APX-DREAM-CIM: An Approximate Digital SRAM-Based CIM Accelerator for Edge AI

Asmae El arrassi*, Chenruiyuan Yu*, Mottaqiallah Taouil*, Rajiv Joshi‡, Said Hamdioui*
*Department of Quantum and Computer Engineering, Delft University of Technology, Delft, The Netherlands
Email: {a.elarrassi, c.ayu, m.taouil, s.hamdioui} @tudelft.nl
‡Thomas J. Watson Research Center IBM, New York, USA Email: rvjoshi@us.ibm.com

***Abstract*—With the increasing demand for energy-efficient solutions in smart edge applications, there is a pressing need for computing architectures that can effectively manage diverse and computation-intensive workloads. To address this, we propose APX-DREAM-CIM, an approximate digital SRAM-based Computation-In-Memory (CIM) accelerator specifically designed to maximize energy and area efficiency with minimal impact on accuracy by investigating cross-layer optimizations. The architecture uses, on the one hand, quantized models and, on the other hand, integrates a range of approximate adder designs, each offering distinct trade-offs between hardware efficiency and computational precision. To further enhance resilience to approximation-induced errors, we introduce a novel Approximate-Aware Training (AAT) methodology, which models the approximate behavior of the hardware during training, enabling the network to adapt accordingly. We evaluate APX-DREAM-CIM using the MNIST and CIFAR-10 datasets with LeNet-5 and ResNet-20 topologies. Experimental results show that APX-DREAM-CIM achieves up to 17% energy and 16% area reduction compared to the exact baseline architecture. Moreover, AAT enables the system to retain, and in some cases, even exceed, the accuracy of standard quantized models.**

***Index Terms*—Computation-in-Memory, SRAM, digital, MAC.**

I. Introduction

Artificial intelligence (AI) is a cutting-edge technology that mimics human intelligence to perform various tasks. Today's AI models have rapidly grown in size and computational complexity, making AI-powered applications increasingly resource-intensive. This growth comes with high **energy and area** consumption and places a significant strain on memory bandwidth [1, 2]. Computation-in-memory (CIM) has been proposed to address these challenges by eliminating the excessive energy spent on moving massive amounts of data between the processing unit and memory [3]. This is achieved by integrating computation and storage in the same physical location [3–7]. To maximize the benefits, cross-layer optimizations are crucial for reducing hardware overhead and enabling energy and area-efficient CIM accelerators.

Several CIM solutions exist to reduce energy and area consumption. These solutions can be classified into *model-* [8, 9], *design and architecture-* [10–12], and *technology*-level [13, 14] optimization. At the *model*-level, techniques such as quantization [8], pruning [9], and sparsity-aware inference [15], aim to reduce model size and computational complexity. At the *Design and architecture-level*, optimization techniques focus on improving circuit efficiency, which can be classified into two subclasses depending on the nature of the Multiply-and-Accumulate (MAC) operation: *analog* CIM (A-CIM) and *digital* CIM (D-CIM). A-CIM optimization relies on reducing the Analog-to-Digital Converter (ADC) high energy and area overhead [14]. On the other hand, D-CIM optimization targets overhead from multipliers and adder trees, using techniques such as inverted logic [10] and hardware approximation [11, 12]. At the *technology level*, approaches such as hybrid threshold logic [7], advanced nodes [13, 16], and emerging memory technologies [14] further enhance energy and area-efficiency. Prior work has addressed optimizations typically at individual abstraction levels. Cross-layer approaches [11, 12] remain relatively unexplored in CIM Designs. Consequently, there is a huge need for cross-layer optimizations to further improve area/energy efficiency.

This paper presents an approximate digital SRAM-based CIM architecture, referred to as *APX-DREAM-CIM*, which builds on the baseline architecture of DREAM-CIM— an efficient D-CIM design that addresses the high energy and area overhead of bulky adder-tree structures [17]. In this work, we combine multiple optimization techniques to maximize area and energy efficiency. The core contribution of this work lies in replacing the exact adders with approximate adders to achieve significant area and energy savings. Furthermore, this work explores Approximate-Aware Training (AAT), a methodology that incorporates the behavior of approximate hardware into the training process. AAT enables the neural networks to adapt to approximation-induced errors, significantly improving inference accuracy and robustness. The proposed design is evaluated using LeNet-5 and ResNet-20 topologies on standard image classification datasets, including MNIST and CIFAR-10. The main contributions of this work are as follows.

- **Approximate MAC units:** The architecture incorporates approximate adders into the MAC units to reduce computational overhead with minimal impact on accuracy.
- **Area and Energy Efficiency:** The proposed design achieves notable improvements in area and energy consumption compared to the exact baseline.
- **Approximation-Aware Training (AAT):** To mitigate the potential accuracy loss from approximation, we propose an AAT method that models the approximate behavior of the accelerator during training.
- **Circuit-to-Application Evaluation:** Both circuit-level

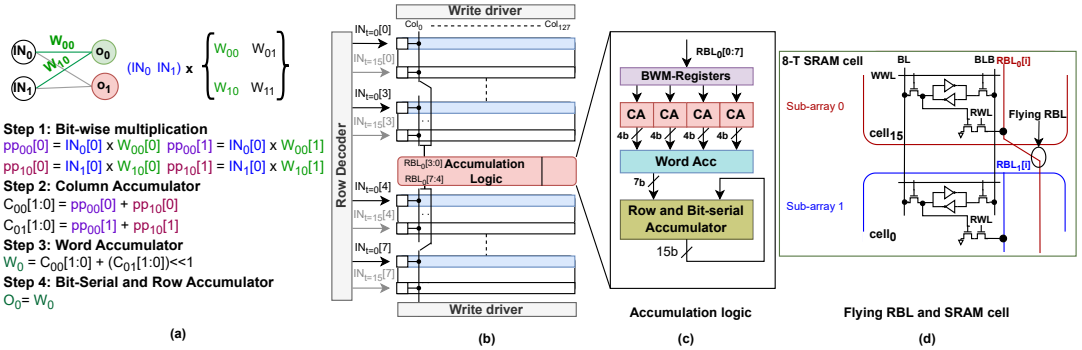

Fig. 1. Mapping MAC operation to DREAM-CIM Accelerator [17]

and application-level evaluations are conducted to validate the efficiency and accuracy of the proposed approximate architecture.

The remainder of the paper is organized as follows. Section II explains the baseline architecture. Section III discusses the proposed architecture. Section IV presents the circuit-level simulation results and State-of-the-Art (SOTA) comparison. Finally, Section V concludes the paper.

II. DREAM-CIM

This section describes the baseline CIM architecture [17].

A. Baseline Architecture Concept

The main concept of the architecture is illustrated using the Neural Network (NN) in Figure 1a. In this example, the NN consists of two input neurons IN_0 and IN_1, and two output neurons o_0 and o_1. o_0 is connected to the input neurons (IN_0 and IN_1) with a strength of W_{00} and W_{10}, respectively. The inputs and weights are represented by 2-bit numbers for simplification. The MAC operation is implemented in four distinct steps as shown in Figure 1a.

Step 1: Bit-wise Multiplication - In this step, a bit-wise multiplication between the first bit of each input bit ($IN_0[0]$ and $IN_1[0]$) with the weights ($W_{00}[1:0]$ and $W_{10}[1:0]$) is performed; this results in two partial products (pp), namely $pp_{00}[1:0]$ and $pp_{10}[1:0]$.

Step 2: Column Accumulator - In this step, bits belonging to the same position within each pp are accumulated together (i.e., column accumulation). In the example, $pp_{00}[0]$ is accumulated with $pp_{10}[0]$, and $pp_{00}[1]$ with $pp_{10}[1]$.
This results in a two-bit intermediate result for each column accumulation, namely $C_{00}[1:0]$ and $C_{01}[1:0]$.

Step 3: Word Accumulator - In this step, the results of the Column Accumulator are added together. It is required that the partial sums of the Column Accumulator are properly aligned here, e.g., a 1-bit left shift operation is performed on C_{01}. The result is the partial sum W_0.

Step 4: Bit-Serial and Row Accumulator - Finally, in the last step, we add the partial results of the serially applied inputs. However, in step four the resulting W_0 is shifted with one bit to the left to compensate for the input weight and then accumulated with the current MAC cycle result. Note that as we access only one row at a time per sub-array. hence, we need a Row Accumulator to sum up the row results.

B. Baseline Architecture Overview

Figure 1b provides an overview of the baseline architecture. It consists of a 16Kb SRAM array partitioned into eight sub-arrays, each containing 16 rows and 128 columns. This partitioning enhances the parallelism of the architecture, allowing eight rows to be activated simultaneously. The results are read through dedicated Read-Bit-Lines (RBLs) for each bank, with each column producing eight products that are processed by the accumulation logic, as illustrated in Figure 1b and c. The architecture is based on an 8-T SRAM cell shown in Figure 1d. The accumulation logic includes a pipeline stage, referred to as Bit-Wise-Multiplication (BWM) registers, which captures the output of the SRAM array from each RBL. The Column Accumulator (CA) then counts the number of ones in the Mem-registers for each column. Since the architecture is designed to support 4-bit weights, four columns are used to represent the different bit positions of the weights. The outputs of the CAs from these four columns are aligned and combined in the Word Accumulator to generate the final result. Inputs to the array are provided sequentially, and rows within each bank are also activated sequentially. To handle this sequential operation, the design incorporates a row and bit-serial accumulator, which accumulates the sequentially produced results efficiently.

III. APX-DREAM-CIM

A. APX-DREAM-CIM Concept

The fundamental operation in neural networks is the MAC operation, typically represented as the dot product between input activations and weights. At a bit-level, this operation can be described by the equation:

$$O_i = \mathbf{IN} \cdot \mathbf{w} = \sum_{i=0}^{P-1} \sum_{j=0}^{Q-1} \sum_{k=0}^{N-1} \left(2^i \cdot \mathrm{IN}_k[i]\right) \cdot \left(2^j \cdot w_k[j]\right) \quad (1)$$

$$= \sum_{i=0}^{P-1} PS[i] \quad (2)$$

where $\mathbf{IN}$ and $\mathbf{w}$ are N-dimensional input and weight vectors represented with P- and Q-bit integers, and $\mathrm{IN}_k[i]$ and $w_k[j]$ denote the i-th and j-th bits of the k-th element. These products are grouped into partial sums (PS) (see Equation 2).

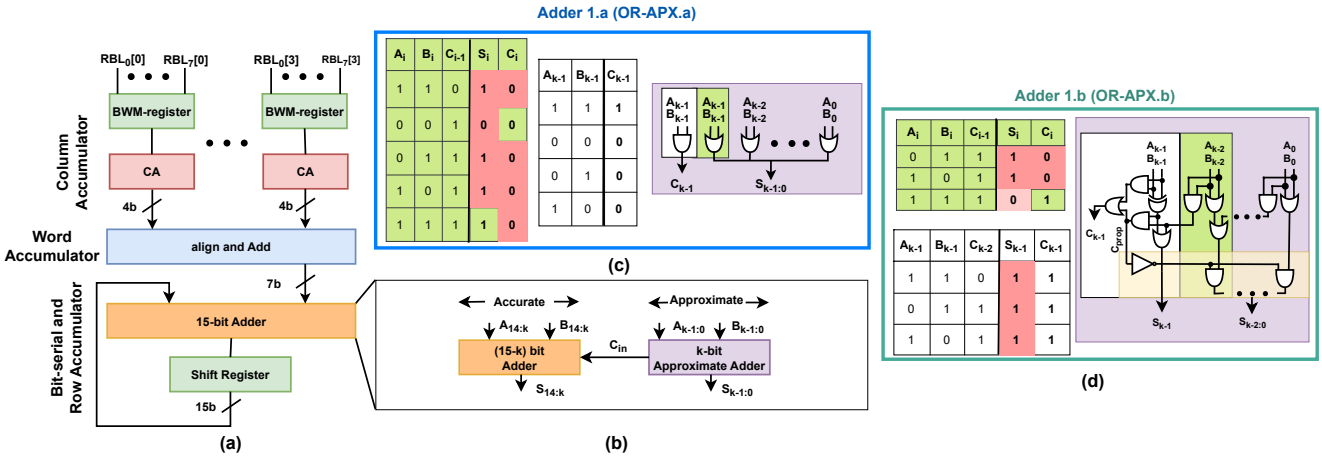

Fig. 2. An Illustration of (a) The Accumulation Logic, (b) The Bit-serial and Row Accumulator Approximation, (c), and (d) OR-based Approximate Adders.

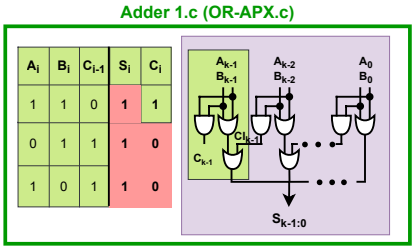

Fig. 3. OR-based Approximate Adder Design

In the APX-DREAM-CIM architecture, the accumulation logic includes column, word, bit-serial, and row accumulators. Among these, the bit-serial and row accumulators consume the most area and energy (see Figure 2a). To optimize the energy and area-efficiency, APX-DREAM-CIM introduces approximate computing by replacing the exact adders in these units with approximate full adders (FAs). The accumulation logic is split into two parts: the k least significant bits (LSBs) use approximate adders, and the remaining most significant bits (MSBs) use exact adders (see Figure 2b). The approximated MAC operation is expressed by:

$$O_i = \sum_{i=k}^{P-1} PS[i] + \sum_{i=0}^{k-1} \text{APX}(\text{PS}[i]) \tag{3}$$

Here, APX() denotes approximate accumulation, and k is the number of approximated bits. To ensure that the network adapts to the hardware-induced approximations, we propose AAT, which models this directly in software during training. Specifically, Equation 3 is used during the forward pass, allowing the model to learn under the same bit-level inaccuracies present in the accelerator. This technique ensures compatibility between the training and the deployed hardware.

B. APX-DREAM-CIM Implementation

APX-DREAM-CIM is implemented based on the concept described in Section III-A. To enable approximation, we integrated various approximate adder designs into the accumulation logic, each introducing a trade-off between accuracy and energy/area overhead. The different types of approximate adders are implemented as follows:

Adder 1.a — OR-APX.a *(see Figure 2c):* This adder [18] employs an OR gate to compute the sum, effectively eliminating the need for carry propagation. The MSB bit of the approximation part uses an additional AND gate to compute the carry-out. As shown in Figure 2c, this simplified logic leads to incorrect outputs for several input combinations (highlighted in red in the truth tables). The boolean equations of the OR-APX.a approximate adder are as follows:

$$S_i = A_i + B_i \quad \text{and} \quad C_{k-1} = A_{k-1}.B_{k-1} \tag{4}$$

Here, A_i and B_i are the adder inputs, and S_i is the sum output at index i and C_{k-1} the carry-out of the approximated adder.

Adder 1.b — OR-APX.b *(see Figure 2d):* This adder [19] uses a more complex approximation logic. At the approximated MSB index k-1 (see white box in Figure 2.d), the sum is produced through cascaded XOR and OR gates with basic carry propagation. A carry prediction mechanism is incorporated, consisting of AND and OR gates, to generate and anticipate the MSB C_{k-1}. The bottom of the figure contains an Error Reduction Logic (ERL) block (yellow box) comprising of multiple AND gates and a single NOT gate. It changes the sum bit values to zero for indices k-2 to 0 when the estimated value is too high. This predictive logic helps reduce the adder's latency while maintaining acceptable accuracy. In the LSBs, the carry is generated using a single AND gate, while the sum is produced through cascaded OR gates with basic carry propagation as shown in the green box. The boolean equations of the OR-APX.b approximate adder are as follows:

$$C_i = A_i \cdot B_i \quad \text{and} \quad C_{prop} = ((A_{k-1} \oplus B_{k-1}) \cdot C_{k-2}) \tag{5}$$

$$S_i = (A_i + B_i + C_{i-1}) \cdot \neg C_{prop} \tag{6}$$

$$C_{k-1} = (A_{k-1} \cdot B_{k-1}) + C_{prop} \tag{7}$$

$$S_{k-1} = (A_{k-1} \oplus B_{k-1}) + C_{k-2} \tag{8}$$

Here, A_i and B_i denote the corresponding input bits, S_i and C_i represent the sum and carry for $i = 0$ to $k-2$, C_{prop} is the propagated carry, and S_{k-1} and C_{k-1} represent the sum and carry-out generated by the MSB.

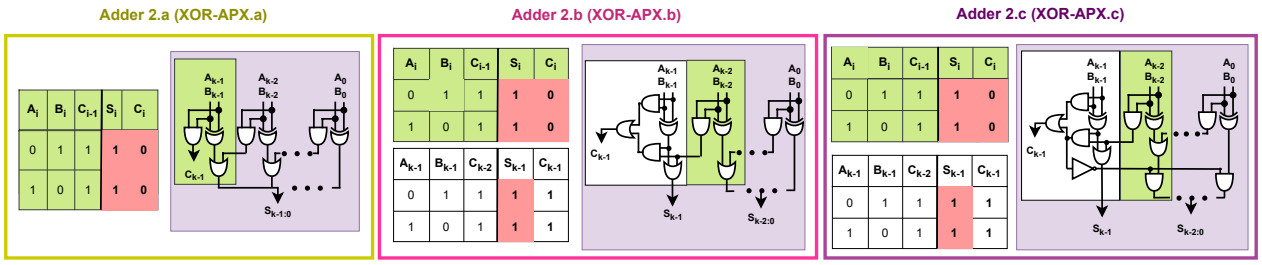

Fig. 4. XOR-based Approximate Adder Designs.

Adder 1.c — OR-APX.c *(see Figure 3):* This design simplifies the OR-APX.b architecture by relying solely on a unified structure for all output bits using an approximated FA design. It consists of an AND gate to generate the carry and a cascaded OR gate to generate the sum. The Boolean equations for the OR-APX.c approximate adder are as follows:

$$C_i = A_i \cdot B_i \quad \text{and} \quad S_i = A_i + B_i + C_{i-1} \tag{9}$$

where A_i and B_i denote the adder inputs for bit i, and S_i and C_i the sum and carry for bit i, respectively.

Adder 2.a — XOR-APX.a *(see Figure 4a):* This design is similar to adder 1.c OR-APX.c, except that the OR gate is replaced by an XOR gate. Figure 4a illustrates the adder architecture. This modification reduces the approximation error compared to OR-APX.c, as can be seen in the truth tables. The Boolean equations for the XOR-APX.a adder is as follows:

$$C_i = A_i \cdot B_i \quad \text{and} \quad S_i = (A_i \oplus B_i) + C_{i-1} \tag{10}$$

where A_i and B_i denote the adder inputs for bit i, and S_i and C_i the sum and carry for bit i, respectively.

Adder 2.b — XOR-APX.b *(see Figure 4b):* This design extends XOR-APX.a by incorporating an accurate carry prediction logic composed of an additional AND and OR gate at the MSB bit. This enhancement generates a more precise carry input for the exact adder, thereby improving the overall accuracy of the adder. The Boolean equations for the XOR-APX.b approximate adder are as follows:

$$C_i = A_i \cdot B_i \quad \text{and} \quad S_i = (A_i \oplus B_i) + C_{i-1} \tag{11}$$

$$C_{k-1} = (A_{k-1} \cdot B_{k-1}) + (A_{k-1} \oplus B_{k-1}) \cdot C_{k-2} \tag{12}$$

where A_i and B_i denote the adder inputs for bit i, and S_i and C_i the sum and carry for bit i, respectively.

Adder 2.c — XOR-APX.c *(see Figure 4c):* This design builds further on the OR-APX.b adder. However, this version replaces the OR gate with an XOR gate as shown in Figure 4c. This results in a more accurate result, as shown in the truth table. The correction logic ensures higher accuracy by compensating for potential inaccuracies introduced during the addition process. The boolean equations of this adder are:

$$C_i = A_i \cdot B_i \quad \text{and} \quad C_{prop} = (A_{k-1} \oplus B_{k-1}) \cdot C_{k-2} \tag{13}$$

$$S_i = (A_i \oplus B_i + C_{i-1}) \cdot \neg C_{prop} \tag{14}$$

$$C_{k-1} = A_{k-1} \cdot B_{k-1} + C_{prop} \tag{15}$$

$$S_{k-1} = (A_{k-1} \oplus B_{k-1}) + C_{k-2} \tag{16}$$

TABLE I
SIMULATION PARAMETERS.

Technology	22 nm
Supply voltage (V)	0.6 V
Simulations	SPICE-level (Cadence Spectre)
Parasitic extraction	Calibre, PEX
Synthesis	Cadence Genus
Temperature	27 °C
SRAM cell	8-T
Unit macro size	16 kb (128x128b)

Here, A_i and B_i denote the corresponding input bits, S_i and C_i represent the sum and carry for $i = 0$ to $k-2$, C_{prop} is the propagated carry, and S_{k-1} and C_{k-1} represent the sum and carry-out generated by the MSB.

IV. CIRCUIT LEVEL SIMULATIONS

A. Experiment Setup

The proposed APX-DREAM-CIM architecture is simulated in SPICE using GlobalFoundries 22 nm CMOS technology. The simulation parameters are presented in Table I. The energy and latency results are extracted using post-layout SPICE-level simulations. The area has been derived using Cadence Genus synthesis results. To estimate the power, it is worth noting that the results were reported for the worst-case scenario where we assumed a bit-sparsity of 50%, i.e., 50% of the weight bits were initialized to '0' and the other 50% to '1'.

To analyze the accuracy, we perform image classification using two representative neural network models, i.e., LeNet-5 [20] trained on the MNIST dataset [21] and ResNet-20 [22] trained on CIFAR-10 [23]. To map the floating-point weights and activation outputs to the integer-based APX-DREAM-CIM macro, quantization is required. We use the QNN method [24] for quantization due to its high accuracy. The networks are trained offline using the RedBit framework [8], employing back-propagation [25] and stochastic gradient descent [26]. The resulting quantized models are then mapped onto the APX-DREAM-CIM architecture for evaluation.

B. Area and energy Results

Figure 5 presents the normalized area and energy results for various approximate adder designs compared to the baseline exact adder architecture presented in Section II. Each subfigure corresponds to a specific type of approximate adder, evaluated across different approximation levels defined by the number of LSBs bits k replaced with approximate logic (x-axis).

Both area and energy values are normalized to the baseline to highlight the efficiency gains; they are represented by the

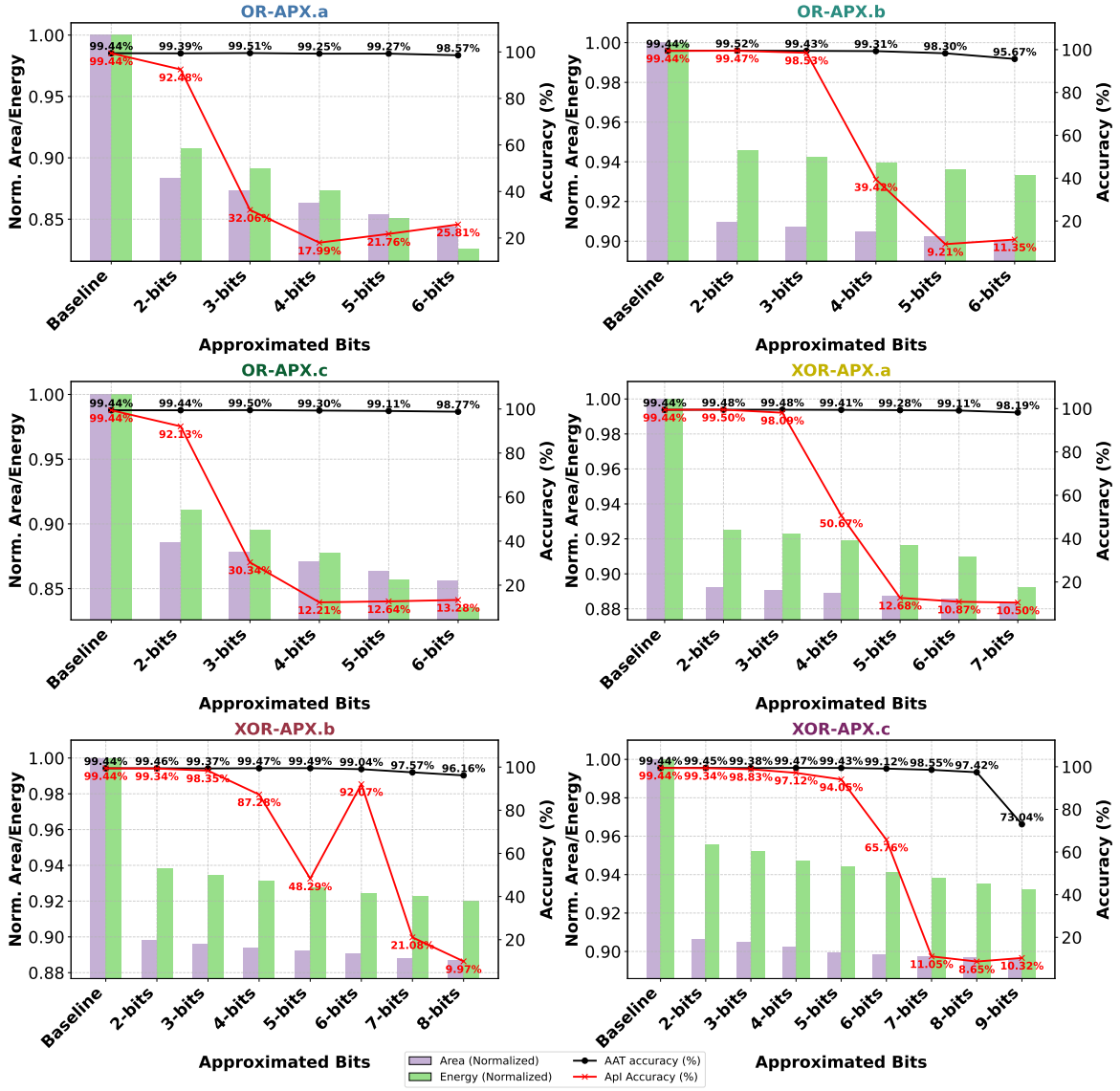

Fig. 5. Normalized Energy and Area Comparison Along with Accuracy for Approximate Inference (Red) and Approximation-Aware Training (Black).

purple and green bars, respectively. As the level of approximation increases, all designs exhibit consistent reductions in area and energy consumption. Notably, OR-APX.a and OR-APX.c achieve area reductions of up to 16% and 14%, respectively, demonstrating the potential of approximate computing in minimizing hardware complexity. Despite its relatively higher structural complexity, OR-APX.b still achieves an area reduction of approximately 10%.

Similarly, the XOR-based designs, XOR-APX.a, XOR-APX.b, and XOR-APX.c, exhibit steady area improvements, with reductions of approximately 12.5%, 12%, and 10.2%, respectively. Overall, the area savings across all designs range from approximately 8% to 16%, depending on the adder type and approximation level.

In terms of energy, improvements also scale with the degree of approximation and the specific adder architecture. OR-APX.a and OR-APX.c have the highest energy savings, achieving up to 17% reduction compared to the baseline. Even the more complex designs, such as OR-APX.b and XOR-APX.c, yield meaningful energy gains of around 7%. For minimal approximation (i.e., using only 2-bit approximation), energy reductions fall in the range of 4% to 9%, highlighting the potential for lightweight energy optimization even with modest approximation.

C. Accuracy results

Figure 5 also illustrates the accuracy results on the MNIST dataset. The first accuracy curve (shown in red) represents inference outcomes from the APX-DREAM-CIM accelerator when the network is trained using exact adders but evaluated using Approximate Inference (ApI) accuracy. In contrast, the second curve (shown in black) corresponds to AAT accuracy, where the network is trained in the presence of the used approximate adder.

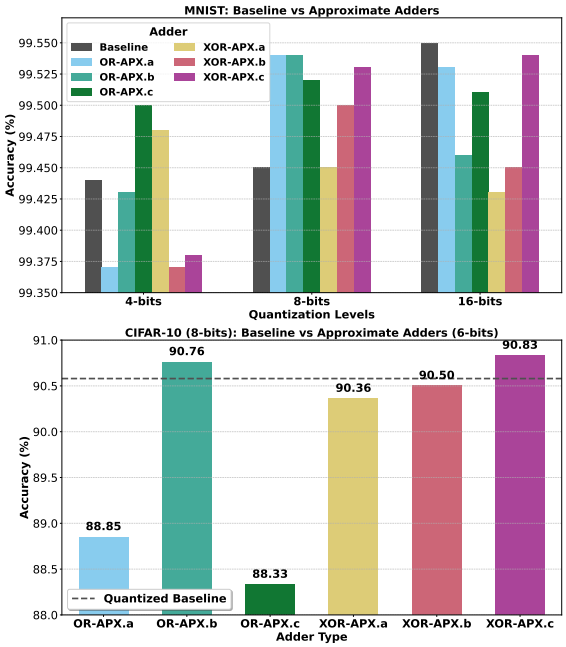

Fig. 6. APX-DREAM-CIM vs. Quantization Accuracy for MNIST and CIFAR-10.

To enable AAT, the full APX-DREAM-CIM accelerator—including its approximate arithmetic components—is integrated into the training process. This allows the network to learn and compensate for the inherent inaccuracies introduced by approximation, thereby improving inference robustness.

Under ApI, OR-APX.a and OR-APX.c exhibit a sharp decline in accuracy, dropping to approximately 31% after just 3 bits of approximation. In contrast, OR-APX.b, XOR-APX.a, and XOR-APX.b maintain higher accuracy up to 4-bit approximation. Notably, XOR-APX.c demonstrates the highest robustness among all designs, sustaining an accuracy of around 94% even with 5 bits of approximation.

With AAT, the accuracy degradation is significantly mitigated. OR-APX.a and OR-APX.c achieve high accuracies of 98.57% and 98.77%, respectively, even with 6-bit approximation. Likewise, XOR-APX.a, XOR-APX.b, and XOR-APX.c show strong resilience, maintaining high accuracy even with 7 to 8 bits of approximation.

D. Quantization Vs Approximation results

Figure 6 illustrates the relationship between quantization and approximation accuracy for MNIST and CIFAR-10, using LeNet-5 and ResNet-20, respectively. The results emphasize that, in addition to quantization, approximation should be considered as a key design parameter to achieve optimal hardware efficiency while maintaining high accuracy.

For the MNIST dataset, Figure 6 (top) shows the 4-bit quantized models using OR-APX.c and XOR-APX.a achieve better accuracy than the 4-bit and 8-bit quantized baselines. Meanwhile, for 16-bit quantization, designs using OR-APX.a and XOR-APX.c deliver accuracy comparable to the exact 16-bit baseline, demonstrating that approximation does not compromise accuracy at higher quantization levels.

TABLE II
COMPARISON OF D-CIM DESIGNS

	ISSCC'21 [10]	ISSCC'22 [11]	ISSCC'23 [12]	Baseline [17]	This Work (XOR-APX.b)
Technology Node	22nm	28nm	28nm	22nm	**22nm**
Array Size	64kb	16kb	16kb	16kb	**16kb**
Macro Type	6T	8T	8T	8T	**8T**
Area Eff. (TOPS/mm^2, 8-bit)	4	2.6	4.2	**10.6**	**11.3**
Energy Eff. (TOPS/W, 8-bit)	24.7	15.5	102	**103**	**106**
Accuracy Loss (%) CIFAR-10, ResNet20	N/A	1.5	0.3	**0**	**0.05**

For the CIFAR-10 dataset, Figure 6 (bottom) presents the baseline 8-bit quantized accuracy (grey curve) and the accuracy achieved under 6-bit approximation. Here, OR-APX.a and OR-APX.c recorded the lowest accuracy, around 88%, compared to 90.57% for the baseline. In contrast, XOR-APX.a and XOR-APX.b show only a minimal drop in accuracy, achieving 90.36% and 90.50%, respectively. Notably, OR-APX.b and XOR-APX.c even outperform the baseline, achieving accuracies of 90.76% and 90.83%, respectively.

AAT can, in some cases, lead to higher accuracy than the baseline quantized network. This observation highlights the inherent robustness of neural networks to noise introduced during training [27, 28]. By incorporating approximate computations, AAT acts as a form of regularization, encouraging the model to generalize better and reduce overfitting.

E. SOTA Comparison

Recent works have focused on exact and approximate computing to enhance energy and area efficiency, as shown in Table II. The design presented in [10] uses a 6T SRAM array that performs multiplication using NOR gates and an adder tree for accumulation. This exact computation approach achieves 4 TOPS/mm^2 area efficiency and 24.7 TOPS/W energy efficiency. In [11], the approximation is introduced by replacing FAs in the adder tree with approximate compressors, improving area efficiency at the cost of 1.5% accuracy loss on the CIFAR-10 dataset. Similarly, [12] proposes a dynamic approximation logic, achieving 102 TOPS/W energy efficiency with a 0.3% accuracy loss. Our proposed architecture, APX-DREAM-CIM, which integrates a range of approximate adders into the MAC units, coupled with approximation-aware training, is based on a 22nm 8T SRAM array. Our design offers the best trade-off, achieving 11.3 TOPS/mm² area efficiency, 103 TOPS/W energy efficiency, and only 0.05% of accuracy loss, demonstrating better than state-of-the-art performance.

V. CONCLUSION

We proposed APX-DREAM-CIM, an SRAM-based CIM accelerator using approximate computing to improve energy and area efficiency. By integrating approximate adders into MAC units, we explored hardware–accuracy trade-offs. Evaluations across circuit and application levels with AAT, using LeNet-5 and ResNet-20 on MNIST and CIFAR-10, show that quantization and approximation can significantly reduce cost with minimal accuracy loss.

ACKNOWLEDGEMENTS

This work is funded by CONVOLVE (Grant No. 101070374).

REFERENCES

[1] S. Srinath *et al.*, "Feedback directed prefetching: Improving the performance and bandwidth-efficiency of hardware prefetchers," in *2007 IEEE 13th International Symposium on High Performance Computer Architecture*, 2007.

[2] D. Fujiki *et al.*, "Multi-layer in-memory processing," in *MICRO*, 2022.

[3] S. Hamdioui *et al.*, "Memristor based computation-in-memory architecture for data-intensive applications," in *DATE*, 2015.

[4] J. Wang *et al.*, "A 28-nm compute sram with bit-serial logic/arithmetic operations for programmable in-memory vector computing," *JSSC*, 2019.

[5] J. Ahn *et al.*, "Pim-enabled instructions: A low-overhead, locality-aware processing-in-memory architecture," *ACM SIGARCH Computer Architecture News*, 2015.

[6] A. Sebastian *et al.*, "Memory devices and applications for in-memory computing," *Nature nanotechnology*, 2020.

[7] H. Fujiwara *et al.*, "A 5-nm 254-tops/w 221-tops/mm 2 fully-digital computing-in-memory macro supporting wide-range dynamic-voltage-frequency scaling and simultaneous mac and write operations," in *ISSCC*, 2022.

[8] A. Santos *et al.*, "Redbit: An end-to-end flexible framework for evaluating the accuracy of quantized cnns," *arXiv preprint arXiv:2301.06193*, 2023.

[9] S. Han *et al.*, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[10] Y.-D. Chih *et al.*, "16.4 an 89tops/w and 16.3 tops/mm 2 all-digital sram-based full-precision compute-in memory macro in 22nm for machine-learning edge applications," in *ISSCC*, 2021.

[11] D. Wang *et al.*, "Dimc: 2219tops/w 2569f2/b digital in-memory computing macro in 28nm based on approximate arithmetic hardware," in *2022 IEEE international solid-state circuits conference (ISSCC)*, vol. 65. IEEE, 2022, pp. 266–268.

[12] Y. He *et al.*, "7.3 a 28nm 38-to-102-tops/w 8b multiply-less approximate digital sram compute-in-memory macro for neural-network inference," in *2023 IEEE International Solid-State Circuits Conference (ISSCC)*. IEEE, 2023, pp. 130–132.

[13] H. Fujiwara *et al.*, "34.4 a 3nm, 32.5 tops/w, 55.0 tops/mm 2 and 3.78 mb/mm 2 fully-digital compute-in-memory macro supporting int12$\times$ int12 with a parallel-mac architecture and foundry 6t-sram bit cell," in *2024 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 67. IEEE, 2024, pp. 572–574.

[14] R. Bishnoi *et al.*, "Special session–emerging memristor based memory and cim architecture: Test, repair and yield analysis," in *VTS*, 2020.

[15] G. Huang *et al.*, "Condensenet: an efficient densenet using learned group convolutions. arxiv," *arXiv preprint arXiv:1711.09224*, 2017.

[16] H. Mori *et al.*, "A 4nm 6163-tops/w/b 4790-tops/mm2/b sram based digital-computing-in-memory macro supporting bit-width flexibility and simultaneous mac and weight update," in *ISSCC*, 2023.

[17] A. El Arrassi *et al.*, "Dream-cim: A digital sram-based cim accelerator for energy- and area-efficient edge ai," *To appear in TCASAI*, May 2025, https://pure.tudelft.nl/ws/portalfiles/portal/246634945/2025.

[18] H. R. Mahdiani *et al.*, "Bio-inspired imprecise computational blocks for efficient vlsi implementation of soft-computing applications," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 57, pp. 850–862, 2009.

[19] J. Lee *et al.*, "A novel approximate adder design using error reduced carry prediction and constant truncation," *IEEE Access*, vol. 9, pp. 119 939–119 953, 2021.

[20] Y. LeCun *et al.*, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, 1998.

[21] L. Deng, "The mnist database of handwritten digit images for machine learning research," *IEEE Signal Processing Magazine*, 2012.

[22] K. He *et al.*, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016.

[23] A. Krizhevsky *et al.*, "Cifar-10 (canadian institute for advanced research)." [Online]. Available: http://www.cs.toronto.edu/ kriz/cifar.html

[24] I. Hubara *et al.*, "Quantized neural networks: Training neural networks with low precision weights and activations," *The Journal of Machine Learning Research*, 2017.

[25] H. J. Kelley, "Gradient theory of optimal flight paths," *Ars Journal*, 1960.

[26] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

[27] A. Neelakantan *et al.*, "Adding gradient noise improves learning for very deep networks," *arXiv preprint arXiv:1511.06807*, 2015.

[28] S. Anwar *et al.*, "Structured pruning of deep convolutional neural networks. corr abs/1512.08571 (2015)," *arXiv preprint arXiv:1512.08571*, 2015.