Proceedings of CONF-SPML 2026 Symposium: The 2nd Neural Computing and Applications Workshop 2025
DOI: 10.54254/2755-2721/2026.TJ30682

In-Memory Computing Elements for Energy-Efficiency Optimization in Deep Neural Network Inference:A Literature Review

Jihong Liu[1*], **Taoyi Zhang**[2]

[1]*Tianjin University, Tianjin, China*
[2]*Whartoon School, Chengdu, China*
Corresponding Author. Email: liujihongln2008@163.com

Abstract. Deep neural networks are facing emerging challenges in energy efficiency, primarily driven by two trends: the development of extremely large-scale models, such as large language models with hundreds of billions of parameters, and the deployment of AI workloads on edge computing devices. As an emerging computing paradigm, in-memory computing offers a promising solution to overcome the resource overhead associated with data movement in the von Neumann architecture, which separates memory and computing units. It significantly enhances both energy efficiency and computational throughput. While substantial research has focused on optimizing novel device materials or algorithms, there remains a relative lack of systematic review regarding macro-level in-memory computing components that bridge device-level circuit elements and system-level computing-on-chip architectures.

This survey provides a review of in-memory computing macros for energy-efficient deep neural network inference. The research findings are categorized into three representative directions: (1) mixed floating-point and integer architectures, (2) outlier-aware quantization schemes, and (3) support for sparse computations. These correspond respectively to optimization strategies for computational precision, numerical representation, and energy efficiency in computational scale. Through a comparative analysis of the trade-offs among energy efficiency, precision, and functional adaptability across different designs, this study aims to offer insights for the development of in-memory computing macros and outline prospective directions for future macro designs.

Keywords: In-Memory Computing, Energy Efficiency Optimization, Mixed-Precision Architecture, Outlier-Aware Quantization

1. Introduction

The widespread adoption of deep neural networks (DNNs) has made energy efficiency a crucial challenge for sustainable system development. While large-scale models with billions of parameters significantly improve performance, the von Neumann architecture's memory-processor separation leads to substantial energy waste from data movement. This undermines energy efficiency in both training and inference phases. Meanwhile, the deployment of AI on edge devices such as IoT nodes and smart terminals requires local inference under tight power and computational constraints. These demands

DOI: 10.54254/2755-2721/2026.TJ30682

drive the need for energy-efficient optimization across high-performance and edge computing domains. In-memory computing (IMC) addresses this by integrating computation within memory units, minimizing data transfer and offering greater energy efficiency and throughput for DNNs workloads. It is gradually emerging as a promising solution.Within this framework, the macro-level IMC component serves as a bridge between device circuits and system-level on-chip architectures. It organizes memory and compute units into scalable modules, which are then integrated and scheduled by higher-level systems. This structure is crucial for advancing IMC beyond individual devices and enabling its practical implementation.

In light of the aforementioned background, this paper focuses on the current macro-level designs of in-memory computing components and energy efficiency optimization approaches in the context of deep neural network inference. Specifically, we examine the array architecture, functional units, and dataflow control of device macros across three key aspects: computational accuracy optimization, numerical representation refinement, and computational scale enhancement. Diverging from existing surveys that primarily emphasize novel memory cells at the material level or algorithmic techniques such as quantization and sparsity, this study specifically concentrates on design methodologies at the device macro level, aiming to bridge circuit implementation and system application.

The following sections organize the remainder of this paper. Section II surveys background on deep language models. Section III introduces a mixed floating-point/integer computation macro that enables genuine in-memory dual-mode operation supporting both floating-point and integer arithmetic. Section IV presents an outlier quantization macro for hybrid computation using low-bit integers and limited floating-point outliers. Section V describes a sparse compression format computation macro supporting in-macro computation for multiple mainstream compression formats. The paper concludes with potential future research directions.

2. Background

Floating-point arithmetic provides large dynamic range and high numerical precision beneficial for deep neural network training and high-precision inference. In contrast, integer arithmetic is widely adopted in quantized inference for its energy efficiency and low power consumption. A floating-point number comprises an exponent and a mantissa, and its operations involve exponent alignment and mantissa calculation, incurring significant power and area overhead. While each data type excels in specific scenarios, most hardware architectures fail to support both efficiently. Thus, a unified architecture supporting both floating-point and integer computation is highly desirable.

Research on FP-based CIM architectures has led to a variety of design directions. One representative example is a mixed-domain FP SRAM-CIM structure, where exponential-related operations are handled directly inside the memory array. Other studies have explored dual-bitcell configurations that integrate dedicated floating-point computation modules, and some works rely on Mitchell's logarithmic approximation to simplify multiplication. While these strategies attempt to improve flexibility and efficiency, they also bring new drawbacks. Mixed-domain implementations typically suffer from high power usage in the readout path, exponent processing in dual-bitcell designs often occurs outside the CIM array, adding extra memory access overhead, and approximation-based methods inevitably sacrifice numerical accuracy.

As model sizes continue to grow, especially in large language models, researchers have turned to low-bit quantization to reduce storage demands. However, simply applying aggressive quantization to these models has not produced satisfactory results. It has been observed that a small set of unusually large weights—referred to as outliers—are largely responsible for the accuracy loss after quantization. To address this issue, outlier-aware quantization (OAQ) techniques were introduced. These methods

DOI: 10.54254/2755-2721/2026.TJ30682

retain outlier values in floating-point form while quantizing the remaining parameters into compact low-bit integers. This hybrid representation has proven effective in maintaining accuracy close to full-precision models, making OAQ a widely adopted solution for deploying LLMs efficiently.

Sparse computing provides another path toward reducing computation and memory usage by exploiting the abundance of zero or near-zero values in neural network weights. During inference, many parameters contribute little to the final output and can be safely skipped or compressed. Removing operations involving these insignificant values and storing weights in compressed formats can greatly reduce computational effort and memory bandwidth. Sparsity can be categorized as static, determined before deployment and encoded using structured compression schemes, or dynamic, where sparsity emerges at runtime depending on inputs or intermediate activations.

3. Mixed Calculation Macro for Floating Point and Integer[1]

3.1. Background

Current FP-CIM systems exhibit several limitations when executing deep neural networks. One category of FP-CIM relies on additional exponent-mantissa alignment circuits and INT-FP quantization circuits, which necessitate extra offline data conversion. As such, these implementations do not fully embody the conceptual essence of true FP-CIM[2]. Another category introduces dedicated exponent summation arrays and alignment circuitry, but such designs incur substantial area and power overheads[3]. Moreover, existing CIM macros often struggle to simultaneously support both floating-point (FP) and integer (INT) operations across diverse application scenarios. Several persistent challenges remain: the exponent array consumes excessive energy in INT mode; readout circuits exhibit high power consumption; mantissa alignment may exceed the bit-width capacity of the CIM array; and large-scale adder trees impose non-negligible area and energy burdens. Therefore, there is a compelling need for a chip architecture capable of performing both FP and INT computations while delivering optimal energy efficiency and high hardware utilization under typical operating conditions.

3.2. Technique route

This paper presents a 28nm dual-macro architecture capable of supporting both floating-point and integer in-memory computing. The proposed design achieves high-precision deep neural network inference while significantly enhancing energy efficiency.

Fig.1 illustrates a dual-macro architecture consisting of two Computing-in-Memory (CIM) macros, the Exponential CIM (ECIM) and Mantissa CIM (MCIM)—interconnected through an Exponent-Mantissa Alignment (EMA) module. The ECIM macro integrates a 64 kb computing array with Dual-Sided Local Computing Units (DSLCCs), an exponential input buffer, exponential adder (EA), accumulator selector (ACC SEL), MAC input buffer, successive approximation analog-to-digital converter (BuADC), word-line driver, read/write I/O (WRIO) unit, and timing controller. The MCIM macro contains a 64 kb computing array with One-Sided Local Computing Units (OSLCCs), a combined MAC and aligned mantissa input (AM IN) buffer, feature-level addition tree (FAT), floating-point quantization unit, word-line driver, write I/O (WIO) unit, and timing controller. Bridging both macros, the EMA module comprises a mantissa input (MAN IN) buffer, exponent comparator, and mantissa alignment unit, all governed by a top-level controller. This ECIM+MCIM architecture supports parallel analog MAC and digital addition within the ECIM macro alongside digital MAC operations in the MCIM macro.

As illustrated in Fig. 2(a), under FP mode all three modules operate concurrently: the ECIM handles exponential accumulation, the EMA aligns mantissae, and the MCIM conducts mantissa MAC

DOI: 10.54254/2755-2721/2026.TJ30682

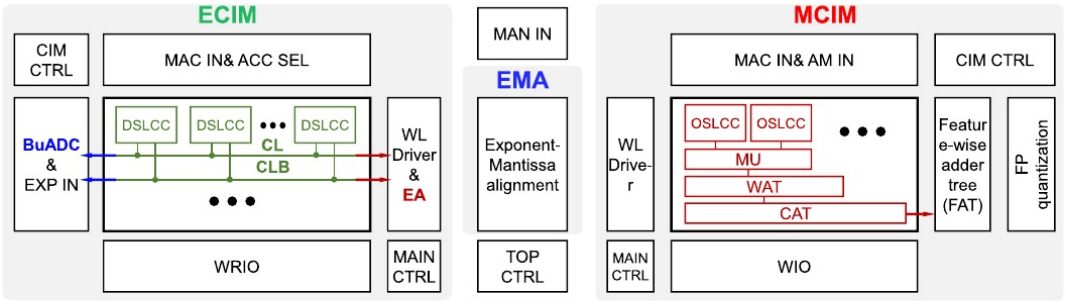

Figure 1: Proposed INT and FP supporting dual-CIM macrostructure

operations and quantization. The ECIM receives four 8-bit input exponents per cycle while its ACC-SEL module selects corresponding weight exponents across 32 DSLCCs. The ECIM array computes digit-wise XOR between input and weight exponents and sends results to the EA, generating four exponent sums per cycle. After four cycles, 16 exponent sums are produced and passed to the EMA, which identifies the maximum exponent and aligns all mantissae accordingly. The MCIM's AM IN module then receives 16 aligned mantissae, processes MAC operations over 16 inputs and weights through 32 output channels, and applies FP quantization to yield 32 final FP results. As illustrated in Fig.2(b), in INT mode, the ECIM and MCIM operate independently while the EMA remains de-activated. The ECIM macro facilitates INT8 multiply-accumulate (MAC) operations with large accumulation lengths by leveraging analog-domain computation. It supports MAC operations between input (IN) and weight (W) for 8 output channels (och) with accumulation lengths configurable from 128 to 1024 through extended computation cycles. For an accumulation length of 128, 8-bit IN data is fed through the feature input buffer into the computing array and processed with stored weights over 8 cycles. When the accumulation length exceeds 128, the 8-bit IN data is input over multiple MAC cycles, with 1 bit of IN processed per cycle. Each MAC cycle consists of AL/128 + 2 cycles, inclusive of one quantization cycle and one reset cycle. The MCIM macro operates in the digital domain, performing MAC operations on 16 INs and Ws for 32 output channels. The 8-bit IN data is loaded into the array over four or five cycles, depending on whether the data is unsigned or signed. In each cycle, INT MAC operations are executed using 16×1-bit or 2-bit INs and 8-bit Ws. Intermediate results are accumulated to produce 13-bit signed or unsigned INT MAC values. Upon completion, the computation yields 32 output values of 17-bit signed or unsigned INT MAC results.

The presented chip is fabricated in a 28nm CMOS logic process and occupies a total area of 0.212 mm². It comprises two 64Kb macros, namely ECIM and MCIM. As illustrated in Figure 3, the chip was evaluated on an FPGA platform. In BF16 mode, it achieves an energy efficiency of 45.34 TFLOPS/W at 0.6 V and a peak area efficiency of 0.929 TFLOPS/mm² at 0.9 V. Under INT8 mode, the ECIM macro demonstrates an energy efficiency ranging from 115.2 to 251.7 TOPS/W and an area efficiency between 1.436 and 4.475 TOPS/mm² across the 0.6–0.9 V supply voltage range. Meanwhile, the MCIM macro exhibits an energy efficiency from 122.18 to 280.1 TOPS/W and an area efficiency ranging from 0.600 to 2.015 TOPS/mm² within the same voltage range.

DOI: 10.54254/2755-2721/2026.TJ30682

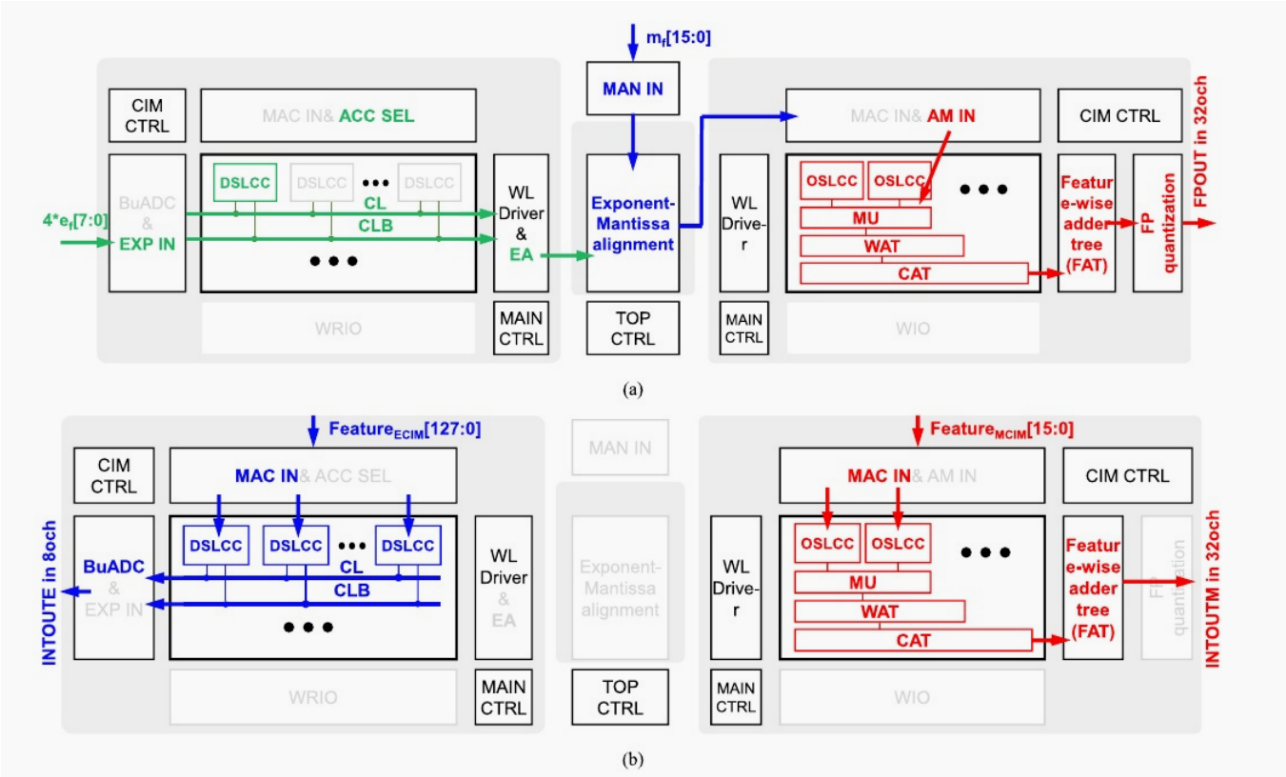

Figure 2: Proposed dual-CIM workflow. (a) FP mode. (b) INT mode

3.3. Conclusion

The proposed 28 nm dual-macro architecture achieves leading results in energy efficiency and utilization. The dual CIM macros realize a FoM (Figure of Merit), calculated as input precision × weight precision × output ratio × energy efficiency, exceeding that of prior SRAM-based CIM works by a factor of 2.32 or more. Specifically, the ECIM and MCIM architectures achieve FoM values higher than previous INT SRAM-CIM works by factors of 6.80 and 10.54, respectively.

Several limitations require further investigation. At larger scales device migration and bandwidth may impose constraints while the dual-macro architecture demands more complex fabrication with circuit stability still lacking thorough validation. Approximate computing remains heavily dependent on specific training methods limiting its generalizability. Future work should prioritize model quantization and sparsity compression to improve the applicability of Computing-in-Memory in deep neural networks.

4. Outlier Perception Calculation Macro[4]

4.1. Background

Developing efficient outlier-aware (OA) computation within CIM architectures remains challenging. Current designs often use macros that toggle between full-integer (INT) and full-floating-point (FP) modes, which cannot process element-wise random outliers[5][6][7]. Other methods use separate INT and FP kernels, leading to high hardware cost and low energy efficiency[8][9], highlighting the need for dedicated OA architectures.

DOI: 10.54254/2755-2721/2026.TJ30682

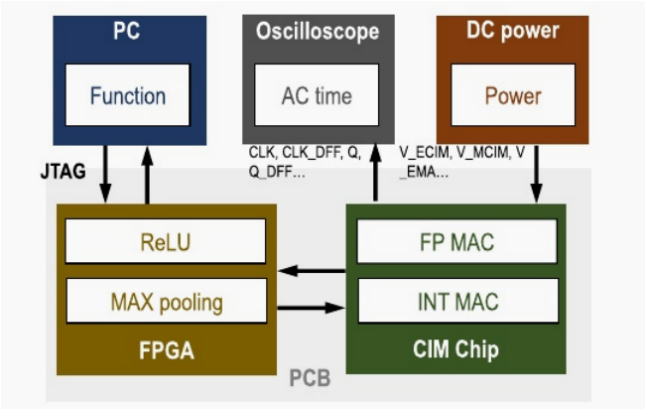

Figure 3: Test platform of proposed work.

This paper presents an OA-CIM architecture enabling mixed-precision computation of BF16 outliers and INT4 normal values via a fixed-index correspondence lookup table within the MAC circuit. The algorithm-hardware co-design achieves memory alignment while balancing precision and storage efficiency. A 22nm prototype OA-CIM macro was fabricated and its energy efficiency characterized.

4.2. Technique route

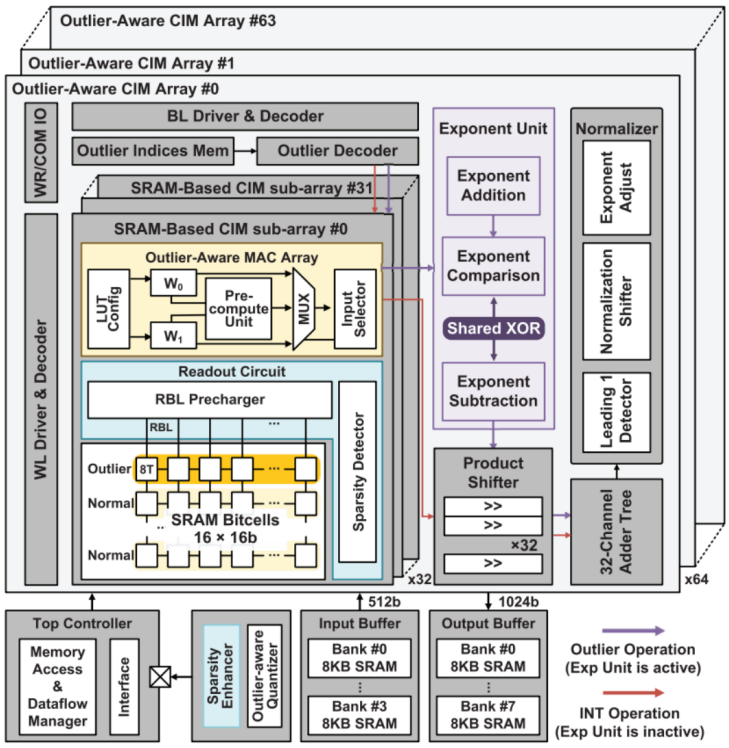

Figure 4: Overall architecture of OA-CIM macro

DOI: 10.54254/2755-2721/2026.TJ30682

A central innovation is the integration of the OA-CIM macro with the Outlier-Normal Value Alignment Quantization (ONAQ) algorithm. As shown in Fig. 4, the OA-CIM macro comprises 64 CIM arrays each sized 512×16b. Each array contains 32 pairs of CIM sub-arrays and MAC arrays, an outlier decoder with index memory, an exponent processing unit, a product shifter, a channel-level adder tree, and a normalization unit. Each CIM array includes 16×16b cells storing either four INT4 values or one BF16 outlier.

The design enables seamless exponent and mantissa processing without pipeline stalls, maximizing computational efficiency and throughput. Weights are stored in CIM arrays. The outlier decoder configures the MAC and exponent units via index memory, fetches MAC values (MACV) and exponent differences. The product shifter aligns MACV by exponent before passing results to the adder tree and normalization—all in one fully parallelized cycle.

This paper also proposes ONAQ, an outlier adjustment framework inspired by [10]. Based on the GPTQ method [11], ONAQ selects outliers per iteration by sensitivity: the squared difference between the calibrated weight matrix and a version where one element is kept in BF16, three adjacent values set to zero, and others quantized to INT4. Elements exceeding a sensitivity threshold are tagged as outliers. Each 32-element block allows at most one outlier; extras are discarded. A Distribution-Oriented Weight Encoding (DOWE) scheme is also introduced, using a sparsity-aware readout circuit to improve efficiency via bit-level sparsity. To avoid large value overflow in DOWE, which risks precision loss, thus integrating it with ONAQ.

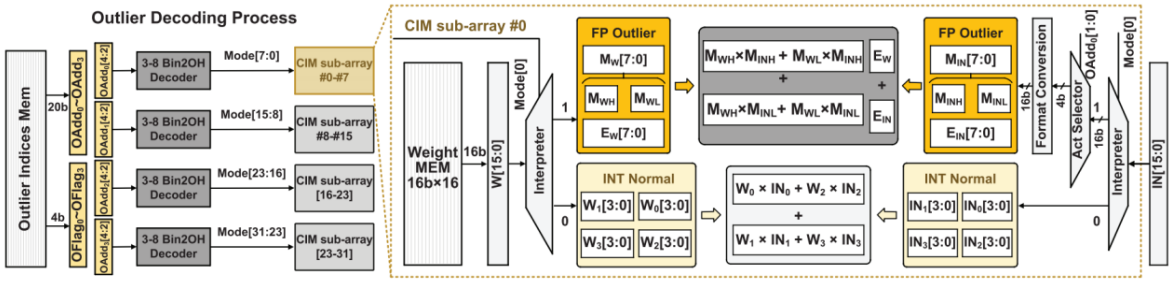

Figure 5: Outlier-normal decoding process and mode configuration

Fig.5 illustrates the decoding and configuration process for mixed outlier-normal computation. Each set of 32 CIM sub-arrays processes four blocks of 32 values each, accompanied by four 6-bit outlier indices (OAdd–OAdd and OFlag–OFlag). Since ONAQ limits outliers to the top 1/32 of values, each block requires only a single 6-bit index. This index consists of a 5-bit address (OA-dd) to locate the outlier within the block and a 1-bit flag (OFlag) to indicate its presence.

4.3. Conclusion

The ONAQ method applied to OPT models delivers accuracy that is only 0.9% lower than BF16 and 6.0% higher than INT4, while achieving memory efficiency close to that of BF16 precision.

Fabricated on a 22nm process, the prototype operates within a voltage range of 0.6V to 1.1V and a frequency range of 55MHz to 270MHz. At 0.6V and 55MHz, it reaches peak energy efficiencies of 346.6 TOPS/W (INT4), 74.9 TFLOPS/W (BF16), and 249.5 TFLOPS/W (in Outlier mode).

DOI: 10.54254/2755-2721/2026.TJ30682

5. Sparse Perception Computing Macro

5.1. Background

Within the domain of static sparsity, various in-memory computing macros have been developed to meet the high energy efficiency requirements of edge devices. Examples such as the DIMC 28nm approximate arithmetic macro[13] and the D6CIM macro[14] successfully reduce energy consumption via approximate arithmetic and digital in-memory fixed-point computing. Yet unlike ASICs, these designs seldom incorporate sparse compression to maximum storage benefits and computation skipping. Previous attempts at weight compression using binary mapping and custom formats [15] show limited effectiveness. This paper[12] presents SP-IMC, an in-memory computing macro that stores and processes coefficient-compressed weights directly. It simultaneously supports three major coefficient compression formats with adjustable precision.

5.2. Technique route

The SP-IMC design starts from the organization of its MAC computation array, which is structured into 16 column groups, and each of these is further divided into 32 row groups (RG). Together with the array, the macro also integrates an accumulation logic (AL) module and a shift-and-accumulate stage. Within each row group, there are two 8-bit computation units along with a multiply–decode–compare (MDC) element. The 8-bit units rely on four 10T bitcells to store weights and an additional four 6T cells that hold the compressed index information. The accumulation logic incorporates an adder tree that is partitioned into two parallel branches, each operating on 32-bit input data, and it supports selectable accumulation widths of either 11 or 14 bits. A single 10T bitcell carries out multiplication between a 2-bit activation and a 1-bit weight, which helps reduce the throughput bottleneck typically associated with bit-serial processing when higher-precision activations are required. The upper and lower sets of four 10T cells are grouped to generate two partial multiplication results corresponding to 4-bit activations and 2-bit weights, which are processed by the shift multiplier in the MDC block. Partial products from each row group are aggregated through the adder tree and transferred cycle-by-cycle at 2-bit granularity to the shift-and-accumulate unit for alignment and precision compensation, completing the equivalent MAC operation. The sparse computation design supports three column-wise compression formats: run-length, coordinate, and N:M sparsity. Each row group includes an RL/COO decoding block, while MDC comparators handle index comparisons. Additionally, SP-IMC implements a set of dedicated hardware logics to handle edge cases in compressed weights. In COO mode, the decoding block directly transmits the bit-cell index, and a local counter generates, in each cycle, the row index corresponding to the weight stored in that column. In RL mode, it computes run-length indices (RLC) based on the previous column group, and an index generator produces zero counts between non-zero weights. The N:M sparsity pattern is implemented using the COO format. Then a comparator decides whether to use the partial product of the row group. To address non-uniform sparsity where matrix row elements span multiple column groups, a spillover counter adjusts indices and a spillover accumulator corrects computation results.

5.3. Conclusion

Fabricated in a 28 nm CMOS process, the SP-IMC architecture achieves an energy efficiency of 8.4–36.6 TOPS/W under fully non-sparse operation (where only one index per column activates all bit-cells and adder tree nodes) and 7.5–115.3 TOPS/W under sparse conditions, measured between 4-bit activations and 4-bit weights at 25°C with a supply voltage ranging from 0.57 V to 1.2 V and

DOI: 10.54254/2755-2721/2026.TJ30682

an input toggle rate of 25%. With respect to the proposed FoM, which is defined as ((TOPS/W) × (TOPS/mm²) × (# of weights stored per kB)), the design shows up to a 5.9-fold improvement compared to the previous state-of-the-art. SP-IMC shows that in inference tasks taking advantage of sparsity can greatly improve energy efficiency.

6. Conclusion

With the rapid growth of large language models and the increasing use of AI on edge devices, deep neural networks are facing serious energy constraints. In-memory computing has emerged as a promising way to address this issue, as it integrates data storage and computation within the same hardware, cutting down the heavy data movement that normally consumes a lot of power. This review summarizes three representative categories of in-memory computing macros. The first type combines floating-point and integer computation, allowing a single chip to switch between high-precision and energy-efficient modes, which helps improve both hardware utilization and overall energy performance. The second type, outlier-aware quantization macros, aims to maintain accuracy while reducing storage requirements by keeping only a small number of outlier values in floating-point form and converting most parameters into low-bit integers. The third type focuses on sparse computing (SP-IMC), making use of different sparsity formats to skip unnecessary operations, significantly lowering computational workload and power consumption. These macro designs each involve their own compromises in terms of precision, efficiency, throughput, and circuit complexity, suggesting that future macro development should be tailored to specific application needs rather than pursuing a single universal design. The internal architecture and dataflow organization of a macro play a key role in its performance. By examining various design strategies, researchers can better translate neural network requirements into hardware implementations, facilitating integration into current computing systems and improving energy efficiency. Looking ahead, combining mixed-precision techniques with sparsity exploitation appears to be a promising direction for designing memory-centric computing components capable of supporting large-scale model inference as well as energy-limited edge applications.

Acknowledgment

Jihong Liu and Taoyi Zhang contributed equally to this work and should be considered co-first authors.

References

[1] A. Guo, X. Dong, F. Dong, D. Li, Y. Zhang, J. Zhang, J. Yang, and X. Si, "A 28 nm 128-kb Exponent-and Mantissa-Computation-In-Memory Dual-Macro for Floating-Point and INT CNNs," *IEEE Journal of Solid-State Circuits*, pp. 1–16, 2025.

[2] A. Guo, C. Xi, F. Dong, X. Pu, D. Li, J. Zhang, X. Dong, H. Gao, Y. Zhang, B. Wang, J. Yang, and X. Si, "A 28-nm 64-kb 31.6-TFLOPS/W Digital-Domain Floating-Point-Computing-Unit and Double-Bit 6T-SRAM Computing-in-Memory Macro for Floating-Point CNNs," *IEEE Journal of Solid-State Circuits*, vol. 59, no. 9, pp. 3032–3044, 2024.

[3] P.-C. Wu, J.-W. Su, L.-Y. Hong, J.-S. Ren, C.-H. Chien, H.-Y. Chen, C.-E. Ke, H.-M. Hsiao, S.-H. Li, S.-S. Sheu, W.-C. Lo, S.-C. Chang, C.-C. Lo, R.-S. Liu, C.-C. Hsieh, K.-T. Tang, and M.-F. Chang, "A 22nm 832Kb Hybrid-Domain Floating-Point SRAM In-Memory-Compute Macro with 16.2–70.2TFLOPS/W for High-Accuracy AI-Edge Devices," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, 2023, pp. 126–128.

[4] S. He, H. Zhu, H. Zhang, Y. Ma, Z. Chen, M. Li, D. Zhai, C. Chen, Q. Liu, X. Zeng, and M. Liu, "A 22-nm 109.3-to-249.5-TFLOPS/W Outlier-Aware Floating-Point SRAM Compute-in-Memory Macro for Large Language Models," *IEEE Journal of Solid-State Circuits*, pp. 1–14, 2025.

[5] A. Guo, X. Si, X. Chen, F. Dong, X. Pu, D. Li, Y. Zhou, L. Ren, Y. Xue, X. Dong, H. Gao, Y. Zhang, J. Zhang, Y. Kong, T. Xiong, B. Wang, H. Cai, W. Shan, and J. Yang, "A 28nm 64-kb 31.6-TFLOPS/W Digital-

DOI: 10.54254/2755-2721/2026.TJ30682

Domain Floating-Point-Computing-Unit and Double-Bit 6T-SRAM Computing-in-Memory Macro for Floating-Point CNNs," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, 2023, pp. 128–130.

[6] W.-S. Khwa, P.-C. Wu, J.-J. Wu, J.-W. Su, H.-Y. Chen, Z.-E. Ke, T.-C. Chiu, J.-M. Hsu, C.-Y. Cheng, Y.-C. Chen, C.-C. Lo, R.-S. Liu, C.-C. Hsieh, K.-T. Tang, and M.-F. Chang, "34.2 A 16nm 96Kb Integer/Floating-Point Dual-Mode-Gain-Cell-Computing-in-Memory Macro Achieving 73.3–163.3TOPS/W and 33.2–91.2TFLOPS/W for AI-Edge Devices," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, vol. 67, 2024, pp. 568–570.

[7] Y. Yuan, Y. Yang, X. Wang, X. Li, C. Ma, Q. Chen, M. Tang, X. Wei, Z. Hou, J. Zhu, H. Wu, Q. Ren, G. Xing, P.-I. Mak, and F. Zhang, "34.6 A 28nm 72.12TFLOPS/W Hybrid-Domain Outer-Product Based Floating-Point SRAM Computing-in-Memory Macro with Logarithm Bit-Width Residual ADC," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, vol. 67, 2024, pp. 576–578.

[8] R. Guo, L. Wang, X. Chen, H. Sun, Z. Yue, Y. Qin, H. Han, Y. Wang, F. Tu, S. Wei, Y. Hu, and S. Yin, "20.2 A 28nm 74.34TFLOPS/W BF16 Heterogenous CIM-Based Accelerator Exploiting Denoising-Similarity for Diffusion Models," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, vol. 67, 2024, pp. 362–364.

[9] S. Yan, J. Yue, C. He, Z. Wang, Z. Cong, Y. He, M. Zhou, W. Sun, X. Li, C. Dou, F. Zhang, H. Yang, Y. Liu, and M. Liu, "A 28-nm Floating-Point Computing-in-Memory Processor Using Intensive-CIM Sparse-Digital Architecture," *IEEE Journal of Solid-State Circuits*, vol. 59, no. 8, pp. 2630–2643, 2024.

[10] C. Guo, J. Tang, W. Hu, J. Leng, C. Zhang, F. Yang, Y. Liu, M. Guo, and Y. Zhu, "OliVe: Accelerating Large Language Models via Hardware-friendly Outlier-Victim Pair Quantization," in *Proc. 50th Annu. Int. Symp. Computer Architecture (ISCA)*, 2023, pp. 1–15.

[11] E. Frantar, S. Ashkboos, T. Hoefler, and D. Alistarh, "GPTQ: Accurate Post-Training Quantization for Generative Pre-trained Transformers," arXiv preprint arXiv:2210.17323, 2023.

[12] A. Sridharan, F. Zhang, J.-S. Seo, and D. Fan, "SP-IMC: A Sparsity Aware In-Memory-Computing Macro in 28nm CMOS with Configurable Sparse Representation for Highly Sparse DNN Workloads," in *Proc. IEEE Custom Integrated Circuits Conf. (CICC)*, 2024, pp. 1–2.

[13] D. Wang, C.-T. Lin, G. K. Chen, P. Knag, R. K. Krishnamurthy, and M. Seok, "DIMC: 2219TOPS/W 2569F2/b Digital In-Memory Computing Macro in 28nm Based on Approximate Arithmetic Hardware," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, vol. 65, 2022, pp. 266–268.

[14] J. Oh, C.-T. Lin, and M. Seok, "D6CIM: 60.4-TOPS/W, 1.46-TOPS/mm^2, 1005-Kb/mm^2 Digital 6T-SRAM-Based Compute-in-Memory Macro Supporting 1-to-8b Fixed-Point Arithmetic in 28-nm CMOS," in *ESSCIRC 2023 – IEEE 49th European Solid State Circuits Conference (ESSCIRC)*, 2023, pp. 413–416.

[15] J. Yue, C. He, Z. Wang, Z. Cong, Y. He, M. Zhou, W. Sun, X. Li, C. Dou, F. Zhang, H. Yang, Y. Liu, and M. Liu, "A 28nm 16.9–300TOPS/W Computing-in-Memory Processor Supporting Floating-Point NN Inference/Training with Intensive-CIM Sparse-Digital Architecture," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, 2023, pp. 1–3.