LEAP: LLM Inference on Scalable PIM-NoC Architecture with Balanced Dataflow and Fine-Grained Parallelism

***Abstract*—Large language model (LLM) inference has been a prevalent demand in daily life and industries. The large tensor sizes and computing complexities in LLMs have brought challenges to memory, computing, and databus. This paper proposes a computation/memory/communication co-designed non-von Neumann accelerator by aggregating processing-in-memory (PIM) and computational network-on-chip (NoC), termed LEAP. The matrix multiplications in LLMs are assigned to PIM or NoC based on the data dynamicity to maximize data locality. Model partition and mapping are optimized by heuristic design space exploration. Dedicated fine-grained parallelism and tiling techniques enable high-throughput dataflow across the distributed resources in PIM and NoC. The architecture is evaluated on Llama 1B/8B/13B models and shows $\sim$2.55$\times$ throughput (tokens/sec) improvement and $\sim$71.94$\times$ energy efficiency (tokens/Joule) boost compared to the A100 GPU.**

***Index Terms*—Processing-in-Memory, Large Language Model, Network-on-Chip, Parallelism**

I. Introduction

Due to the massive data volume and computational intensity of large language models (LLMs), current hardware platforms face significant bottlenecks in memory capacity/bandwidth, compute scheduling, and hardware communication energy overhead. Processing-in-memory (PIM) is a widely explored design technique to accelerate AI workloads by bringing compute into the memory [1], [2]. PIM speeds up matrix multiplication ($A \cdot B$) with a dynamic matrix A and a static matrix B (DSMM), which is suitable for the operations with pre-trained weights, *e.g.*, the projection and fully connected layers in LLMs. However, LLMs also contain immense matrix multiplications between runtime-generated dynamic matrices A and B (DDMM) in the attention operations. These DDMMs are less suitable for traditional PIM due to high time and energy costs of dynamically reprogramming memory cells. Moreover, as LLMs scale in model size and input sequence length, the proportion of DDMMs increases substantially.

To address this, existing PIM-based systems often offload DDMMs to separate computing units, including hybrid PIM arrays with configurable precision [3], transposable structures [4], or fully digital accelerators [5]. This results in heterogeneous architectures, where computation mapping and scheduling depend heavily on the stationarity/dynamicity of the data. Such mapping challenges are further intensified in systems scaled via network-on-chip (NoC), which introduces additional design complexity and interconnect overhead. However, most current PIM systems only support algorithm-specific DDMMs and use custom interconnects at limited scales [4], [6], [7], falling short in terms of system scalability and data flow flexibility.

In this work, we present a hardware-software co-design approach to enable scalable and flexible acceleration of LLM inference on heterogeneous PIM architectures. Our end-to-end framework provides partitioning, mapping, and scheduling of LLM inference workloads with awareness of data stationarity and system heterogeneity. In addition, the hardware architecture integrates local compute and memory units within a scalable NoC to support both DDMM-specific dataflows and general aggregation operations. The key contributions of this work are summarized below:

- A fine-grained model partitioning and a heuristically optimized spatial mapping strategy enable high PIM utilization and structured layout.
- Temporal scheduling incorporates dedicated context window tiling and efficient key-value caching (KV cache), ensuring balanced NoC traffic and utilization.
- A custom NoC capable of efficient data communications, DDMMs, and aggregations, with re-programmability via a dedicated instruction set.
- The overall system achieves $\sim$2.55$\times$ throughput improvement and $\sim$71.94$\times$ energy efficiency in the inference of the Llama model compared to the A100 GPUs.

II. Preliminaries

A. Data Stationarity in LLMs

Recent commercial LLMs, such as GPT [8] and the Llama series [9]–[12], are predominantly decoder-only Transformers. Each decoder layer comprises attention and feed-forward sublayers, which involve successive matrix multiplications (MMs) and matrix-vector multiplications (MVMs). Although these models rely on static pre-trained weights, the attention mechanism generates significant dynamic data during inference. To quantify this, consider an attention layer with embedding dimension D and sequence length S. The amount of static data (pre-trained weights) is:

$$DA_{static} = 4D^2 \tag{1}$$

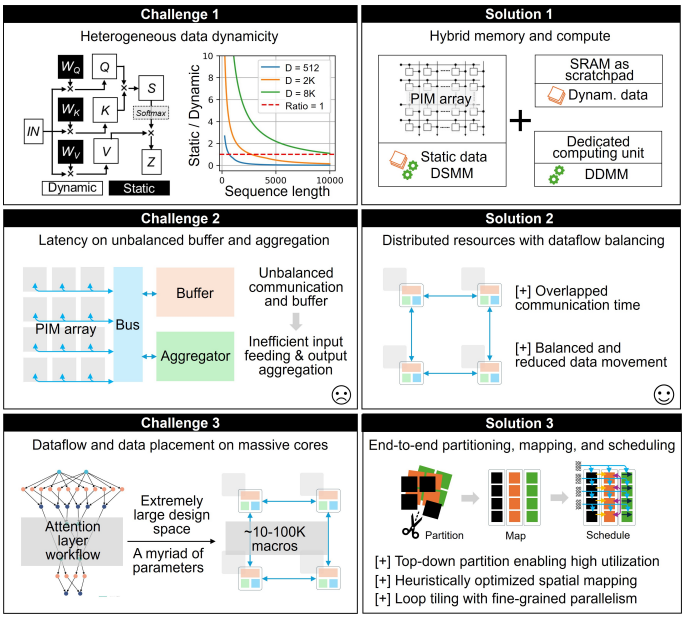


Fig. 1. Design challenges and solutions in accelerating LLM inference.

which is independent of the input sequence length. The dynamic data generated at runtime is:

$$DA_{dynamic} = 5SD + S^2 \tag{2}$$

which correlates to S. As S increases, the ratio of static to dynamic data decreases:

$$\frac{DA_{static}}{DA_{dynamic}} = \frac{4D^2}{5SD + S^2} \overset{S=D}{=} \frac{2}{3} \tag{3}$$

In real-world applications, where $S \gg D$, dynamic data increasingly dominates, particularly under the high-demand sequence length scaling. This insight arrives at the **Challenge 1**: the heterogeneous nature of data in LLMs necessitates differentiated compute and memory strategies for static and dynamic data.

B. PIM Scaling-up

PIM accelerates MMs/MVMs with static weights, *e.g.*, DSMMs, by performing computation within non-volatile memory. However, the typical array size is limited to 32 $\sim$ 256 [13]–[15], making large-scale operations reliant on partitioning across many arrays. This introduces significant overhead in buffering and aggregating partial results, which greatly diminishes overall efficiency [16] if the shared buffer and aggregators are allocated in an unbalanced manner, as shown in Fig. 1. Therefore, **Challenge 2** is that scaling PIM-based MMs/MVMs requires efficient interconnection and aggregation mechanisms to mitigate performance bottlenecks.

C. Target Architecture

To address these challenges, we target a hybrid architecture that combines PIM with a scalable NoC, referred to as aggregated PIM-NoC. The architecture integrates: i) PIM processing elements (PEs) – the non-volatile memory units capable of in-place DSMM computations; and ii) computational routers

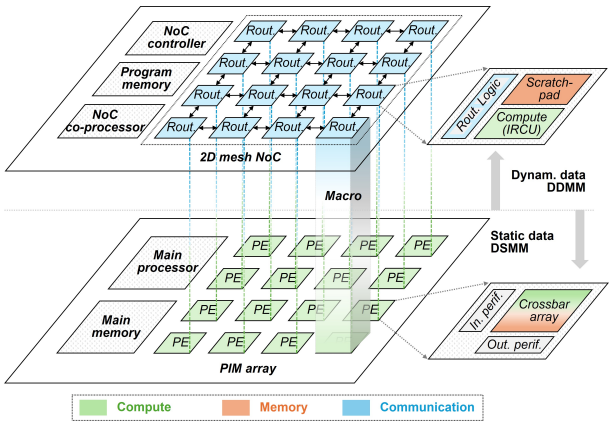


Fig. 2. The proposed aggregated PIM-NoC architecture with distributed fine-grained compute-memory-communication resources.

– the dedicated computing units, termed in-router computing units (IRCUs) and SRAM-based scratchpad, optimized for DDMMs and partial results aggregation, as shown in Fig. 2. Each router-PE pair forms a macro, the basic building block of a distributed 2D mesh system with unified compute, memory, and communication resources. Given these novel features compared to traditional von Neumann architecture, the **Challenge 3** is that efficiently partitioning, mapping, and scheduling LLM workloads on such a spatially distributed and heterogeneous architecture demands compilation framework innovations due to the vast design space. In this work, we demonstrate an end-to-end framework that systematically addresses these challenges.

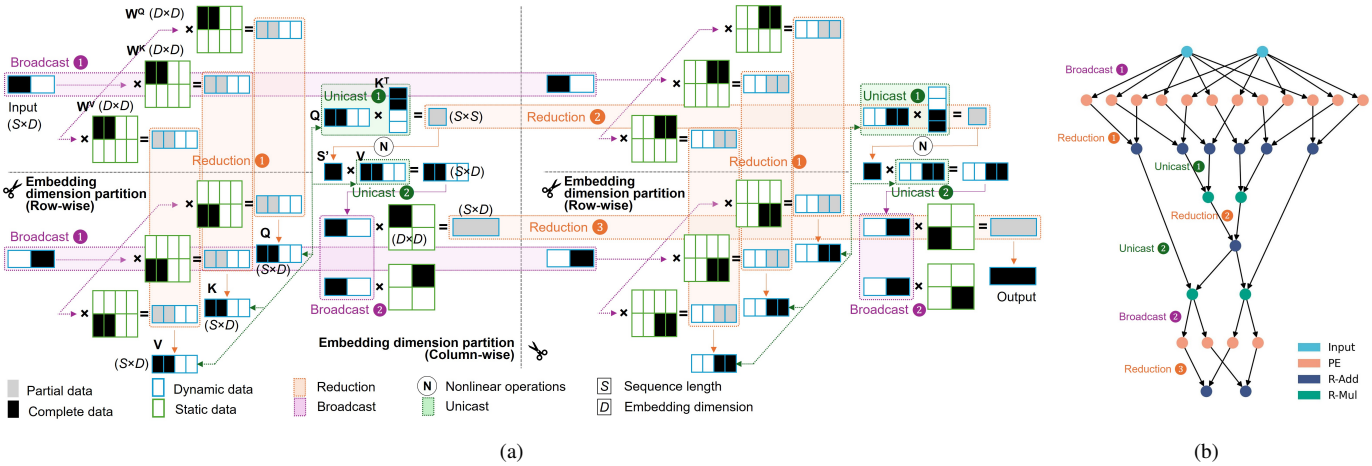
III. Model Partitioning and Spatial Mapping

This section introduces the partitioning scheme for the projection weight matrices and a spatial mapping strategy that deploys the partitioned matrices onto the PIM PEs.

A. Partitioning

Partitioning is applied along both row and column dimensions of the static weight matrices, $\mathbf{W_Q}$, $\mathbf{W_K}$, $\mathbf{W_V}$, and $\mathbf{W_O} \in \mathbb{R}^{D\times D}$, to fit the dimensions of the crossbar arrays, as illustrated in Fig. 3 (a) using an attention layer as an example. The number of crossbar arrays required to store each matrix after partitioning is $\lceil \frac{D}{C} \rceil^2$, where C denotes the width and height of a crossbar array. Intermediate data such as $\mathbf{Q}$, $\mathbf{K}$, $\mathbf{V}$, and $\mathbf{S}$ are also partitioned, introducing additional collective communication steps for broadcasting partitioned inputs (Broadcast ❶/❷) or reducing partial outputs (Reduction ❶/❷/❸). The data dependencies and communication requirements among partitioned matrices and operations are represented by the directed acyclic graph (DAG) $\mathcal{G}$ shown in Fig. 3 (b).

To execute the whole attention layer on the PIM-NoC architecture, all nodes and edges in $\mathcal{G}$ must be mapped onto either PEs or routers. This involves two main steps: (i) spatial mapping, which assigns partitioned static weights and their associated DSMM operations (represented as orange nodes)



(a) (b)

Fig. 3. (a) Partitioning of an attention layer. This illustration considers multi-head attention (MHA), whereas other attention variants like group-query attention (GQA) can degrade to this scheme by matrix duplication accordingly. (b) DAG represents the data and operations in the partitioned attention layer. "R-Add" and "R-Mul" are short for addition and multiplication operations in routers.

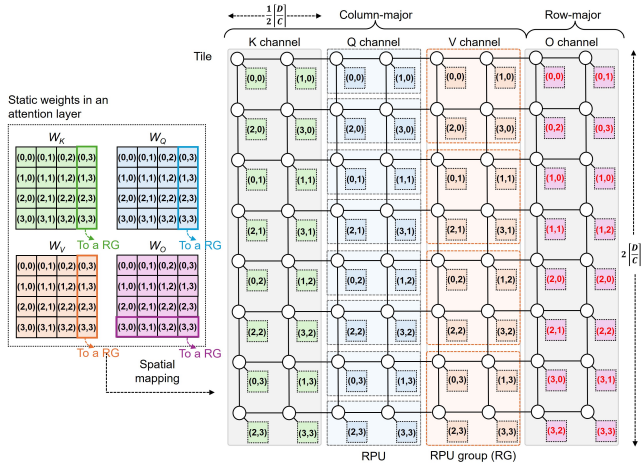


Fig. 4. The spatial mapping used in this work. Static weight matrices are mapped spatially across the crossbar arrays in PEs.

in $\mathcal{G}$ to the crossbar arrays in PEs, and (ii) temporal mapping, which schedules the storage of intermediate data in scratchpads, assigns DDMM operations to IRCUs, and orchestrates the temporal dataflow across the NoC.

B. Spatial Mapping on PEs

A naïve approach to achieving optimal spatial mapping is an exhaustive design space exploration, which is computationally prohibitive due to the vast number of possible mappings. For instance, a static weight matrix of size 1024×1024 can be partitioned into 64 sub-matrices, each fitting a 128×128 crossbar array. The total number of possible mappings is ${}^{64}P_{64} \approx 1.27 \times 10^{89}$, leading to an extremely large and impractical search space. To efficiently obtain a near-optimal mapping, we introduce the following heuristic constraints:

- Sub-matrices originating from the same weight matrix must be placed within a spatially proximate region.
- This region should have a rectangular shape to facilitate regular dataflows and reduce routing complexity.
- The sub-matrices within this region should be ordered in a row-major or column-major fashion.

These constraints dramatically reduce the number of mapping candidates by approximately $10^{86}\times$, resulting in only 1440 valid configurations. We define the cost function for spatial mapping as the total communication time, $\mathbb{C} = T^{\text{tot}}_{\text{comm}}$, and use a naïve X–Y routing algorithm as the baseline for communication cost estimation. With the constrained search space, the spatial mapping exploration completes within 20 seconds. The selected spatial mapping strategy is visualized in Fig. 4, and its optimality will be evaluated in Section VI. The entire attention layer is mapped onto a square region comprising $2\lceil \frac{D}{C} \rceil \times 2\lceil \frac{D}{C} \rceil$ macros, referred to as a tile. Each individual projection weight matrix is allocated to a rectangular region of $2\lceil \frac{D}{C} \rceil \times \frac{1}{2}\lceil \frac{D}{C} \rceil$ macros within this space, referred to as a channel. Sub-matrices from $\mathbf{W_Q}/\mathbf{W_K}/\mathbf{W_V}$ are mapped in a column-major, while those from $\mathbf{W_O}$ are mapped in a row-major. We define the following terminology used throughout the rest of the paper: i) A row-wise processing unit (RPU) refers to a single row of macros within a channel; and ii) An RPU group (RG) consists of the RPUs that store a column-wise partition of $\mathbf{W_Q}/\mathbf{W_K}/\mathbf{W_V}$ or a row-wise partition of $\mathbf{W_O}$ as denoted in Fig. 4.

IV. Dataflow in Temporal Mapping

This section introduces the temporal mapping, which involves storing dynamic data in scratchpads and managing dynamic data movement and multiplication (DDMM) on IRCUs. While spatial mapping addresses partitioning along the embedding dimension, partitioning along the context window (*i.e.*, token sequence length) is achieved through loop tiling in the temporal mapping stage. LLM inference typically consists of two computing phases: i) the prefill stage, which is MM-intensive and processes all input tokens in batch, and ii) the

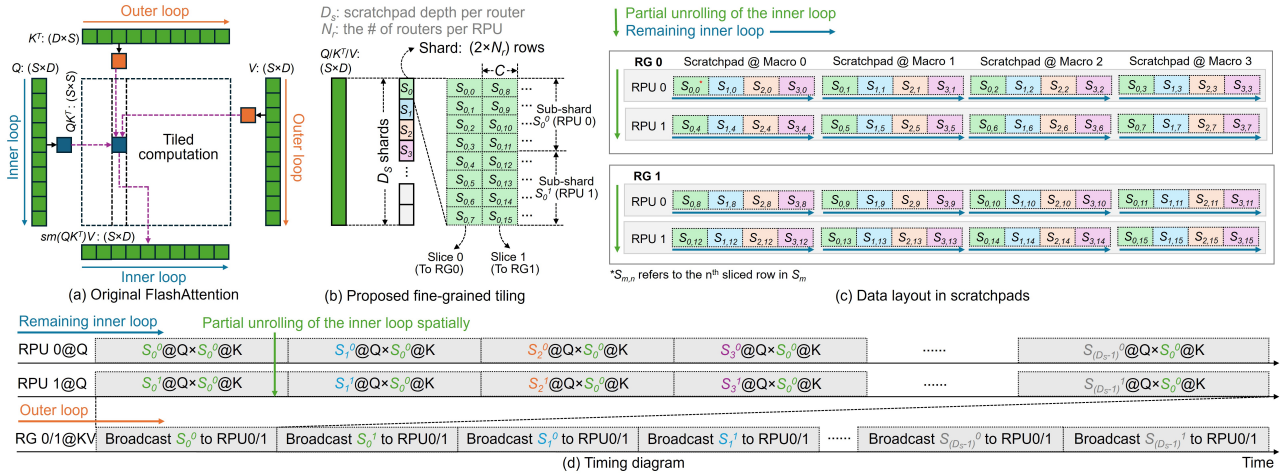


Fig. 5. Context window tiling. (a) Original FlashAttention. (b) $Q/K/V$ tiling. (c) Data layout in scratchpads. (d) Timing diagram.

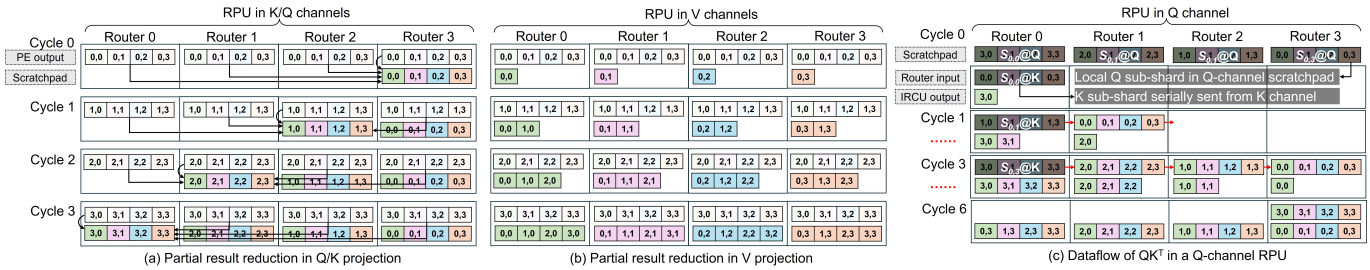


Fig. 6. Dataflow in RPUs when processing a shard by demystifying on a 4×4 matrix. (a) Row-major reduction of Q/K. (b) Column-major reduction of V. (c) Dataflow of $\mathbf{QK}^T$ operation in prefill stage.

decode stage, which is MVM-intensive and generates output tokens one at a time. The dataflow strategies for both the prefill and decode stages are introduced in the following subsections.

A. Context Window Tiling

The attention score $\mathbf{S} \in \mathbb{R}^{S\times S}$ computed as $\mathbf{QK}^T$, has a size quadratic in the context window length, which often exceeds on-chip memory capacity. In GPU-based scenarios, FlashAttention [17], a loop-tiling method, is widely adopted to reduce the off-chip memory access and avoid materializing the full $\mathbf{S}$ matrix at once. FlashAttention tiles $\mathbf{Q/K/V}$ matrices along the sequence length dimension (S), using two nested loops: the outer loop iterates over tiled $\mathbf{K/V}$ matrices, while the inner loop processes the tiled $\mathbf{Q}$ matrices as illustrated in Fig. 5 (a).

We adopt this nested loop structure in our design, but introduce three key distinctions: **(i)** A dedicated fine-grained tiling scheme is applied to $\mathbf{Q/K/V}$. These matrices are partitioned into shards along two dimensions, as shown in Fig. 5 (b). Each row of a shard is distributed across different routers within a RG, as illustrated in Fig. 5(c). The capacity of each shard is $C_S = 2 \cdot N_r = \lceil \frac{D}{C} \rceil$, where $N_r = \frac{1}{2}\lceil \frac{D}{C} \rceil$ is the number of routers in an RPU. With this scheme, the context window length supported by a tile is $D_S \cdot C_S$, where D_S is the scratchpad depth per router. **(ii)** The inner loop is spatially unrolled across the RPUs, exploiting their parallelism to improve throughput. **(iii)** The outer loop is implemented by rotational broadcasting of the $\mathbf{K/V}$ shards across the RPUs within each RG as shown in Fig. 5(d). The detailed dataflow for processing a single shard is described in the following subsections.

B. Prefill Dataflow

1) DSMM: During the projection step, input activations are fed from the leftmost column into Q/K/V channels (Broadcast ❶ as in Fig. 3(b)). Each PE within the Q/K/V channels generates a vector of partial results per cycle. These partial results are then aggregated within each RG (Reduction ❶). It is noted that the aggregation sequence differs across channels: row-major in the K/Q channels and column-major in the V channel, as shown in Fig. 6 (a) and (b), respectively. The aggregated results are stored in the scratchpad according to the layout strategy introduced in the previous subsection.

2) DDMM: Each shard $\mathbf{K}^s$ within the K-channel RPU is read from the scratchpad and transmitted rightward to the corresponding Q-channel RPU within the same row, whose pipelining is shown in Fig. 6 (Unicast ❶). Each Q-channel RPU computes a local partial attention score, which is then aggregated through a vertical reduction across Q-channel RGs to obtain the full attention score shard $\mathbf{S}^s$ (Reduction ❷). A

local Softmax operation is applied as $\mathbf{S}^s$ is passed to the V channel. We adopt the softmax algorithm from FlashAttention, which requires storing intermediate values such as $\mathbf{O}^s$ and `rowmax`, etc. These are held in the O-channel scratchpad. Partial results received from the V channel are combined element-wise with previously accumulated values and written back to the O-channel scratchpad (Unicast ❷). Once completed, each full $\mathbf{O}^s$ shard is broadcast across the corresponding O-channel RG (Broadcast ❷), followed by a vertical reduction to finalize the output (Reduction ❸).

Under the proposed spatial and temporal mapping strategy, dataflow aligns cleanly along horizontal and vertical paths. This regularity enables an expected balance between minimizing traffic collisions and maximizing parallelism.

C. Decode Dataflow

In the decode stage, two key differences distinguish it from the prefill stage: **(i)** only a single newly-generated Q vector is involved in the attention computation, and **(ii)** newly-generated K/V vectors are incrementally appended into the scratchpad at each timestep. Due to this limited parallelism, the $\mathbf{QK}^T$ pipeline shown in Fig. 6 may be underutilized, leading to reduced throughput compared to the prefill stage, as will be demonstrated in Section VI. Nevertheless, the caching of newly generated $\mathbf{K}/\mathbf{V}$ vectors adheres to the same placement strategy shown in Fig. 5(b), which inherently ensures balanced scratchpad utilization across routers. This approach eliminates the need for additional data movement or shifting, offering an improvement over prior KV-cache management techniques such as those in [18], especially on scalable architectures.

V. Hardware Implementation

A dedicated NoC is designed to facilitate effective communication among PEs and to support the DDMMs and aggregations in the IRCU. The NoC architecture comprises three key components: the program memory, the main controller, and the router mesh.

A. Instruction Format

The co-processor programs the NoC program memory (NPM) with instructions. Each instruction comprises two components: a command pair (CMD1, CMD2) and a configuration word, which are written to the command register and configuration register, respectively, as illustrated in Fig. 7. The configuration word encodes the command repetition count (CMD_rep) and the router selection bits (Sel_bits). CMD1 and CMD2 can be executed concurrently, each directing data along a distinct, non-conflicting path. This design aligns with the earlier elaborated dataflow, which shows that concurrent data movement typically occurs in at most two directions.

An alternative instruction-reading scheme combined with a double-bank design is employed to minimize idle time. NPM consists of two independent banks, each containing a set of command and configuration registers. These banks are configured alternatively by the co-processor: while the controller reads instructions from one bank, the co-processor programs

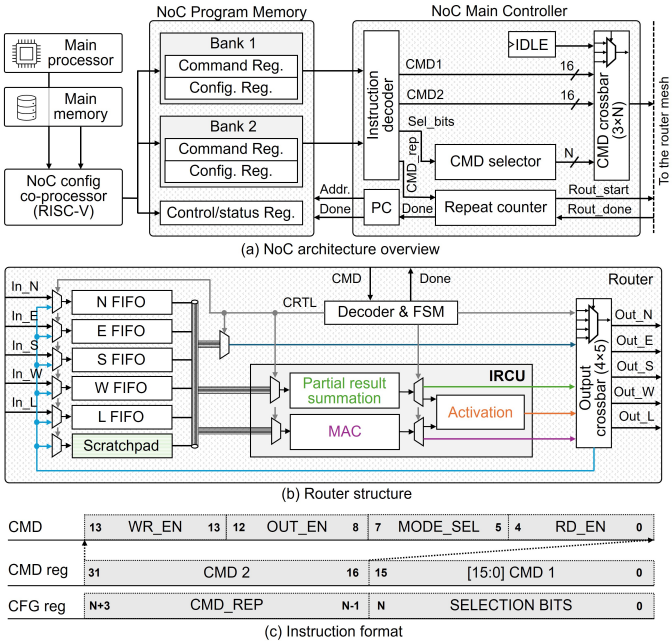


Fig. 7. Overview of the NoC system architecture.

the other. For example, when the controller is reading from Bank 2, Bank 1 is simultaneously configured, and vice versa.

The NoC main controller (NMC) is responsible for fetching and decoding the instructions from the NPM to orchestrate data movement. During decoding, the instruction is split into two commands, CMD1 and CMD2, which are dispatched to the router command crossbar. The command crossbar is a 3-input, N-output structure, where N corresponds to the number of routers in the network. Each router concurrently executes either CMD1, CMD2, or remains IDLE, repeating the operation for the number of times specified by CMD_rep in the configuration word. A command repeat counter keeps track of the remaining repetitions by decrementing on each cycle. Once the counter reaches zero, it signals the program counter (PC) to advance to the next instruction.

A Python API is provided to facilitate programming the LLM inference dataflow to the 2D mesh NoC. The compiler then translates the user's Python code into a corresponding hex file that can be loaded into the NPM for execution.

B. Router Implementation

Each router includes five data I/O ports: four for interconnection with adjacent routers (North, East, South, and West), and one for communication with the locally attached PE. Incoming data from these ports are buffered in dedicated FIFOs. The IRCU supports key operations including: partial result summation (used in Reductions ❶/❷/❸), activation functions (*e.g.*, Softmax), and multiply-accumulate (MAC) computations (used in DDMMs). The output crossbar switch is a 4-input-5-output crossbar, allowing data to be routed to adjacent routers or the local PE. This architecture supports

TABLE I
SYSTEM-LEVEL HARDWARE CONFIGURATION

Component	Specs	Component	Specs
Architecture level (for Llama 3.2-1B)			
Tile #	64	Channel #	4 per tile
RPU #	32 per channel	Macro #	8 per RPU
Macro level			
XB size	128×128	XB cell	8-bit
Scratchpad size	32 KB	Scratchpad width	16-bit
Rout. buf. size	256 B	Rout. buf. width	16-bit
Packet width	64-bit	MAC #	16

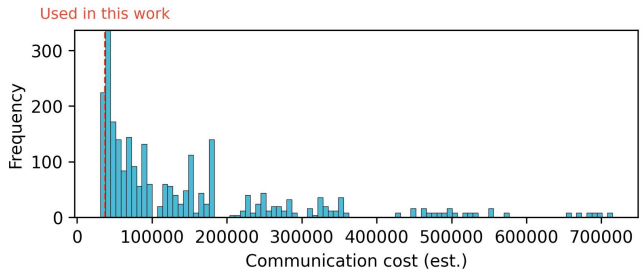


Fig. 8. Distribution of the communication cost in a spatial mapping space exploration of mapping an attention layer in Llama 3.2-1B to 1024 macros.

multi-cast, enabling a single data packet to be forwarded to up to five destinations concurrently.

VI. RESULTS AND DISCUSSIONS

A. Experimental Setup

Hardware testbed: The hardware configurations are summarized in Table I. The digital components (routers and controllers) are implemented in Verilog HDL and synthesized using Synopsys Design Compiler at a 45 nm technology node [19]. Power estimation is performed with Synopsys PrimeTime, using switching activity data from post-synthesis simulations, and place-and-route is carried out using Cadence Innovus. Scratchpad area and power are estimated via CACTI [20]. The area and power of the PIM PE, featuring a 128×128 RRAM crossbar array, are adopted from [15].

Performace benchmark: End-to-end throughput is evaluated on various LLMs: Llama 3.2-1B [12], Llama 3-8B [10], and Llama 2-13B [9], using an instruction-level simulator customized for the proposed NoC instruction set.

B. Mapping Space Exploration

To evaluate the optimality of the proposed spatial mapping strategy, we perform a design space exploration based on the heuristics described in Section III-B. Fig. 8 shows the distribution of the communication cost for mapping an attention layer of Llama 3.2-1B onto 1024 macros, across 2,592 evaluated spatial mapping candidates. The results confirm that the adopted strategy yields one of the lowest communication costs among all evaluated mappings. It is worth noting that this communication cost evaluation is based on a coarse-grained X-Y routing algorithm and does not incorporate the fine-grained temporal mapping strategies discussed earlier. This explains why the selected mapping, while near-optimal, is not the absolute minimum in the distribution.

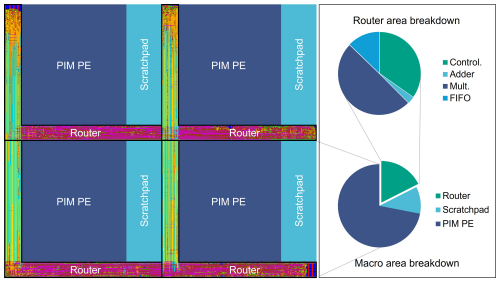


Fig. 9. Example layout of a 2×2 macro array and area breakdown on the macro-level and router level.

TABLE II
MACRO-LEVEL POWER AND AREA BREAKDOWN

	Power (μW)	Breakdown	Area (mm^2)	Breakdown
PIM PE	32.37 [15]	15.08%	0.0864 [15]	73.16%
Scratchpad	37.80	23.53%	0.0125	10.58%
Router*	90.48	56.32%	0.021	17.78%
Total	160.65	100%	0.1181	100%

* The digital results are obtained on 45 nm PDK and then scaled to 7 nm.

C. Power and Area Breakdown

The power and area breakdown of a macro, scaled to the 7 nm technology node, is shown in Table II. Further breakdowns at both the macro and router levels are illustrated in Fig. 9. Although the router accounts for only 17.78% of the macro's area, it contributes to 75.10% of the energy consumption due to its central role in data communication and dynamic processing. Thanks to the scalability of the 2D mesh topology, the area distribution remains consistent even as the system scales.

Table III compares the proposed system with state-of-the-art GPUs, A100 and H100, in terms of throughput, power, and energy efficiency. The throughput is evaluated with a full context window of 2048 tokens: 1024 input tokens and 1024 output tokens. Compared to A100, the proposed system achieves ∼2.55× higher throughput and ∼71.94× higher energy efficiency. While its throughput is lower than that of the H100, it still delivers a ∼24.22× improvement in energy efficiency. The significant improvement in energy efficiency is attributed to the reduced data movement overhead enabled by the fully distributed compute/memory architecture and its highly optimized dataflow, in contrast to the conventional shared-memory design of GPUs.

D. End-to-end Throughput

Fig. 10 illustrates the inference throughput across various models and context window sizes, with further throughput breakdown into the prefill state and the decode stage. The decode throughput is generally 4∼6× less than that of the prefill stage. This degradation is primarily due to two factors: i) The number of past tokens that each newly generated token

TABLE III
COMPARISON TO GPU PLATFORMS

		Ours	A100 [18]	H100
Frequency (GHz)		1	1.4	1.7
Throughput* (tokens/s)	Llama 3-8B	202.25	78.36	274.26
	Llama 2-13B	120.62	47.86	167.51
Power (W)		10.53	~300	~350
Energy efficiency (tokens/J)	Llama 3-8B	19.21	0.2612	0.7836
	Llama 2-13B	11.45	0.1628	0.4786

* Tested context window: 1024 input tokens, and 1024 output tokens.

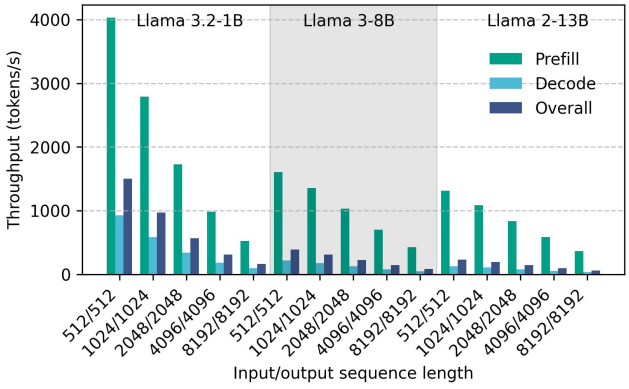


Fig. 10. Throughput under various models and input/output sequence lengths.

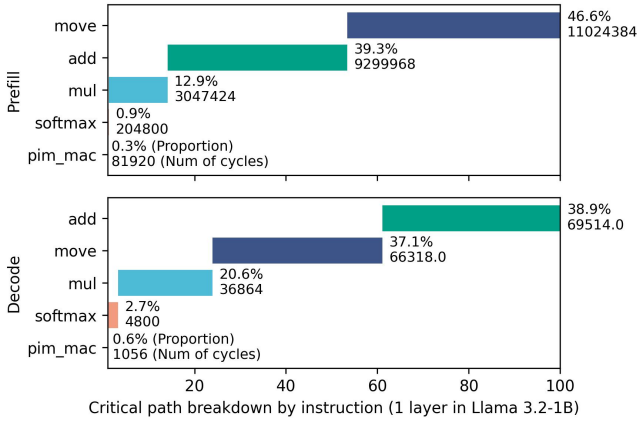


Fig. 11. Breakdown of clock cycles on the critical path by instructions in processing an attention layer and its subsequent MLP in Llama 3.2-1B. `mul` and `add` stands for the computations in IRCU.

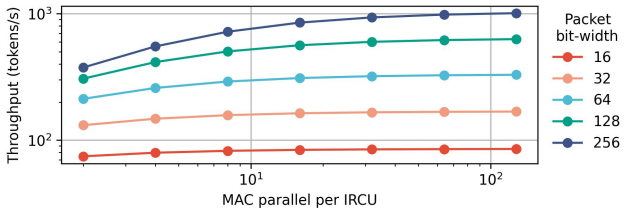


Fig. 12. Trend projection of throughput under increased packet bit-width and IRCU parallelism.

must attend to keeps increasing; and ii) Only a single Q vector is involved in the $\mathbf{QK}^T$ operation in the decode stage, leading to underutilization of the pipelined routers in the Q-channel RPUs.

The throughput drops sublinearly with the increase of model sizes, which stems from both how models scale and the critical path in the proposed architecture. Model size typically scales along three dimensions: the embedding dimension, the MLP hidden dimension, and the number of layers, whose scaling factors are denoted as s_e, s_h, and s_l, respectively. Under such scaling, the attention and MLP layers approximately increase in parameter count by factors of $(s_e^2)\times$ and $(s_e \cdot s_h)\times$, respectively. For example, when comparing Llama 3.2-1B and Llama 3-8B, it is $s_e = 2$, $s_h = 1.75$, and $s_l = 2$, resulting in an overall model size increase of roughly $\sim 8\times$ model. However, thanks to the proposed row-wise and column-wise partitioning in both spatial and temporal mapping, the critical path for operations such as broadcast, reduction, and DDMMs is primarily determined by the longest horizontal or vertical communication route. As a result, the critical path scales approximately with $s_e \cdot s_l$ or $s_h \cdot s_l$, instead of the full $s_e \cdot s_h \cdot s_l$ factor, which explains the sublinear drop in throughput.

Fig. 11 shows the breakdown of clock cycles by instruction along the critical path when processing an attention layer and its corresponding MLPs in Llama 3.2-1B, for both the prefill and decode stages. Thanks to the overlapping of computation and communication, along with the high parallelism of PIM PEs, PIM operations rarely lie on the critical path. Instead, latency is predominantly bottlenecked by data movement and DDMM operations within the IRCUs. To investigate ways to alleviate this bottleneck, we evaluated throughput under different packet bit-widths and IRCU parallelism levels. The resulting trends are presented in Fig. 12, which illustrates the trade-offs between communication bandwidth and compute parallelism. The roofline analysis confirms that the configuration used in this work — 64-bit packet width and 16-way parallelism in the IRCU — achieves near-optimal throughput at the performance frontier, without incurring excessive resource overhead.

VII. RELATED WORK

A. Parallelism in Machine Learning

1) LLM in Distributed GPUs: Various parallelism strategies on datasets, model weights, context windows, etc, have been adopted for scaling LLMs on GPU clusters [11]. These approaches often rely on high-performance collective communication fabrics such as NVLink to manage synchronization overhead. In contrast, the fine-grained parallelism enabled by tensor partitioning and RPU dataflow optimization in this work is tailored to low-level on-chip interconnects, making it more suitable for domain-specific and energy-constrained applications.

2) Neural Network Accelerators: Neural networks (NNs) have been widely accelerated using spatial architectures, including fully digital designs [21], [22] and PIM-based architectures [13], [14], [23], [24]. Much of the analysis in NN accelerators focuses on exploiting parallelism by unrolling the

TABLE IV
COMPARISON TO SOTA PIM-RELATED LLM ACCELERATORS

	Projection	Attention	Interconnect	Dataflow
LEAP (**This work**)	PIM	In-router dataflow accelerator	2D mesh network-on-chip	Fine-grained parallelism
ReTransformer [6]	PIM	PIM	Customized on-chip bus	Matrix decomposition + fine-grained pipelining
TranCIM [4]	PIM	PIM	Customized on-chip bus	Coarse-grained pipelining
CPSAA [7]	Hybrid PIM	Hybrid PIM	Customized on-chip bus	Sparsity-aware
HALO [5]	Hybrid PIM	Systolic array	2.5D network-on-package	Coarse-grained mapping optimization
H3D-Transformer [3]	PIM	PIM + systolic array	2.5D network-on-package	Hybrid-precision + coarse-grained pipelining

nested loops inherent in convolution operations [25], [26]. Compared to attention layers in LLMs, these nested loops tend to be deeper, but the operand matrices are smaller and generally limited to DSMMs rather than the hybrid of DSMMs and DDMMs of LLMs.

B. Emerging Spatial Architecture

Spatial architectures distribute modularized memory and computing resources across a spatial array, enabling flexible parallel execution and enhanced data locality. Emerging examples in ML acceleration include coarse-grained reconfigurable arrays (CGRAs) [27], [28] and Cerebras Wafer-Scale Engine (WSE) [18]. In contrast to these existing designs, this work incorporates heterogeneous memory and compute resources within each PIM-NoC macro, providing heterogeneous optimization space for both DSMMs and DDMMs.

C. PIM-based LLM Accelerator

A range of customized PIM designs have been proposed for LLM acceleration, as summarized in Table IV. ReTransformer [6] enhances attention pipelining by decomposing intermediate matrices to improve dataflow efficiency. TranCIM [4] introduces transposable SRAM arrays and implements a coarse-grained pipeline across Q/K/V stacks using a dedicated streaming interconnect. CPSAA [7] improves the attention pipeline by applying transposition on input activations and supports unstructured dynamic sparsity pruning for DDMMs. HALO [5] leverages a 2.5D integrated architecture that combines PIM-based chiplets for DSMMs and systolic array-based chiplets for DDMMs, optimizing inter-chiplet communication through coarse-grained mapping strategies. H3D-Transformer [3] adopts a hybrid-precision approach: low-precision PIM arrays approximate DSMMs, while high-precision digital units refine results to preserve accuracy. While most existing works focus on algorithm-specific optimizations with limited scalability, this paper proposes a scalable architecture that integrates modular compute/memory/communication resources and flexible dataflow to enhance LLM acceleration.

VIII. CONCLUSION

This paper presents an aggregated NoC-PIM architecture for LLM inference acceleration that fully amortizes the computations in the memory and routers. A dedicated end-to-end framework orchestrates the model partitioning, mapping, and scheduling, ensuring high resource utilization and parallelism. Evaluation results demonstrate substantial improvement in throughput and energy efficiency for the Llama model family compared to the on-shelf GPUs. Furthermore, the architecture is highly scalable, accommodating model growth and expanding context windows, with potential for integration with future wafer-scale and network-on-package technologies.

REFERENCES

[1] A. Sebastian, M. Le Gallo, R. Khaddam-Aljameh, and E. Eleftheriou, "Memory devices and applications for in-memory computing," *Nature nanotechnology*, vol. 15, no. 7, pp. 529–544, 2020.

[2] Z. Sun, S. Kvatinsky, X. Si, A. Mehonic, Y. Cai, and R. Huang, "A full spectrum of computing-in-memory technologies," *Nature Electronics*, vol. 6, no. 11, pp. 823–835, 2023.

[3] Y. Luo and S. Yu, "H3D-Transformer: A heterogeneous 3d (H3D) computing platform for transformer model acceleration on edge devices," *ACM Transactions on Design Automation of Electronic Systems*, vol. 29, no. 3, pp. 1–19, 2024.

[4] F. Tu, Z. Wu, Y. Wang, L. Liang, L. Liu, Y. Ding, L. Liu, S. Wei, Y. Xie, and S. Yin, "Trancim: Full-digital bitline-transpose CIM-based sparse transformer accelerator with pipeline/parallel reconfigurable modes," *IEEE Journal of Solid-State Circuits*, vol. 58, no. 6, pp. 1798–1809, 2022.

[5] A. Jaiswal, K. S. Shahana, S. Ravichandran, K. Adarsh, H. B. Bhat, B. K. Joardar, and S. K. Mandal, "HALO: Communication-aware heterogeneous 2.5 D system for energy-efficient LLM execution at edge," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 2024.

[6] X. Yang, B. Yan, H. Li, and Y. Chen, "ReTransformer: ReRAM-based processing-in-memory architecture for transformer acceleration," in *Proceedings of the 39th International Conference on Computer-Aided Design*, 2020, pp. 1–9.

[7] H. Li, H. Jin, L. Zheng, X. Liao, Y. Huang, C. Liu, J. Xu, Z. Duan, D. Chen, and C. Gui, "CPSAA: Accelerating sparse attention using crossbar-based processing-in-memory architecture," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 43, no. 6, pp. 1741–1754, 2023.

[8] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[9] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale *et al.*, "Llama 2: Open foundation and fine-tuned chat models," *arXiv preprint arXiv:2307.09288*, 2023.

[10] A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Yang, A. Fan *et al.*, "The llama 3 herd of models," *arXiv preprint arXiv:2407.21783*, 2024.

[11] Meta, "Introducing Llama 3.1: Our most capable models to date," 2024, accessed: 2025-03-19. [Online]. Available: https://ai.meta.com/blog/meta-llama-3-1/

[12] ——, "Llama 3.2: Revolutionizing edge AI and vision with open, customizable models," 2024, accessed: 2025-03-19. [Online]. Available: https://ai.meta.com/blog/llama-3-2-connect-2024-vision-edge-mobile-devices/

[13] P.-Y. Chen, X. Peng, and S. Yu, "NeuroSim: A circuit-level macro model for benchmarking neuro-inspired architectures in online learning," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 12, pp. 3067–3080, 2018.

[14] X. Peng, S. Huang, H. Jiang, A. Lu, and S. Yu, "DNN+NeuroSim V2.0: An end-to-end benchmarking framework for compute-in-memory accelerators for on-chip training," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 1–1, 2020.

[15] X. Peng, R. Liu, and S. Yu, "Optimizing weight mapping and data flow for convolutional neural networks on processing-in-memory architectures," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 67, no. 4, pp. 1333–1343, 2019.

[16] Z. Dai, S. Yan, Z. Cong, Z. Guo, Y. He, W. Sun, C. Dou, F. Zhang, J. Yue, Y. Liu *et al.*, "A 41.7 TOPS/W@ INT8 computing-in-memory processor with zig-zag backbone-systolic CIM and block/self-gating CAM for NN/recommendation applications," in *2024 IEEE Symposium on VLSI Technology and Circuits (VLSI Technology and Circuits)*. IEEE, 2024, pp. 1–2.

[17] T. Dao, D. Fu, S. Ermon, A. Rudra, and C. Ré, "FlashAttention: Fast and memory-efficient exact attention with IO-awareness," *Advances in neural information processing systems*, vol. 35, pp. 16 344–16 359, 2022.

[18] C. He, Y. Huang, P. Mu, Z. Miao, J. Xue, L. Ma, F. Yang, and L. Mai, "WaferLLM: A wafer-scale LLM inference system," *arXiv preprint arXiv:2502.04563*, 2025.

[19] J. E. Stine, I. Castellanos, M. Wood, J. Henson, F. Love, W. R. Davis, P. D. Franzon, M. Bucher, S. Basavarajaiah, J. Oh, and R. Jenkal, "FreePDK: An open-source variation-aware design kit," in *2007 IEEE International Conference on Microelectronic Systems Education (MSE'07)*, 2007, pp. 173–174.

[20] N. Muralimanohar, R. Balasubramonian, and N. Jouppi, "Optimizing nuca organizations and wiring alternatives for large caches with CACTI 6.0," in *40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2007)*. IEEE, 2007, pp. 3–14.

[21] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, 2017.

[22] Y.-H. Chen, T.-J. Yang, J. Emer, and V. Sze, "Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 2, pp. 292–308, 2019.

[23] Y. Wang, Z. Zou, and L. Zheng, "Design framework for SRAM-based computing-in-memory edge CNN accelerators," in *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2021, pp. 1–5.

[24] Y. Wang and X. Fong, "Benchmarking dnn mapping methods for the in-memory computing accelerators," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 2023.

[25] H. Kwon, P. Chatarasi, M. Pellauer, A. Parashar, V. Sarkar, and T. Krishna, "Understanding reuse, performance, and hardware cost of DNN dataflow: A data-centric approach," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '52. New York, NY, USA: Association for Computing Machinery, 2019, p. 754–768. [Online]. Available: https://doi.org/10.1145/3352460.3358252

[26] X. Yang, M. Gao, Q. Liu, J. Setter, J. Pu, A. Nayak, S. Bell, K. Cao, H. Ha, P. Raina *et al.*, "Interstellar: Using halide's scheduling language to analyze DNN accelerators," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2020, pp. 369–383.

[27] Y. Luo, C. Tan, N. B. Agostini, A. Li, A. Tumeo, N. Dave, and T. Geng, "ML-CGRA: An integrated compilation framework to enable efficient machine learning acceleration on CGRAs," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2023, pp. 1–6.

[28] T. K. Bandara, D. Wu, R. Juneja, D. Wijerathne, T. Mitra, and L.-S. Peh, "Flex: Introducing flexible execution on CGRA with spatio-temporal vector dataflow," in *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. IEEE, 2023, pp. 1–9.