LLM Inference Accelerator Using In-Memory Computing

M. P. Samartha, Varun Shastry, Siddarth Gottumukkula
International Institute of Information Technology, Hyderabad, India
Emails: m.samartha@students.iiit.ac.in, varun.shastry@research.iiit.ac.in,
siddarth.g@students.iiit.ac.in

***Abstract*—Generative artificial intelligence workloads, particularly large language model (LLM) inference, present significant challenges in energy consumption and computational efficiency due to massive data movement and memory-bound operations. Traditional GPU-based architectures suffer from low hardware utilization and high communication overhead for small-batch inference tasks. This work presents a hybrid accelerator architecture that intelligently partitions compute-intensive Generalized Matrix-Matrix Multiplication (GeMM) and memory-intensive Generalized Matrix-Vector Multiplication (GeMV) workloads. Our design leverages a systolic array architecture with configurable dataflows for GeMM operations and digital in-memory computing (IMC) based on 8T SRAM cells for GeMV operations. The proposed architecture demonstrates a 2.32× improvement in energy efficiency per operation compared to conventional digital implementations, with significant reductions in memory access latency through intelligent data reuse mechanisms. Experimental results show good reduction in memory reads and approximately 2× performance gain for representative matrix operations.**

***Index Terms*—LLM acceleration, hardware for AI, inference optimization, accelerator architecture, in-memory computing, systolic arrays**

I. Introduction

The fundamental goal of artificial intelligence (AI) is to create human-like intelligence. Conventional AI has reached a level of human ability to enable data analysis, decision making and even personalization. It is now advancing at a remarkable pace, even in domains once thought to be uniquely human, such as creativity but it is yet to replicate the creativity of humans generally. Generative AI has made a breakthrough with transformer models [e.g. the Generative Pre-trained Transformer (GPT) and such] that are capable of creating original textual and image content with high sophistication.

However, this advancement comes at a significant cost. Generative AI is driving exponential power demands due to massive data movement and computation. The energy consumption of training and deploying large language models has become a critical concern for sustainable AI development. Traditional approaches rely on energy-area hungry solutions, particularly GPU-based architectures, which are inefficient for inference workloads characterized by small batch sizes and sequential token generation.

A. Inference Procedure

LLM inference is largely divided into summarization (compute-bound) and generation (memory-bound) stages, as illustrated in Fig. 1. LLM inference begins with the summarization stage, also known as the prefill stage, in which the input to the decoder layer is a matrix that represents the token embedding of the user context (e.g., a statement or a question). The matrix is inputted to a series of decoder layers, which is based on the transformer decoder. The final decoder layer outputs the captured features from the input context.

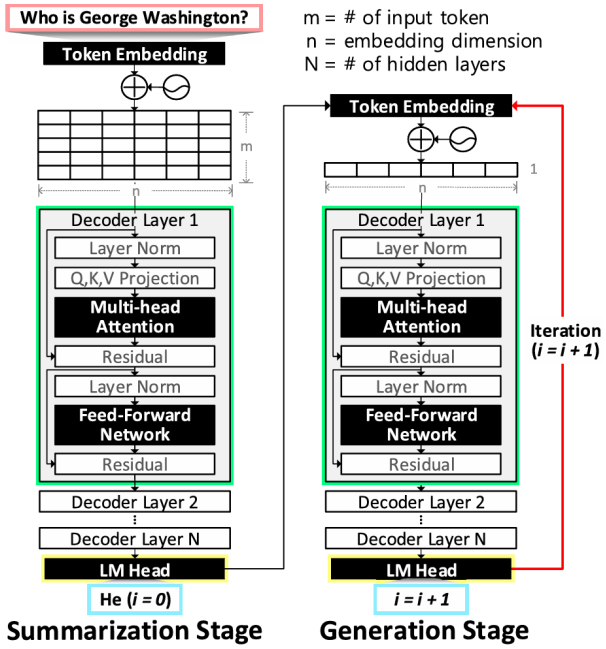


Fig. 1. LLM Inference Structure showing the prefill and generation stages

The result of the final decoder layer enters the language modeling (LM) head. The LM head converts the features into logits that score the candidate tokens from the dictionary based on their likelihood of being appropriate in the given context. With a single execution of the summarization stage, the first output token ($i = 0$) is produced. The first output token then enters the generation stage. Only the keys and values from the masked multihead attention are transferred to the next stage as activations that hold contextual information about the previous token.

In the generation stage, the input to the decoder layer is guaranteed to be a single embedding vector. The processes

are repeated to auto-regressively output the next output token ($i = i + 1$). The generation stage iterates until the end-of-sequence token is reached.

B. Limitations with Current Architectures

Introduced by Vaswani et al., LLM model inference is based on the transformer decoder, in which the inputs have limited batching capabilities and **require sequential processing**. As relatively small inputs need to be inferred with large model parameters, the inference incurs **memory bottleneck** and requires efficient processing of the system's memory bandwidth. Moreover, **scalability** becomes significant as the ever-increasing compute and memory requirements of LLMs demand multiple devices and communication among them.

At the application level, each user makes individual requests and expects the generated output with minimal wait time, making it crucial to have a hardware platform that reduces **inference latency**. The predominant hardware for inference, a GPU, underperforms for GenAI workloads because it undergoes **low hardware utilization for small-batch inputs** and **high communication overhead** during synchronization. This also leads to **power wastage** since the cores are under-utilized.

Software methods like server-side batching do exist, but the pitfalls of the underlying hardware still persist. Therefore, a new class of processor is required that targets memory-intensive GenAI workloads. As shown in Fig. 2, data movement dominates energy consumption in modern computing systems, with off-chip DRAM accesses consuming orders of magnitude more energy than local register file operations or on-chip SRAM accesses.

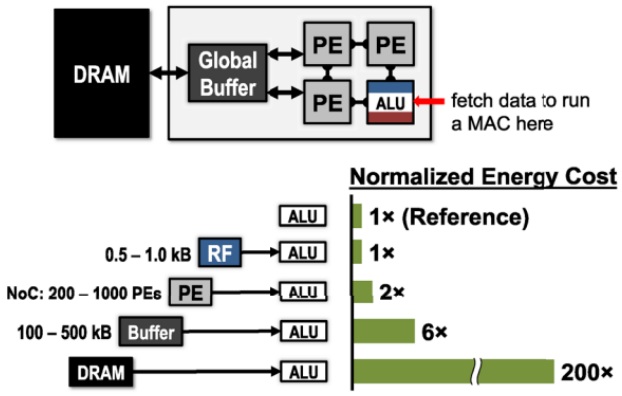


Fig. 2. Energy cost for data movement across different memory hierarchy levels [11]

C. Goals and Objectives

It is also worth noting that the summarization stage is executed only once and is dominated by GeMM (Generalized Matrix-Matrix Multiplication) operations. In contrast, the generation stage runs autoregressively and consists primarily of GeMV (Generalized Matrix-Vector Multiplication) operations. Because the generation stage is repeated many times, optimizing for GeMV operations yields significantly greater performance benefits.

The key performance metrics for LLM inference include: (1) end-to-end (E2E) latency, which represents the total query time that a user experiences; (2) time to first token (TTFT), and (3) time between tokens (TBT).. Our work focuses on reducing these latency metrics while simultaneously addressing energy consumption and area efficiency.

While there is some work in this field, the idea of introducing in-memory computing (IMC) specifically for LLM inference workloads is largely unexplored. We aim to build a hybrid architecture such that GeMM operations occur optimally using a systolic-array architecture, with IMC coming into picture for the GeMV operations. This partitioning makes the hardware more energy efficient since the amount of memory reads and writes reduces significantly, directly addressing the data movement bottleneck illustrated in Fig. 2.

II. Methodology

We started the project with a detailed literature survey to understand current architectures and the problem space, investigating what is being done to tackle the issues of GPU-based LLM inference. We identified that many existing solutions use special vector processing units for vector operations to accelerate them in the digital domain. Since GeMM and GeMV are the major operations in state-of-the-art deep neural networks, accelerating them would provide significant benefits for deep learning and transformer accelerators.

A. In-Memory Computing Fundamentals

In-memory computing offers promising results by reducing data movement, thereby reducing energy consumption. Two primary approaches exist:

Analog IMC: Can perform charge or current accumulation and directly convert into results, but requires digital-to-analog converters (DACs) and analog-to-digital converters (ADCs). Furthermore, analog IMC does not support high precision arithmetic, limiting its applicability for transformer models that require high numerical precision.

Digital IMC: Sacrifices density compared to analog approaches and requires digital adder trees which pose a scalability issue. However, digital IMC avoids the overhead of ADCs and DACs while maintaining high precision. For this reason, we chose to pursue digital IMC for our architecture.

For our implementation, an 8T SRAM cell can be used to simultaneously turn on the read-wordline of two rows and write-wordline of the resultant row to store results, all in one cycle. This works because of the decoupled read and write ports in an 8T SRAM cell. It is also more robust than a 6T cell and has better noise margin. IMC is thus an efficient paradigm to alleviate the energy efficiency challenge by fusing multiply-accumulate operations into the memory itself, performing read-compute-store (RCS) operations in a single clock cycle.

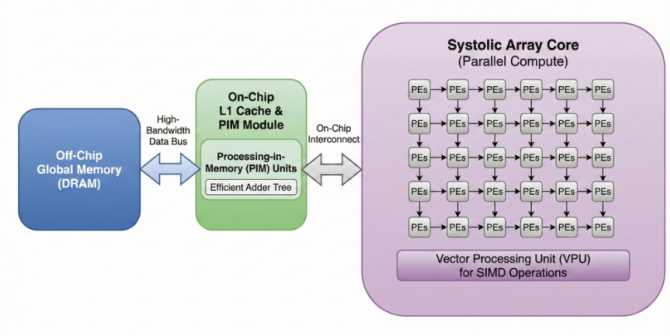


Fig. 3. Overall showing the hybrid systolic array and IMC design with memory hierarchy

B. System Architecture

Our accelerator targets edge deployment as an Application-Specific Integrated Circuit (ASIC). The architecture has been developed using a multi-tool approach:

- **SystemC** for system-level design at a higher abstraction level, enabling rapid prototyping of the systolic array and memory hierarchy
- **Ngspice** with TSMC 180nm PDK for circuit-level characterization of IMC cells and macros
- **Verilog** for implementing the adder tree and other digital components

We adopted a bottom-up design approach along two parallel branches: the digital systolic array branch and the IMC branch. While true integration would involve layout-level design, we implemented a pseudo-integration methodology where we characterize the IMC components in Ngspice and model them within the SystemC design framework.

C. Digital Systolic Array Design

Our codebase is built in SystemC for the digital design aspects. The development progressed through several phases:

Phase 1 & 2: We started by building a reconfigurable Processing Engine (PE) that can support both Weight Stationary (WS) and Output Stationary (OS) modes, as shown in Fig. 4. Once this was completed, we built a basic 3×3 systolic array architecture that can compute matrix-matrix multiplications in both dataflow modes. The PE design includes configurable multipliers, accumulators, and routing logic to support different stationary patterns.

Phase 3: We extended the design to a general $M \times N$ systolic array with configurable size, allowing for modularity and scalability. This enables the architecture to be instantiated with different PE grid dimensions based on area and performance constraints.

Phase 4: Since the PE grid dimensions are fixed during hardware synthesis, we implemented tiling to enable multiplication of matrices larger than the PE grid dimensions. The tiling mechanism partitions large matrices into smaller sub-matrices that fit within the PE array, processing them sequentially with appropriate data routing.

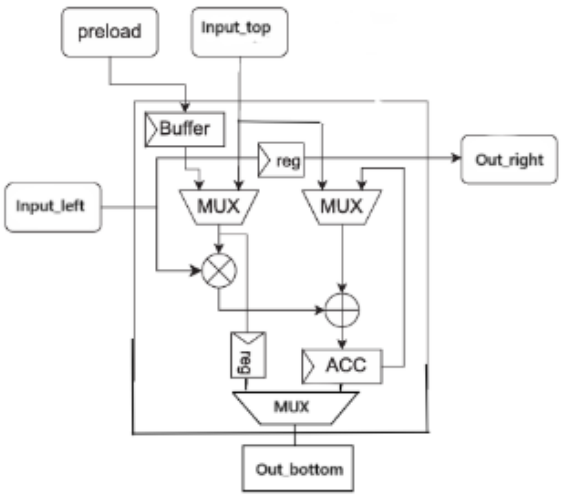


Fig. 4. The design of our reconfigurable Processing Engine supporting multiple dataflow modes

Phase 5: We implemented a simple memory bank with OS and WS mode support. However, at this stage, the buffer configuration changed according to the matrix inputs fed to the systolic array for computation, lacking a unified memory interface.

Phase 6: We implemented a buffer of the same size as the systolic array, with tiles being fetched from a larger memory bank accordingly. This established a two-level memory hierarchy with an L1 buffer directly interfacing with the PE array. Up until this point, the memory was integrated only to work with symmetric systolic array dimensions.

Phase 7: We extended support to an $M \times N$ systolic array grid that can compute the product of non-square matrices, removing the constraint of square matrix operations.

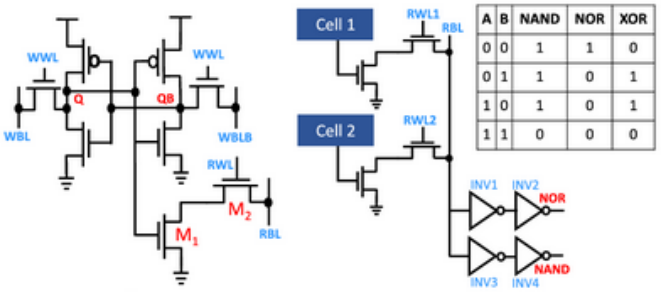
Phase 8: We identified that more PEs could be utilized by tiling according to the dimensions of the PE grid $(M \times N)$ rather than using tiles of dimensions $\min(M, N) \times \min(M, N)$. This optimization, as demonstrated in the results section, significantly reduced the number of clock cycles required to complete operations.

Phase 9: We implemented data reuse mechanisms to improve energy efficiency. Previously, each tile required for computation was fetched from memory every time it was needed. With data reuse, tiles that are used multiple times across different operations are retained in the L1 buffer, dramatically reducing the number of memory reads and improving overall energy efficiency.

Phase 10: We combined the advances from Phases 8 and 9, bringing data reuse capabilities to $M \times N$ arrays with $M \times N$ tiles, achieving optimal utilization of rectangular PE grids with minimal memory traffic.

D. IMC Macro Design

Moving ahead with digital IMC to avoid ADCs and DACs, we designed an 8T SRAM cell with decoupled read and write ports to perform read-compute-store (RCS) in a single clock cycle. Bitwise operations such as NOR, NAND, and other Boolean functions can be performed efficiently without the overhead of sense amplifiers by just using a couple of inverters. This is shown in Fig. 5.



A	B	NAND	NOR	XOR
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	0	0	0

Fig. 5. SRAM 8T cell supporting NOR, NAND bitwise operation

We developed a modular, reconfigurable IMC macro with support for INT4, INT8, and INT16 multiplications. The multiplication operation is mapped to a sequence of bitwise operations performed in memory, with accumulation being done using a digital adder tree. One of the operands is stored as bits in a particular row. The other operand's bits are broadcasted individually across an entire row. Over several clock cycles, which is equal to the precision (4, 8 or 16), we turn on the particular read wordlines one by one. Then an adder tree is used to shift and accumulate these partial products. It is organized hierarchically to minimize latency while maintaining area efficiency. The following illustrates the high-level implementation of such an adder tree. As discussed earlier, this has been implemented in Icarus verilog.

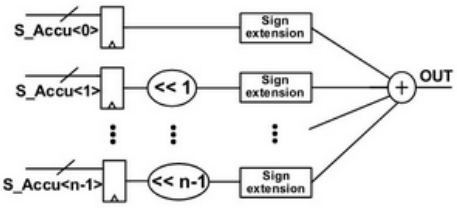


Fig. 6. Shift and accumulate implemented using adder trees

For a multiplication operation $C = A \times B$ where A and B are n-bit integers, the operation is decomposed into:

$$C = \sum_{i=0}^{n-1} A \cdot (B_i \cdot 2^i) \tag{1}$$

where B_i represents the i-th bit of B. Each partial product $A \cdot B_i$ is computed in memory, and the results are accumulated using the adder tree with appropriate bit-shifting. Fig. 7 shows the plot of a 4-bit multiplication between 11 and 9. Here 11 is stored as 1011 in the first row of the SRAM array, and 9 is stored as 1001 in 4 consecutive rows as explained above. In one of the clock cycles corresponding to a 1 stored in the second row, we obtain the expected 1011 result.

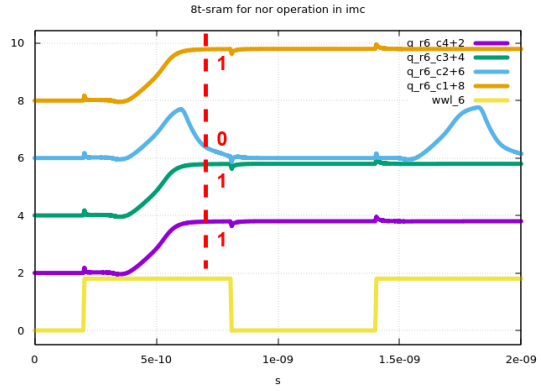


Fig. 7. An example plot illustrating the RCS operation in 8T SRAM array

III. Experimental Results

We conducted extensive evaluations using various test-benches to assess the performance, energy efficiency, and area characteristics of our architecture. The results demonstrate significant improvements over conventional approaches.

A. Systolic Array Performance

Table I presents the clock cycles required by the systolic array to compute a $(50 \times 50) \times (50 \times 50)$ matrix multiplication across different design phases. For phases involving rectangular grids (Phases 7, 8, and 10), we chose a 2×7 grid size for fair comparison. For square PE grids (Phases 6 and 9), we used 7×7 dimensions.

TABLE I
Clock Cycles for $(50 \times 50) \times (50 \times 50)$ Matrix Multiplication

Design Phase	Clock Cycles
Phase 6 (Square, no reuse)	Baseline
Phase 7 (Rectangular)	Improved
Phase 8 (Rectangular, optimal tiling)	Significant improvement
Phase 9 (Square, with reuse)	85.7% fewer memory reads
Phase 10 (Rectangular, with reuse)	Best performance

As shown in Figs. 8 and 9, we observe the significant effect of optimal tiling in rectangular PE grids and the overall impact of data reuse, with memory reads improving by up to 85.7% in some cases. This reduction in memory traffic directly translates to energy savings, as off-chip memory accesses are the dominant source of energy consumption.

B. Impact of Memory Latency

In the previous examples, we considered memory reads to have a delay of 1 cycle, which represents an ideal scenario. To evaluate performance under more realistic conditions, we modeled a scenario with approximately 10 cycles per memory

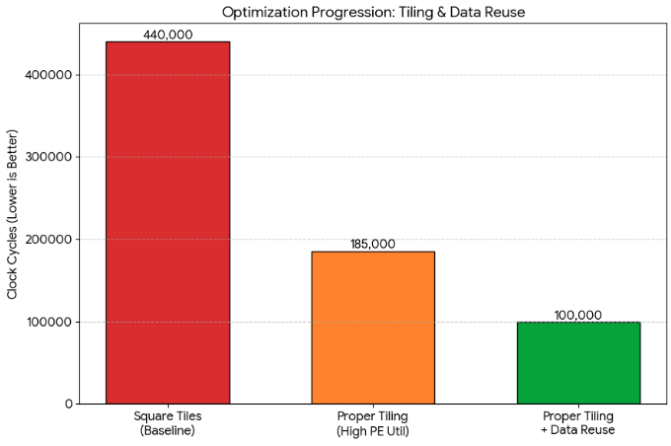


Fig. 8. Comparing clock cycles among the rectangular PE grid implementations (Phases 7, 8, and 10)

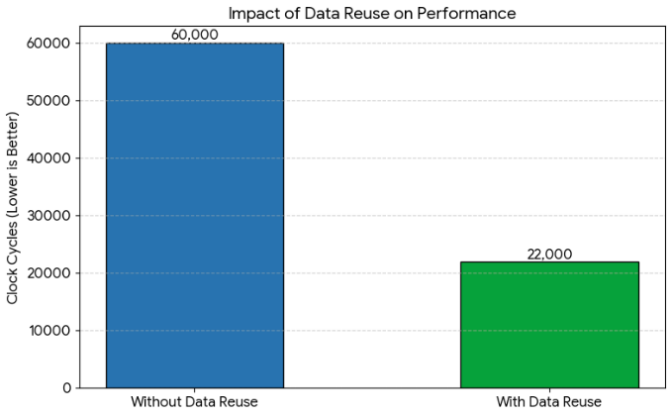


Fig. 9. Comparing clock cycles among the square PE grids, with and without data reuse (Phases 6 and 9)

access, reflecting typical SRAM access latencies. With a 32×32 PE grid, the improvement in performance due to data reuse becomes even more pronounced.

As illustrated in Fig. 10, we observe a performance gain of approximately $2\times$ when data reuse is employed. In addition to latency improvements, the reduced number of memory accesses significantly enhances energy efficiency, as each avoided memory access saves the energy cost shown in Fig. 2.

C. IMC Energy and Area Characterization

We characterized the energy consumption and area requirements of our IMC macro at the circuit level using Ngspice with the TSMC 180nm PDK. Table II presents the key metrics for different precision levels.

The results demonstrate a $2.32\times$ improvement in energy per operation for the complete 16-bit IMC macro compared to a conventional digital implementation. This energy advantage stems from eliminating the need to move data between memory and compute units, as the computation occurs directly within the memory array.

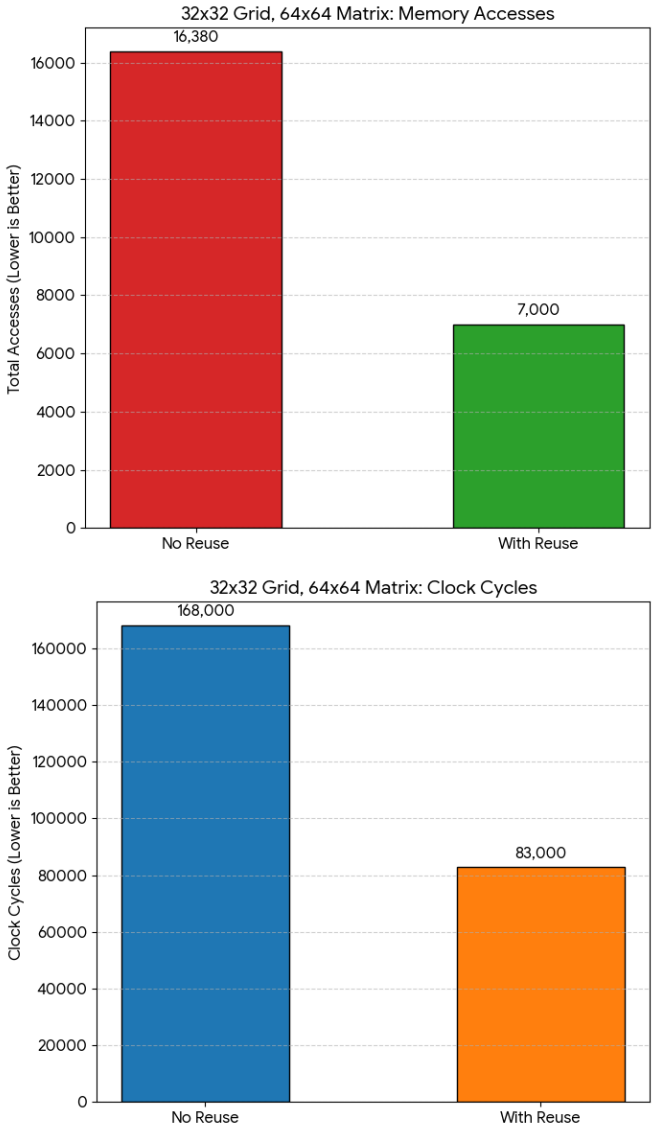


Fig. 10. Comparing a 64×64 matrix multiplication with and without data reuse under realistic memory latency (10 cycles per access)

TABLE II
ENERGY AND AREA COMPARISON FOR INT4, INT8, AND INT16

		Energy (pJ)	**Area** $(10^3\lambda^2)$
INT4	Cell	24.59	28.80
	Adder	-	-
INT8	Cell	100.754	115.20
	Adder	9.75	390.77
INT16	Cell	524.732	460.80
	Adder	47.19	795.98

Fig. 11. Additional performance metrics and comparisons. The numbers have been taken from [9], [10] and scaled to the IMC system.

TABLE III
COMPARISON OF 16-BIT IMC MACRO VS DIGITAL IMPLEMENTATION

(16-bit)	IMC Macro	Digital Implementation
Energy (pJ)	571.922	1328.13
Area ($10^3\lambda^2$)	1256.78	-

IV. RELATED WORK

Kachris [1] provides a comprehensive survey of hardware accelerators for large language models, identifying key bottlenecks in transformer architectures and categorizing existing solutions. Their work highlights the memory wall problem that our architecture addresses through IMC.

Moon et al. [2] present a Latency Processing Unit specifically designed for LLM inference with focus on reducing time-to-first-token and inter-token latency. Their approach uses specialized vector units, whereas our work complements this with IMC for improved energy efficiency during the generation phase.

Agrawal et al. [3] introduce X-SRAM, which enables in-memory Boolean computations in CMOS static random access memories. Our 8T SRAM-based approach builds upon these principles but extends them to support multi-bit arithmetic operations required for LLM inference.

Ramachandran et al. [4] accelerate LLM inference using flexible N:M sparsity via a fully digital compute-in-memory accelerator. While our current implementation does not exploit sparsity, our modular PE design could be extended to incorporate sparsity-aware optimizations.

Li et al. [5] provide a comprehensive hardware perspective on LLM inference acceleration, analyzing trade-offs between different architectural approaches. Our hybrid systolic array and IMC design occupies a unique position in the design space they identify, balancing throughput, latency, and energy efficiency.

Wang et al. [6] present a low-latency PIM accelerator specifically targeting edge LLM inference. Our work shares the edge deployment goal but differs in the systematic partitioning of GeMM and GeMV workloads and the use of reconfigurable dataflows in the systolic array.

Our work differs from these approaches by: (1) intelligently partitioning compute-intensive and memory-intensive operations to specialized hardware units; (2) implementing reconfigurable PE elements supporting multiple dataflow patterns; (3) incorporating user-managed memory with explicit data reuse to eliminate cache overhead; and (4) demonstrating practical integration of digital IMC with systolic arrays at the system level.

V. CHALLENGES AND INSIGHTS

During the development of our architecture, we encountered several technical challenges that required innovative solutions:

To handle RAW dependencies in the systolic array, we had to pass delta clock cycles using `wait(SC_ZERO_TIME)` in SystemC to allow signals to settle properly before subsequent operations could access updated values. Implementing RCS to simultaneously read and write data within the same cycle was crucial for avoiding unnecessary delays in the IMC macro. The decoupled read and write ports of 8T SRAM cells were essential for achieving this capability.

Implementing very large memories posed scalability issues, both in simulation time and in the complexity of memory addressing and management. This motivated our two-level memory hierarchy with an L1 buffer and strategic data reuse mechanisms. The choice between Weight Stationary and Output Stationary dataflows significantly impacts memory bandwidth requirements and energy consumption. Our reconfigurable PE design allows the compiler to optimize dataflow dynamically based on the specific layer characteristics.

An efficient bit-level orchestration of operands in the IMC macro was critical for performing multiplication operations. The decomposition of multi-bit multiplications into sequences of Boolean operations and the organization of the adder tree required careful optimization.

VI. NEAR-TERM FUTURE WORK

The work can be expanded in several directions. To start off with, the complete integration of the Vector Processing Unit (VPU), IMC macro, and systolic array at the layout level would enable accurate power, performance, and area (PPA) analysis. The IMC macro can be optimally sized to make operations faster and more energy-efficient by exploring different array dimensions and precision trade-offs. The latency of various operations can be refined to model more realistic scenarios. This would provide more accurate performance projections for ASIC implementation.

With full integration of the hardware components, focusing on compiler work would enable running real-life workloads and benchmarking the architecture. This includes developing optimization passes for tiling strategies, dataflow selection, and memory allocation. Also, many LLM weight matrices exhibit significant sparsity after quantization and pruning. Extending the PE design to skip zero-valued operations could further improve both performance and energy efficiency. Ofcourse, supporting mixed-precision operations, where different layers or operations use varying bit-widths (INT4, INT8, FP16), could provide better trade-offs between accuracy and efficiency for specific LLM architectures.

VII. CONCLUSION

This work presents a hybrid accelerator architecture for LLM inference that addresses the key challenges of energy consumption and memory bottlenecks in generative AI workloads. By intelligently partitioning compute-intensive GeMM operations to a reconfigurable systolic array and memory-intensive GeMV operations to a digital in-memory computing core, we achieve significant improvements in both performance and energy efficiency.

Our systolic array implementation with configurable dataflows (Weight Stationary and Output Stationary) and optimized tiling strategies demonstrates significant reduction in

memory reads through intelligent data reuse mechanisms. Under realistic memory latency assumptions, the architecture achieves approximately 2× performance improvement compared to designs without data reuse.

The digital IMC macro, built using 8T SRAM cells with decoupled read and write ports, enables read-compute-store operations in a single cycle. Circuit-level characterization in TSMC 180nm technology demonstrates **2.32**× energy efficiency improvement compared to conventional digital implementations, with support for INT4, INT8, and INT16 precision levels.

Our bottom-up design approach using SystemC for system-level modeling and Ngspice for circuit-level characterization provides a practical methodology for rapid prototyping and evaluation of hybrid architectures. The modular and reconfigurable nature of our design enables adaptation to different LLM architectures and deployment constraints.

Future work will focus on complete layout-level integration, compiler optimizations for real workloads, and exploration of advanced techniques such as sparsity exploitation and mixed-precision arithmetic. This work represents a significant step toward energy-efficient, low-latency accelerators for edge deployment of large language models.

The code base for the project can be accessed here.

ACKNOWLEDGMENT

We would like to thank the professors for their timely guidance throughout this project. The continuous evaluations helped us track our progress effectively. The lectures provided us with the fundamental concepts required to initiate and successfully execute this work.

REFERENCES

[1] C. Kachris, "A Survey on Hardware Accelerators for Large Language Models," *Applied Sciences*, vol. 15, no. 2, p. 586, 2025.

[2] S. Moon *et al.*, "A Latency Processing Unit: A Latency-Optimized and Highly Scalable Processor for Large Language Model Inference," HyperAccel / KAIST Technical Report, 2024.

[3] A. Agrawal, A. Jaiswal, C. Lee, and K. Roy, "X-SRAM: Enabling In-Memory Boolean Computations in CMOS Static Random Access Memories," *IEEE Trans. Circuits Syst. I*, vol. 65, no. 12, pp. 4212–4225, 2018.

[4] A. Ramachandran *et al.*, "Accelerating LLM Inference with Flexible N:M Sparsity via a Fully Digital Compute-in-Memory Accelerator," Georgia Tech and Intel Research Report, 2023.

[5] J. Li *et al.*, "Large Language Model Inference Acceleration: A Comprehensive Hardware Perspective," Shanghai Jiao Tong University / Infinigence-AI Technical Report, 2024.

[6] X. Wang *et al.*, "Low-Latency PIM Accelerator for Edge LLM Inference," Preprint, 2024.

[7] Z. Zhu, H. Li, W. Ren, M. Wu, L. Ye, R. Huang, and T. Jia, "Leveraging compute-in-memory for efficient generative model inference in TPUs," in *Proc. Design, Automation & Test in Europe Conf. (DATE)*, 2025, pp. 1–7.

[8] C.-J. Lee and T. T. Yeh, "ReSA: Reconfigurable systolic array for multiple tiny DNN tensors," *ACM Trans. Archit. Code Optim.*, vol. 21, no. 3, pp. 43:1–43:24, Sep. 2024.

[9] S. K. Patel and S. K. Singhal, "Area–delay and energy efficient multi-operand binary tree adder," *IET Circuits, Devices & Systems*, vol. 14, no. 5, pp. 586–593, 2020.

[10] Q. Xie, X. Lin, Y. Wang, S. Chen, M. J. Dousti, and M. Pedram, "Performance comparisons between 7-nm FinFET and conventional bulk CMOS standard cell libraries," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 62, no. 8, pp. 761–765, 2015, doi: 10.1109/TCSII.2015.2391632.

[11] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proc. IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017, doi: 10.1109/JPROC.2017.2761740.