net) path. These two pathways exchange intermediate feature maps via an activation connector, allowing for mutual enhancement and improved performance in learning new tasks. The working principle is that the backbone model provides basic and general features, while the parallel, tiny Rep-Net path focuses on assimilating new knowledge from local data. By tweaking intermediate activations and training a new classifier for each task, this architecture effectively tackles new tasks with minimal weight updates (e.g., 5% of total size reported in [23]), keeping the majority of weights fixed.

To minimize memory overhead and facilitate efficient on-device continual learning, the backbone model remains fixed (both in terms of weights and architecture), as highlighted in Fig. 6 ❶. This part is mapped onto our MRAM PEs, taking advantage of the non-volatile memory's characteristics. Conversely, only the Rep-Net path and the shared final classification layer (marked in orange) learn new task data, which will be mapped to SRAM PEs, leveraging SRAM's rapid operation speed and ease of reprogramming. Compared to the backbone model, the Rep-Net path is very compact, consisting of just several convolutional modules and requiring minimal additional memory (for activation storage and extra parameters).

To support the on-device learning for the Rep-Net modules implemented in SRAM PEs, those below computations need to be supported by the system for backpropagation:

$$\text{Error propagation}: \ e^{l-1} = (W^l)^T \times e^l \tag{1}$$

$$\text{Gradient}: \ g^l = a^l \times (e^l)^T \tag{2}$$

$$\text{Weight Update}: \ W^l_{new} = W^l_{old} - g^l \tag{3}$$

Where,a, W, e, and g represent activation, weight, error, and gradient, respectively. The upper script 'l' represents the $l-th$ layer. It can be seen that the major computation is still matrix multiplication that is discussed earlier, while the key missing parts are the transposed matrix of W^l and e^l. In our system, as shown in Fig. 6 ❷, we adopt the design of *transposed SRAM PE buffer*. During backpropogration for Rep-Net, the current layer trained weights and generated error are transposed and written into such transposed SRAM PEs for error propagation and gradient calculation using the previously discussed in-memory matrix multiplication.

Note that, since the error and gradients are calculated layer by layer, the number of such transposed SRAM PE should be optimized depending on the system parallelism requirement and upper bounded by the maximum size of learned parameters for each layer. Due to the small percentage of learned parameters in Rep-Net structure and our N:M structured sparse processing hardware, the number of such transposed SRAM PE buffers is limited and depending on the model sparsity level, which will be discussed in our later evaluation section.

5 EXPERIMENT AND ANALYSIS

5.1 Algorithm Evaluation

As discussed in the SRAM and MRAM PE circuits, our system mainly support INT8 digital bit-serial N:M structured sparse processing, to assess the performance in continual learning, we conducted extensive experiments using five popular downstream datasets including Flowers [24], Food[25], Pets[26], Cifar10 and Cifar100. For all the experiments, we report the accuracy of new task learning with different sparsity and precision, as shown in Table 1. Our experiments utilize an ImageNet pre-trained ResNet-50 as the backbone fixed main branch and assign 6 learnable Rep-net modules, each consisting of 1 pooling layer and 2 convolution layers where one of the convolution kernel is 1×1.

The 'backbone@imagenet' column indicates the backbone model accuracy. We only performed INT8 Post-Training Quantization

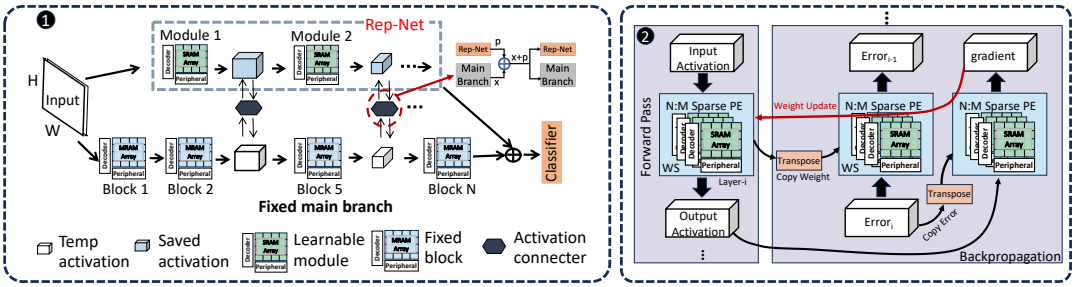

Figure 6: The fixed main branch is mapped to MRAM PE while the parallel tiny Rep-Net branch is mapped to SRAM PE to provide the learning capability.

Table 1: Accuracy Evaluation Result

Configure	Precision	Backbone@imagenet	flower102	Pets	Food101	Cifar10	Cifar100
Dense RepNet[23]	FP32	76.14%	96.1 %	91.8 %	80.5 %	95.9 %	81.9 %
Sparse RepNet (1:8)	FP32	71.34%	94.94%	90.27%	80.47%	94.28%	78.73%
	INT8	71.11%	94.56%	88.71%	78.14%	91.63%	77.32%
Sparse RepNet (1:4)	FP32	74.61%	95.24%	91.31%	82.02%	95.78%	80.38%
	INT8	74.62%	95.16%	90.33%	79.57%	92.21%	80.06%

(PTQ) and applied the N:M sparsity pattern, to assess their impacts on the backbone model. The results show that the higher sparsity level leads to larger accuracy drop. For example, the 1:4 (75%) sparsity only degrades around 1.5% accuracy, but the 1:8 (87.5%) sparsity significantly drops the accuracy more than 5%. The INT8 model can maintain the accuracy at the same level of its FP32 version. In the subsequent phase of our study, we examined the learning capabilities of these models on new downstream tasks by only updating the Rep-Net path modules and the final shared classifier as discussed earlier. To implement an accurate N:M sparsity pattern, we initially conducted a one-epoch gradient calculation across all weights on the RepNet path to identify the most crucial N weights among every consecutive M weights, based on magnitude. This was followed by a 30-epoch fine-tuning phase to learn the sparse weights. As can be seen in Table 1, even with 1:8 sparsity and INT8, the accuracy on the transfer dataset is still close to its baseline (i.e., FP32 dense model) with less than 2% accuracy drop. As anticipated, the 1:4 sparsity configuration achieves much smaller accuracy drop with less sparsity level. Notably, the 1:4 model exhibits superior performance on the Food101 dataset. This is attributed to the smaller size of Food101, which features only 750 training images and 250 testing images per class. This smaller dataset size likely contributes to the overfitting observed in the dense model.

5.2 Hardware Evaluation

To establish a comprehensive cross-layer device-architecture framework for system evaluation and comparison, we developed an in-house evaluation framework utilizing advanced array-level circuit modeling techniques. This framework incorporates the use of PIMA-SIM[27], NVSIM[28], and the TSMC 28nm Product Development Kit (PDK), enabling us to analyze and compare various aspects of our proposed hybrid PIM design thoroughly. The framework includes the development and extraction of SPICE-compatible STT-MRAM device models, allowing for detailed circuit-level performance analysis of sub-array circuits, and architecture-level performance estimation for peripheral circuits. At the circuit level, we constructed the memory sub-array with peripheral circuits, simulated using Cadence Spectre with the 28nm TSMC PDK library. This setup provides us with an accurate assessment of the circuit-level performance, crucial for the efficiency and feasibility of the proposed memory sub-array designs. To evaluate the timing, energy, and area of different memory technologies at the architecture level, we utilize PIMA-SIM and NVSIM. These tools offer flexibility in memory configuration, enabling the organization of banks, mats, and subarrays, as well as the design of peripheral circuitry.

Table 2 presents the hardware specifications for both SRAM and MRAM PEs. For the SRAM PE, Cadence Virtuoso was employed for designing the bit-cell layout, and HSPICE are used for evaluating the memory array along with other memory peripherals. The digital peripherals are designed in RTL and synthesized using Synopsys Design Compiler, followed by implementation in APR. Post-layout power and latency evaluations were conducted using PrimeTime and PrimePower. The reported area for the SRAM PE pertains to one 128x96 PIM array with 8 128-input, 8-bit adder trees, and the index decoder includes 128x8 comparators and index generators. Regarding the MRAM PE, we construct a 1024 x 512 memory sub-array with all required peripheral circuits, such as parallel shift-and-accumulators, decoders, and adder trees, which are also simulated in Cadence Spectre using the 28nm TSMC PDK library.

From the architecture level, each core contains 4×4 banks, with each bank comprising 4×4 MRAM sub-arrays as PEs. Given that the weights in the Rep-Net path are approximately 5% of the backbone model, we proportionately reserve the required storage in a fixed number of SRAM PEs. To accommodate the around 26MB storage requirement of the dense RepNet model, we adopt a dual-core configuration in our evaluations for the non-sparse supported SRAM/MRAM designs, as a single core could only store 16MB.

We adopt the prior fully digital in-SRAM computing design ISSCC'21[29] and fully digital MRAM-based design ISCAS'23[30] as baselines to evaluate our hybrid sparse design. Since [29] and [30] do not support sparse encoding, we mapped the entire model onto them without any compression. Fig. 7 shows the inference power and area consumption normalized to the SRAM design from [29]. Notably, due to SRAM's high leakage, [29] exhibits the highest power usage during inference, while the MRAM-based [30] is the most power-efficient. Although our hybrid design benefits from sparse compression, reducing the scale of the design, the integrated SRAM part offsets these advantages, positioning our hybrid design's power efficiency between the SRAM- and MRAM-based designs. It's important to note that the power plot's y-axis is log-scaled. In terms of area, [30] capitalizes on MRAM's high storage density, requiring almost half the area of the SRAM-based [29] for the same model. Our hybrid design, leveraging N:M structured sparsity, needs only about 30% of the area compared to [29]. Although the SRAM PE occupies a larger area than the MRAM PE, in our hybrid design, only

Table 2: Hardware Specs

	SRAM PE			MRAM PE	
	Area(mm^2)	Power(mW)		Area(mm^2)	Power(mW)
Decoder	0.0168	0.96	Memory Array (1024 x 512)	0.00686	-
Bit Cell	0.0231	1.2	Parallel Shift Acc	0.00258	0.834
Shift Acc	0.0148	4.2	Col Decoder + Driver	0.0243	1.58
Index Decoder	0.06	7.4	Row Decoder + Driver	0.0037	0.68
Adder	0.14	12.11	Adder Tree	0.044	16.3
Global Buffer	0.0065	0.0004/bit/access	Resistance	4408Ω / 8759Ω	
Global ReLU	0.00719	0.12	Single bit Set/Reset Energy	0.048pJ	

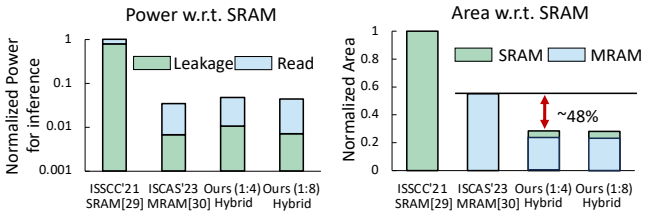

Figure 7: Power and area comparison

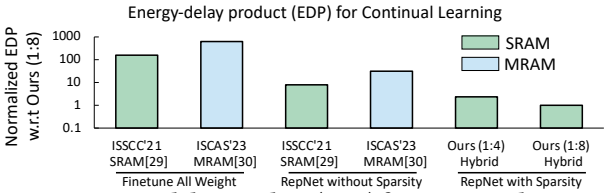

Figure 8: Energy-delay product (EDP) for Continual Learning

about 4% of the area is dedicated to SRAM PEs, with the majority allocated to MRAM PEs for the backbone model.

While the SRAM-PE exhibits higher leakage and area overhead, it offers advantages in terms of lower write energy and latency. This results in a considerably reduced energy-delay product (EDP) compared to the MRAM PE, making the SRAM PE more suitable for weight updates and learning. Initially, we mapped the backbone model onto prior works [29] and [30]. These designs need to fine-tuning the entire model to absorb new knowledge, resulting in the highest EDP for continual learning processes, as shown in Fig. 8. Changing the model to RepNet reduces the volume of weights updates, thereby lowering the EDP. Our proposed hybrid design takes this a step further. By integrating the compact and compressed RepNet path, it drastically diminishes the requirement of writing operations. This reduction is due to the fewer weights updates in compressed sparse model. Additionally, these updates, executed within the SRAM PE, showcase the lowest EDP during continual learning phases. This combination of strategies underlines the effectiveness of our design in optimizing continual learning processes.

6 CONCLUSION

In this work, we propose an efficient on-device continual learning system with the fully-digital hybrid PIM design. This system is a product of synergistic hardware and algorithm co-design. We develop SRAM- and MRAM-based sparse PEs for hardware, supporting both inference and on-device learning. Algorithmically, we deploy the RepNet structure in our system and optimize the mapping methodology. This design features NVM-based PE, capitalizes on its dense storage and low leakage attributes by storing the majority of the weights. The weights stored in the NVM-based PE are frozen and remain unchanged during on-device learning. Complementing this, the SRAM-based PE is leveraged for its fast operation and rapid rewriting capabilities, playing a crucial role in enabling the system's learning capability. Extensive experiments prove the system's ultra-high energy efficiency, showcasing its potential for efficient AI acceleration.

ACKNOWLEDGMENTS

This work is supported in part by the National Science Foundation under Grant No.2314591, No.2414603, No.2349802, No.2342726, No.2328805.

REFERENCES

[1] S. Mittal. A survey of reram-based architectures for processing-in-memory and neural networks. *CD-MAKE*, 1(1):75–114, 2019.
[2] L. Song et al. Pipelayer: A pipelined reram-based accelerator for deep learning. In *2017 HPCA*, pp. 541–552.
[3] F. Zhang et al. Mitigate parasitic resistance in resistive crossbar-based convolutional neural networks. *J. Emerg. Technol. Comput. Syst.*, 16(3), may 2020.
[4] F. Zhang et al. Xma: A crossbar-aware multi-task adaption framework via shift-based mask learning method. DAC '22, pp. 271–276, New York, NY, USA.
[5] Z. He et al. Accelerating low bit-width deep convolution neural network in mram. In *2018 ISVLSI*, pp. 533–538.
[6] S. Angizi et al. Mrima: An mram-based in-memory accelerator. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(5):1123–1136.
[7] D.-Q. You et al. An 8b-precision 8-mb stt-mram near-memory-compute macro using weight-feature and input-sparsity aware schemes for energy-efficient edge ai devices. *IEEE Journal of Solid-State Circuits*, pp. 1–12, 2023.
[8] H. Yan et al. icelia: A full-stack framework for stt-mram-based deep learning acceleration. *IEEE Transactions on Parallel and Distributed Systems*, 31(2):408–422.
[9] Y.-C. Chiu et al. A 22nm 4mb stt-mram data-encrypted near-memory computation macro with a 192gb/s read-and-decryption bandwidth and 25.1-55.1tops/w 8b mac for ai operations. In *2022 ISSCC*, volume 65, pp. 178–180, 2022.
[10] Y.-C. Chiu et al. A 22-nm 1-mb 1024-b read data-protected stt-mram macro with near-memory shift-and-rotate functionality and 42.6-gb/s read bandwidth for security-aware mobile device. *IEEE Journal of Solid-State Circuits*, 57(6).
[11] F. Zhang et al. Xbm: A crossbar column-wise binary mask learning method for efficient multiple task adaption. In *2022 ASP-DAC*, pp. 610–615, 2022.
[12] F. Zhang et al. Xma2: A crossbar-aware multi-task adaption framework via 2-tier masks. *Frontiers in Electronics*, 3, 2022.
[13] A. Sridharan et al. Dspimm: A fully digital sparse in-memory matrix vector multiplier for communication applications. In *2023 DAC*, pp. 1–6, 2023.
[14] H. Mori et al. A 4nm 6163-tops/w/b **4790 − TOPS/mm^2/b** sram based digital-computing-in-memory macro supporting bit-width flexibility and simultaneous mac and weight update. In *2023 ISSCC*, pp. 132–134, 2023.
[15] H. Fujiwara et al. A 5-nm 254-tops/w 221-tops/mm2 fully-digital computing-in-memory macro supporting wide-range dynamic-voltage-frequency scaling and simultaneous mac and write operations. In *2022 ISSCC*, volume 65, pp. 1–3, 2022.
[16] S. Liu et al. 16.2 a 28nm 53.8tops/w 8b sparse transformer accelerator with in-memory butterfly zero skipper for unstructured-pruned nn and cim-based local-attention-reusable engine. In *2023 ISSCC*, pp. 250–252, 2023.
[17] Y. Luo et al. Accelerating deep neural network in-situ training with non-volatile and volatile memory based hybrid precision synapses. *IEEE TC*, 2020.
[18] G. Krishnan et al. Hybrid rram/sram in-memory computing for robust dnn acceleration. *IEEE TCAD*, 41(11):4241–4252, 2022.
[19] J. Pool et al. Channel permutations for n:m sparsity. In M. Ranzato et al., editors, *Advances in Neural Information Processing Systems*, volume 34, 2021.
[20] A. Mishra et al. Accelerating sparse deep neural networks, 2021.
[21] Y.-H. Chen et al. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. In *2016 ISCA*, pp. 367–379, 2016.
[22] Y.-D. Chih et al. 16.4 an 89tops/w and 16.3tops/mm2 all-digital sram-based full-precision compute-in memory macro in 22nm for machine-learning edge applications. In *2021 ISSCC*, volume 64, pp. 252–254, 2021.
[23] L. Yang et al. Rep-net: Efficient on-device learning via feature reprogramming. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 12277–12286, June 2022.
[24] M.-E. Nilsback et al. Automated flower classification over a large number of classes. In *Indian Conference on Computer Vision, Graphics Image Processing*.
[25] L. Bossard et al. Food-101 – mining discriminative components with random forests. In *European Conference on Computer Vision*, 2014.
[26] O. M. Parkhi et al. Cats and dogs. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3498–3505, 2012.
[27] S. Angizi et al. Pima-sim 1.0: A simple & flexible processing-in-memory support for nvsim. https://github.com/ASU-ESIC-FAN-Lab/PIMA-SIM.
[28] X. Dong et al. Nvsim: A circuit-level performance, energy, and area model for emerging nonvolatile memory. *IEEE TCAD*, 31(7):994–1007, 2012.
[29] Y. D. Chih et al. An 89tops/w and 16.3tops/mm2 all-digital sram-based full-precision compute-in memory macro in 22nm for machine-learning edge applications. *2021 ISSCC*, 64:252–254.
[30] L. Lu et al. A 129.83 tops/w area efficient digital sot/stt mram-based computing-in-memory for advanced edge ai chips. In *2023 IEEE ISCAS*, pp. 1–5, 2023.