FlashGEMM: Mesh-Aware Efficient GEMM for 3D-Stacked LLM Accelerators

Xin Fan[1], Chen Bai[1,2,†], Xin Yang[2], Zhenhua Zhu[1,3], Yanhong Wang, Zhaode Zhang, Yuan Xie[1]
[1] The Hong Kong University of Science and Technology [2] Fudan University [3] Tsinghua University

***Abstract*—Large language models (LLMs) are foundational to artificial general intelligence (AGI), while imposing stringent hardware demands in computational power and memory bandwidth. To meet these demands, recent advances in hybrid bonding offer new opportunities through high-bandwidth, low-latency logic-memory 3D integration. Due to the spatial distribution of bandwidth in 3D-stacked DRAM, accesses to remote DRAM banks need to traverse the network-on-chip (NoC). Since GEMM is the dominant operator in LLM workloads, we analyze existing distributed GEMM algorithms and show that they suffer from communication inefficiency on mesh topologies. To address these challenges, we present FlashGEMM, a mesh-aware distributed GEMM algorithm that achieves efficient communication-computation overlap and optimized execution flow for 3D-stacked LLM accelerators. Evaluations show that FlashGEMM delivers up to 1.50× improvement in time-to-first-token (TTFT) and 7.11× improvement in throughput over prior methods.**

I. Introduction

Large Language Models (LLMs) underpin diverse applications, from conversational assistants to enterprise analytics [1]–[3]. Due to their versatility, the efficiency of LLM inference has become the foremost concern for users in real-world deployments. The inference consists of prefill and decode phases. The prefill phase is compute intensive and dominated by general matrix multiplications (GEMM). The decode phase is memory-bound due to general matrix-vector multiplications (GEMV) and its batched version, flat GEMM [1]. These phases place extreme demands on computational power and memory bandwidth, posing significant challenges for LLM accelerator design.

Serveral accelerators have been proposed to meet the compute and memory bandwidth requirements [5]–[9]. Despite architectural differences, two common design principles are summarized. First, specialized compute units such as tensor cores, 3D cubes, and systolic arrays are designed to maximize computational power for GEMM operations [5]–[8]. Second, high-bandwdith memory (HBM) with advanced 2.5D integration is used to improve the memory bandwidth [9]–[11]. This approach benefits memory-bound operations.

Now, hybrid bonding, an advance in 3D integration, opens new opportunities for LLM accelerator design [12]–[15]. It enables high-density vertical interconnects with up to 110,000 connections per mm^2 at a 3 μm bonding pitch between adjacent dies, while maintaining acceptable thermal power constraints [14], [16]. Such advancement makes 3D-stacked LLM accelerators a feasible future direction. Namely, multiple DRAM dies can be stacked and directly integrated with a logic die through hybrid bonding, as shown in Fig. 4. The adoption of 3D-stacked DRAM offers two key advantages for LLM workloads. First, it reduces memory access latency. The physical distance between DRAM and logic dies is significantly shorter than in 2.5D integration [14], [17], [18]. Hybrid bonding eliminates the need for long wires in redistribution layers, which are required in 2.5D integration. So, direct die-to-die signal transmission is permitted. As a result, compute-intensive GEMM operations enjoy faster memory access. Second, 3D-stacked DRAM provides higher memory bandwidth compared to HBM3E [19]. The bonding pitch can be ten times finer than that of microbumps, enabling a much larger number of I/O connections between memory and logic dies [14], [17], [18]. Hence, several times higher bandwidth than contemporary HBM can be further achieved, allowing higher execution efficiency of flat GEMM operations.

Nevertheless, a defining characteristic of 3D-stacked LLM accelerators introduces a challenge for GEMM operations. Unlike HBM, which delivers memory bandwidth from the periphery of the chip, 3D-stacked DRAM provides vertical bandwidth above the logic die. The vertical integration connects each processing element (PE) directly to the DRAM banks stacked immediately above it, establishing a tightly coupled "local" data path [2]. In contrast, when a PE needs to access a non-local DRAM bank, the data must traverse the NoC. Although 3D-stacked DRAM supplies several times higher aggregate bandwidth than HBM, this bandwidth is "spatially" distributed across these local data paths. GEMM operations often involve intensive data sharing and coordination among multiple PEs, leading to frequent remote DRAM banks accesses. The increased traffic causes NoC congestion and raises data delivery latency. This sustained pressure transforms the NoC into a critical performance bottleneck in 3D-stacked LLM accelerators.

The communication issue and spatially distributed nature of 3D-stacked DRAM bandwidth necessitate an efficient distributed GEMM algorithm. And the algorithm can fully leverage the architectural features. Currently, Cannon-based and SUMMA-based methods are two related prior arts [3]. Cannon-based methods require specific data layouts to perform GEMM operations, but this incurs substantial initialization overhead due to data layout transformation [20], [21]. On the contrary, SUMMA-based methods eliminate the need for data layout transformation [22]. However, they incur heavy NoC traffic due to inefficient broadcast communications [23]. An alternative, SUMMA with all-gather, avoids both the long initialization of Cannon-based methods and the broadcast overhead of traditional SUMMA. Yet it suffers from an inefficient output-stationary GEMM execution flow [20], [21]. Output-stationary GEMM execution flow requires repeated movement of weight parameters across PEs, which is time-consuming during the LLM decode phase.

As existing distributed GEMM implementations are not well optimized for 3D-stacked LLM accelerators, in this paper, we propose FlashGEMM to solve the problem. FlashGEMM embodies our algorithm design philosophy. First, communication-computation overlap can effectively hide NoC communication latency. Specifically, we integrate an overlap mechanism that goes beyond prior approaches, which mainly focus on inter-chip designs [21]. Second, tailoring communication patterns to the NoC topology is essential. Our algo-

† Corresponding author.

[1]Flat GEMM denotes $\boldsymbol{C}_{m\times n} = \boldsymbol{A}_{m\times k}\boldsymbol{B}_{k\times n}$ where $m \ll k, n$. For example, $m = 8$, and $k = n = 2048$ [4].

[2]A PE contains compute units and associated control logic. They are interconnected with a mesh topology for low engineering cost.

[3]SUMMA is an acronym for Scalable Universal Matrix Multiplication Algorithm.

978-3-9826741-1-7/DATE26/© 2026 EDAA

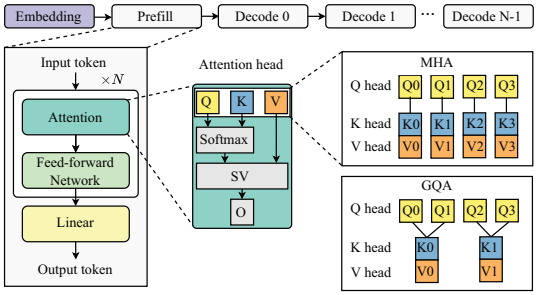


Fig. 1 Architectural overview of a Transformer-based LLM and attention variants: MHA and GQA.

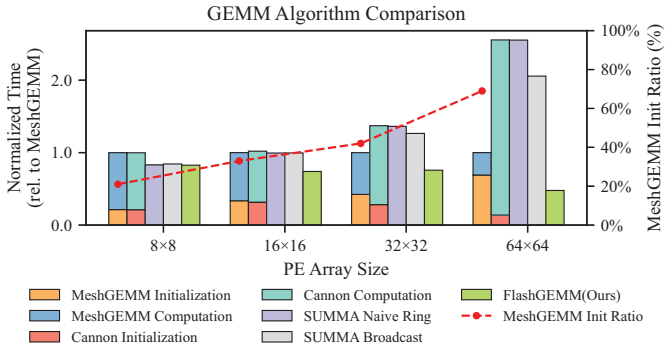


Fig. 2 Performance comparison of GEMM algorithms on 2048 × 2048 × 2048 matrices in FP8 precision. Results are normalized to MeshGEMM. Each PE delivers 4096 GFLOPs of FP8 computational power and 32 GB/s of unidirectional cross-PE bandwidth. The line represents MeshGEMM initialization time fraction.

rithm shortens communication paths on the mesh and dynamically selects data movement strategies based on the workload. Overall, FlashGEMM delivers high performance through three key innovations, which also constitute our primary contributions:

- We introduce a two-level tiling strategy that achieves communication-computation overlap by fully exploiting the memory hierarchy of 3D-stacked LLM accelerators.
- We propose a skipped all-gather algorithm that aligns with the mesh topology and eliminates long communication paths.
- We propose a weight-stationary GEMM execution flow and dynamically select output-stationary or weight-stationary dataflows to minimize communication overhead.
- Evaluations show that FlashGEMM achieves 1.23∼1.50× lower TTFT and 5.44∼7.11× higher throughput compared to state-of-the-·art baselines.

The remainder of this paper is organized as follows. Section II provides background and motivation. Section III details FlashGEMM. Section IV is for experiments, and Section V concludes this paper.

II. Background and Motivation

In this section, Section II-A outlines LLM inference. Section II-B introduces 3D integration and its architectural impact, and Section II-C presents current distributed GEMM algorithms and their challenges.

A. LLM inference

LLM inference is an autoregressive process that generates output tokens conditioned on previously generated tokens. As illustrated in Fig. 1, LLM first runs a prefill pass over the input, followed by iterative decode steps that generate tokens sequentially (Decode 0 to Decode $N-1$). LLMs are built on transformer decoder layers, each consisting of a multi-head attention (MHA) and a feed-forward network (FFN). MHA is defined in Equation (1).

$$\mathrm{MHA}(\boldsymbol{Q},\boldsymbol{K},\boldsymbol{V}) = \mathrm{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^{\top}}{\sqrt{d}}\right)\boldsymbol{V}, \tag{1}$$

where $\boldsymbol{Q}$, $\boldsymbol{K}$, and $\boldsymbol{V}$ are the query, key, and value matrices. d is the dimension of the query or key. During autoregressive decode phase, the model maintains a growing KV cache to avoid recomputation.

While MHA uses independent query, key, and value heads for each MHA, variants like grouped-query attention (GQA) shares KV heads across multiple query heads to reduce compute and memory costs [24]. FFN employs a nonlinear activation, such as SwiGLU [25], exemplified by Equation (2):

$$\mathrm{FFN}(\boldsymbol{x}) = \boldsymbol{W}_2\left(\mathrm{Swish}(\boldsymbol{W}_1\boldsymbol{x}) \otimes (\boldsymbol{W}_3\boldsymbol{x})\right), \tag{2}$$

where $\boldsymbol{W}_1$, $\boldsymbol{W}_2$, and $\boldsymbol{W}_3$ are weight matrices, and $\otimes$ denotes the element-wise product.

B. 3D Integration and Architectural Impact

3D integration enables the vertical stacking of multiple dies within a single package using microbumps or hybrid bonding [13], [14], [26]. Microbumps have diameters ranging from tens of micrometers. Hybrid bonding pads can achieve a pitch of several micrometers [12]. 3D integration supports two schemes: memory-on-logic (MoL) and logic-on-memory (LoM). We adopt the MoL design (Fig. 4), as LoM requires through-silicon vias (TSVs) in the memory die to deliver power, increasing die area and reducing memory density [27].

Stacking DRAM dies directly on the logic die with hybrid bonding brings architectural impact to the 3D-stacked LLM accelerators. Each PE can directly access only the DRAM banks immediately stacked above it, with remote banks reachable via the NoC, resulting in a distributed DRAM architecture. This architectural shift distinguishes our system from traditional LLM accelerators. The DRAM bandwidth improves but spatially distributed across PEs. Consequently, GEMM algorithm design must account for the overhead of cross-PE data transfers over the NoC.

C. Challenge for Distributed GEMM Algorithm

Current distributed GEMM algorithms are typically grouped into two main categories, Cannon-based methods and SUMMA-based methods, along with their variants. We analyze these algorithms to highlight their limitations when applied to mesh topologies.

1) Cannon-based algorithms

Cannon-based algorithms and their mesh-optimized variant MeshGEMM consist of two sequential phases, an initialization phase followed by a computation phase [23]. The initialization phase rearranges input matrices into a required data layout, while the computation phase parallelizes matrix multiplication with communication. However, both phases cannot overlap and consume a significant amount of NoC resources. This leads to NoC congestion caused by conflicts among concurrent communication paths. As multiple paths compete for limited NoC bandwidth, contention delays accumulate. We illustrate the initialization overhead of MeshGEMM in Fig. 2. For the example GEMM workload, the initialization overhead increases as the PE array scales from 8 × 8 to 64 × 64, eventually exceeding the duration of the computation phase and accounting for up to 69.09% of the total GEMM execution time.

2) SUMMA-based algorithms

SUMMA eliminates the initialization cost in Cannon-based algorithms. Consider a GEMM operation with input matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ and result matrix $\boldsymbol{C}$, where $\boldsymbol{C} = \boldsymbol{A}\boldsymbol{B}$. As shown in Fig. 3, $\boldsymbol{A}$ and

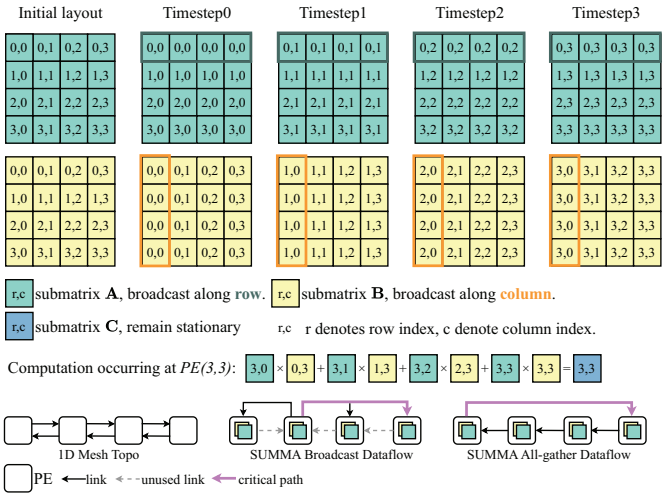


Fig. 3 Illustration of SUMMA with broadcast execution on a 4×4 PE array for $\boldsymbol{C} = \boldsymbol{A} \times \boldsymbol{B}$. For initial layout, matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ are partitioned into 16 sub-matrices, with sub-matrix indices matching the coordinates of the PE array (e.g., PE(0,0) initially holds sub-matrices $\boldsymbol{A}(0,0)$ and $\boldsymbol{B}(0,0)$). From Timestep 0 to Timestep 3, broadcast is needed in the matrix multiplication.

$\boldsymbol{B}$ are partitioned according to the PE array layout, producing sub-matrices distributed across PEs. Each PE stores one sub-matrix of $\boldsymbol{A}$ and one of $\boldsymbol{B}$, and the PE's coordinate in the mesh uniquely identifies its data. In each timestep, SUMMA computes partial results by broadcasting sub-matrices of $\boldsymbol{A}$ along their respective rows and sub-matrices of $\boldsymbol{B}$ along their respective columns. This broadcast multicasts data to all PEs and leaves many NoC links underutilized, as indicated by the gray dashed arrows in Fig. 3. SUMMA with all-gather, an optimized variant, replaces the broadcast with a ring all-gather communication to better utilize these idle links.

Despite this improvement, SUMMA with all-gather faces critical challenges on mesh topologies. First, directly mapping the ring all-gather onto a mesh introduces long NoC routing paths, such as the wrap-around path highlighted by the red arrow in Fig. 3. Even with high link utilization, the wrap-around path in all-gather introduces communication latency and limits scalability. Against SUMMA with broadcast, SUMMA with all-gather yields only a marginal 1.54% improvement on the 8×8 PE array and leads to a 19.47% performance degradation on the 64×64 PE array (Fig. 2). Second, SUMMA with all-gather for $\boldsymbol{C} = \boldsymbol{AB}$ adopts an output-stationary execution flow, whereas weight-stationary dataflow is preferred during the LLM decode phase due to its minimum communication volume [28]. These limitations motivate our design of FlashGEMM.

III. FlashGEMM

In this section, we detail FlashGEMM. Specifically, Section III-A provides an architecture overview. Then, three optimizations are proposed, including a two-level tiling strategy (Section III-B), an optimized all-gather algorithm (Section III-C), and a weight-stationary GEMM design (Section III-D) tailored for the LLM decode phase.

A. Architecture Overview

Fig. 4 illustrates the 3D-stacked LLM accelerator overview. PEs on the logic die form a mesh topology, with multiple layers of DRAM stacked vertically. Each PE is directly connected to a subset of the DRAM banks through hybrid bonding, forming what we refer to as the local data path (Section I). Within a PE, memory controllers

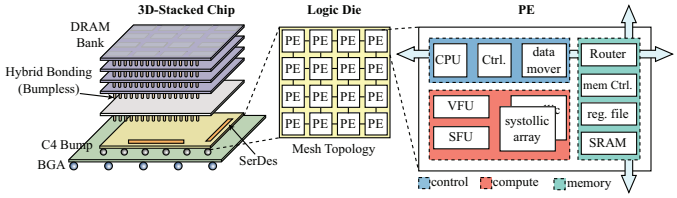


Fig. 4 Architecture overview of the 3D-stacked LLM accelerator.

manage DRAM accesses over this local path, while remote DRAM accesses are routed through the NoC (Section II-B). Besides, each PE integrates an SRAM, a systolic array, a vector unit, a special function unit (SFU), and a data movement accelerator (DMA). The SRAM is software-controlled by CPU. The systolic array handles matrix multiplications. The vector unit supports vector operations. The SFU handles complex mathematical operations (e.g., exponentiation). And the DMA, similar to NVIDIA's tensor memory accelerator (TMA), generates remote memory accesse addresses and offloads asynchronous data transfers over the NoC.

B. Two-level Tiling Strategy

To exploit the distributed bandwidth and low latency of 3D-stacked DRAM, we propose a two-level tiling strategy that enables efficient computation-communication overlap, as detailed in Section III-B1 and Section III-B2. Then, we extend the analysis to PE arrays with non-square sizes in Section III-B3. We consider the example of computing $\boldsymbol{C} = \boldsymbol{AB}$ on a 2×2 PE array to illustrate the two-level tiling strategy, as visualized in Fig. 5. Similar to SUMMA (Section II-C2), $\boldsymbol{A}$ and $\boldsymbol{B}$ are partitioned across the PE array, with each PE storing one sub-matrix of $\boldsymbol{A}$ and one of $\boldsymbol{B}$, which we denote as $\boldsymbol{A}_{\text{sm}}$ and $\boldsymbol{B}_{\text{sm}}$.

1) First-level tiling

The first-level tiling serves to overlaps all-gather communication with computation to hide NoC latency. The first-level tiling proceeds in three steps. *First*, $\boldsymbol{A}_{\text{sm}}$ and $\boldsymbol{B}_{\text{sm}}$ are tiled into blocks along the reduction dimension, which is the dimension to be summed over in the GEMM operation. Fig. 5 illustrates the case with a tiling factor of 2. The dark and light blocks are computed in the first (*iter1*) and second (*iter2*) iterations, respectively. Each PE load one block of $\boldsymbol{A}_{\text{sm}}$ and $\boldsymbol{B}_{\text{sm}}$ from 3D-stacked DRAM into the SRAM per iteration. *Second*, the block of $\boldsymbol{A}_{\text{sm}}$ is all-gathered row-wise, and that of $\boldsymbol{B}_{\text{sm}}$ is all-gathered column-wise. Each PE then holds the corresponding row and column results in local SRAM, denoted as $\boldsymbol{A}_{\text{tile}}$ and $\boldsymbol{B}_{\text{tile}}$. *Third*, GEMM computation is performed on $\boldsymbol{A}_{\text{tile}}$ and $\boldsymbol{B}_{\text{tile}}$. Through this design, 3D-stacked DRAM accesses, NoC communication, and computation are effectively overlapped.

The first-level tiling strategy offers two key advantages. First, since SRAM capacity is limited, the all-gather operation that collects row and column data blocks can exceed available storage. We address this issue by applying tiling to limit the per-iteration data footprint. Second, the distributed bandwidth in 3D-stacked DRAM necessitates explicit communication (i.e., all-gather) to exploit data reuse. Tiling enables overlap of computation and communication across iterations, thereby reducing exposed communication time.

2) Second-level tiling

The second-level tiling further decomposes $\boldsymbol{A}_{\text{tile}}$ and $\boldsymbol{B}_{\text{tile}}$ into finer tiles $\boldsymbol{A}'_{\text{tile}}$ and $\boldsymbol{B}'_{\text{tile}}$. It consists of two steps. *First*, $\boldsymbol{A}'_{\text{tile}}$ and $\boldsymbol{B}'_{\text{tile}}$ are loaded from SRAM into register files. When previous $\boldsymbol{C}'_{\text{tile}}$ cannot be reused, $\boldsymbol{C}'_{\text{tile}}$ is fetched from DRAM into the register file. *Next*, the systolic array computes the product $\boldsymbol{A}'_{\text{tile}}\boldsymbol{B}'_{\text{tile}}$ and accumulates the result into $\boldsymbol{C}'_{\text{tile}}$. $\boldsymbol{C}'_{\text{tile}}$ is written back to DRAM if no further reuse is required. These two steps are executed in parallel.

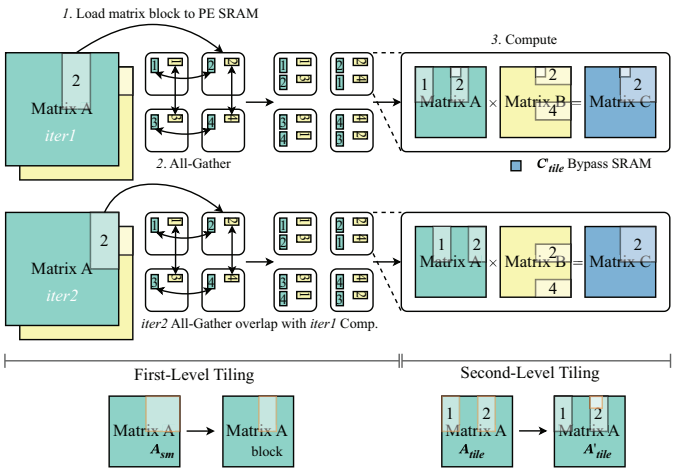


Fig. 5 Overview of the two-level tiling strategy on a 2 × 2 PE array.

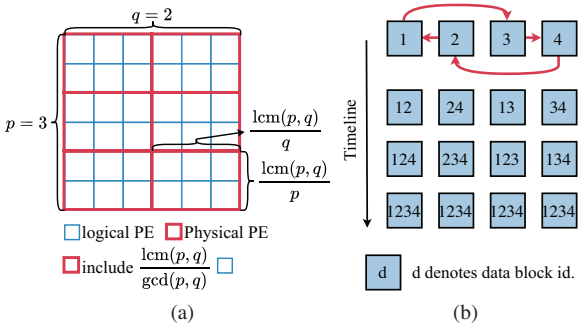


Fig. 6 (a) Overview of physical PE and logical PE for PE array with non-square size. (b) Skipped ring all-gather.

The difference of our second-level tiling, compared to that in architectures where DRAM resides at the chip periphery (e.g., Dojo [11], Cerebras [10]), lies in enabling the register file to access stationary variables (e.g., $\boldsymbol{C}_{\text{tile}}$) directly from local DRAM, bypassing SRAM. Leveraging the low latency and high bandwdith of 3D-stacked DRAM, this design eliminates intermediate SRAM accesses and reduces overall SRAM capacity requirements.

3) PE Array with Non-Square Size

We previously considered tiling in PE arrays with a square size and now extend the discussion to non-square sizes (Fig. 6(a)). For a physical PE array of size $p \times q$, we map it to an equivalent logical square PE array of size $l \times l$, where $l = \text{lcm}(p, q)$ denotes the least common multiple of p and q. Each physical PE includes $\frac{\text{lcm}(p,q)}{\text{gcd}(p,q)}$ logical PEs, where $\text{gcd}(p, q)$ represents the greatest common divisor of p and q. This mapping allows us to apply the tiling developed for square PE arrays to PE arrays of arbitrary sizes.

C. Optimized All-Gather Algorithm

The two-level tiling requires inter-PE all-gather communication. To avoid the long NoC routing paths like wrap-around communication paths introduced in SUMMA with all-gather (Section II-C), we present optimized all-gather algorithms in this section. Specifically, we propose two all-gather methods, as listed in Fig. 6(b), which remove the critical communication path shown in Fig. 3.

Consider four PEs extracted from a row of a mesh topology, numbered left to right as one to four in Fig. 6(b). The skipped ring all-gather constructs a cyclic communication pattern with a maximum hop count of two. Particularly, odd-numbered PEs (e.g., PE_{2n-1}) send data to the next odd-numbered PE (PE_{2n+1}), while even-numbered PEs (e.g., PE_{2n+2}) send data to the preceding even-numbered PE (PE_{2n}). PEs at the periphery receive data from their neighbors. By removing the wrap-around path, which incurs communication latency linear in the number of PEs, this method reduces the communication complexity from $O(N)$ to nearly $O(1)$. Compared with the baseline all-gather, our skipped-ring all-gather improves the performance of GEMM by 5.34× on the 64 × 64 PE array (Fig. 2). Moreover, SUMMA with our skipped-ring all-gather consistently outperforms SUMMA with broadcast across all PE array sizes.

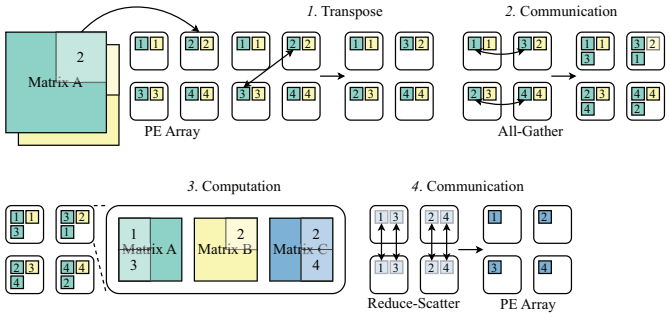


Fig. 7 Weight-stationary dataflow in decode on a 2 × 2 PE array.

D. Weight-Stationary GEMM Design

In Fig. 5, $\boldsymbol{C} = \boldsymbol{AB}$ adopts an output-stationary dataflow. However, output-stationary is not always optimal for LLM workloads. Consider $\boldsymbol{xW_q}$, the input token $\boldsymbol{x}$ has the shape $b \times s \times h$, where b is the batch size, s is the sequence length, and h is the hidden dimension. The weight $\boldsymbol{W_q}$ has the shape $h \times h$. In prefill s is typically large (e.g., $s = 1\text{K}$), whereas in decode $s = 1$, with input and output tensors of shape $b \times 1 \times h$. The communication volume of output-stationary relative to weight-stationary is on the order of $O\left(\frac{h}{bs}\right)$. Still applying output-stationary in decode incurs substantial weight communication, which significantly reduces LLM throughput.

Fortunately, performing $\boldsymbol{C} = \boldsymbol{A}^\top \boldsymbol{B}$ with SUMMA follows a weight-stationary dataflow. To obtain the weight-stationary formulation, we introduce an additional transpose of matrix $\boldsymbol{A}$ and rewrite the multiplication as $\boldsymbol{C} = (\boldsymbol{A}^\top)^\top \boldsymbol{B}$. We then analyze the communication time for $\boldsymbol{C} = \boldsymbol{AB}$ and $\boldsymbol{C} = (\boldsymbol{A}^\top)^\top \boldsymbol{B}$. Assume that $\boldsymbol{A}$ has dimension $m \times k$ and $\boldsymbol{B}$ has dimension $k \times n$, mapped onto a $p \times p$ PE array. We estimate communication time using the α-β model, where α denotes link latency and β is the inverse bandwidth [29], [30]. We ignore the computation time of additions in the reduce-scatter. The communication latency of $\boldsymbol{C} = \boldsymbol{AB}$ stems from the skipped ring all-gather of $\boldsymbol{A}$ and $\boldsymbol{B}$, and can be expressed as

$$L_{\text{os}} = \left(4\alpha + \frac{(mk + kn)}{p^2}\beta\right)(p - 1). \tag{3}$$

The communication latency of $\boldsymbol{C} = (\boldsymbol{A}^\top)^\top \boldsymbol{B}$ is given by the transpose of $\boldsymbol{A}$ and all-gather of $\boldsymbol{A}^\top$, together with the reduce-scatter of $\boldsymbol{C}$, and can be expressed as

$$L_{\text{ws}} = \left((4 + p)\alpha + \frac{2mk + mn}{p^2}\beta\right)(p - 1). \tag{4}$$

Thus, whenever $L_{\text{ws}} < L_{\text{os}}$, it is advantageous to employ the formulation $\boldsymbol{C} = (\boldsymbol{A}^\top)^\top \boldsymbol{B}$ under a weight-stationary dataflow.

Fig. 7 shows weight-stationary dataflow. First, matrix $\boldsymbol{A}$ is transposed to $\boldsymbol{A}^\top$. Next, row-wise all-gather is performed on $\boldsymbol{A}^\top$, while the matrix $\boldsymbol{B}$ remain stationary. Each PE then computes to produce partial results. Subsequently, these partial results are accumulated through column-wise reduce-scatter, which also leverages the skipped

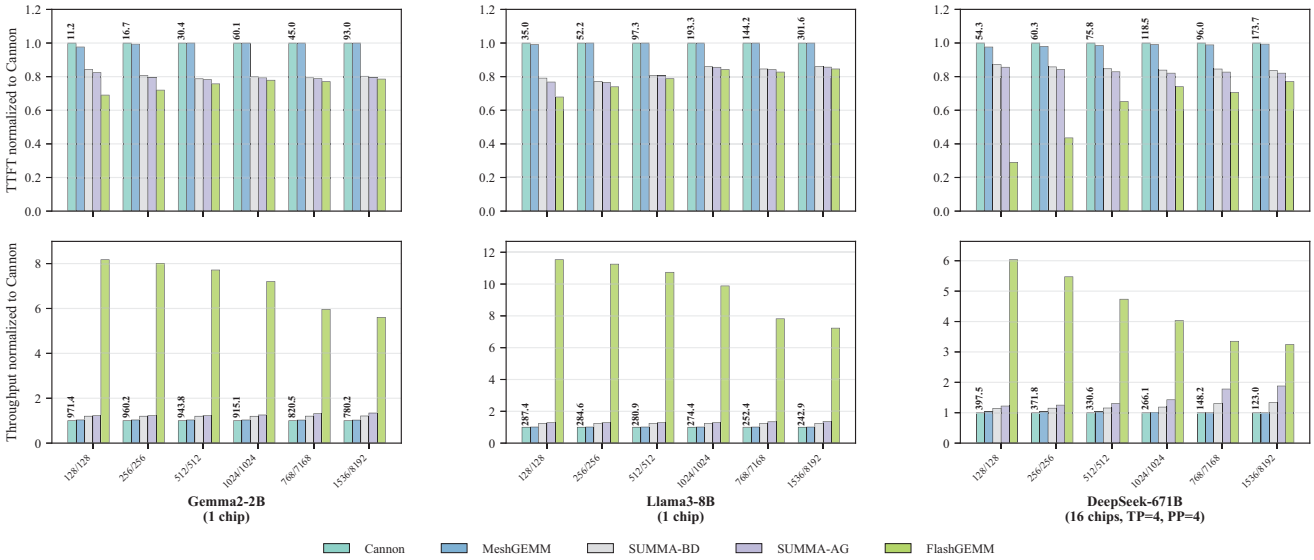


Fig. 8 Performance comparison of distributed GEMM algorithms on 3D-stacked LLM accelerators. The top row of bar graphs reports TTFT normalized to Cannon (lower is better), and the bottom row reports throughput normalized to Cannon. Numbers above the bars denote absolute performance, i.e., TTFT in milliseconds and throughput in tokens per second. The notation 768/7168 indicates input and output sequence lengths of 768 and 7168 tokens, respectively.

ring communication pattern described in Section III-C. Note that if the decoding phase adopts the weight-stationary dataflow, the KV cache will remain stationary, similar to the weight matrix $\boldsymbol{B}$.

In LLM inference, FlashGEMM can dynamically select the output-stationary or weight-stationary dataflow with the least communication overhead. The weight-stationary scheme also exploits the low latency of 3D-stacked DRAM. Similar to how output-stationary bypasses SRAM for the $\boldsymbol{C}$ matrix (Section III-B2), weight-stationary bypasses SRAM for the $\boldsymbol{B}$ matrix. Together, these strategies reduce data movement in 3D-stacked LLM accelerators.

IV. Evaluation

This section presents the experiments. First, we introduce experimental methodology in Section IV-A. Then, we provide performance results and analysis in Section IV-B and Section IV-D.

A. Experimental Methodology

We detail the implementations, LLM workloads, and baselines.

We implement the simulation framework in over 33K lines of Python code based on Calculon [31]. We use cycle-accurate results from Scale-Sim and BookSim2 to calibrate and validate the outputs of the analytical model, with an average error of 13.29% [32], [33]. The 3D-stacked LLM accelerator integrates 4 DRAM dies, providing a total memory capacity of 64 GB and an aggregate bandwidth of 24 TB/s. The architecture delivers a peak computational power of 1024 TFLOPS at FP8 precision and the NoC achieves a bisection bandwidth of 1 TB/s. The PE array consists of 16×16 PEs, with 0.25 GB DRAM bank for local data reuse. The latency of a single router is 3 ns. For system scale-up, the architecture employs an all-to-all interconnect topology, achieving a per-link chip-to-chip bandwidth of 800 GB/s [7].

The LLMs evaluated are listed in TABLE I, covering dense models (e.g., Llama-3, Gemma2 [34], [35]) and Mixture-of-Experts (MoE) models (e.g., DeepSeek [1]). Evaluation is driven by a production trace capturing realistic LLM usage patterns, with input lengths ranging from 128 to 1536 tokens and output lengths from 128 to 8192 tokens [36]. A default batch size of 8 is used, which reflects a common configuration in production LLM serving systems.

TABLE I The LLMs used for experiments.

Model	Parameters	Vocab size	Context window	Layers	Attention [1]
Llama3	8B/70B/405B	125K	8K	126	MHA & GQA
Gemma2	2B	250K	8K	26	GQA
Qwen2	72B	148K	32K	80	GQA
DeepSeek-v3	671B	126K	128K	61	MLA

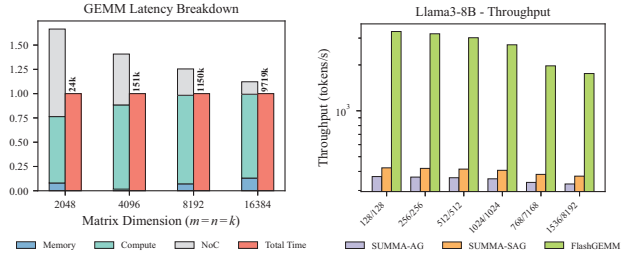
[1] MLA are short for multi-latent attention.

We adopt two kinds of baselines. First, for distributed GEMM algorithms, we employ Cannon, MeshGEMM, SUMMA with broadcast (SUMMA-BD), and SUMMA with naive ring all-gather (SUMMA-AG). Second, we use Calculon as the baseline to compare the difference between HBM and 3D-stacked DRAM-enabled architectures [31]. Calculon supports NVIDIA H100 platforms, supporting the evaluation of HBM-enabled systems. Notably, the 3D-stacked LLM accelerator achieves 51.74% of the peak compute performance of the H100. This stems from a conservative estimate of the silicon area available for matrix computation in the 3D-stacked LLM accelerators.

B. LLM Performance with Different GEMM Algorithm

Fig. 8 shows the impact of distributed GEMM algorithms on prefill and decode performance for different LLMs under the 3D-stacked architecture. FlashGEMM dynamically select output stationary or weight-stationary dataflow, whereas all other methods adopts an output-stationary dataflow. All algorithms implement communication-computation overlap through tiling to ensure a fair comparison. Across all models, SUMMA-based algorithms outperform Cannon-based algorithms since they avoid an initialization phase.

During prefill, FlashGEMM outperform all baselines. FlashGEMM achieves on average 1.23∼1.50× lower TTFT than Cannon (1.50×), MeshGEMM (1.49×), SUMMA-BD (1.25×), and SUMMA-AG (1.23×). Relative to Cannon, FlashGEMM delivers 1.28×,



(a) (b)

Fig. 9 (a) Latency breakdown and execution time of GEMM. Numbers above the bars indicate execution time in nanoseconds. Latency breakdown normalized to the total execution time. (b) LLM throughput under different communication optimization settings.

1.34×, and 1.89× speedups on Llama3-8B, Gemma2-2B, and DeepSeek-671B, respectively. This trend arises because GEMM is compute-bound in prefill, where large batches, long sequences, and large matrix multiplication limit the benefit of communication optimizations.

During decode, FlashGEMM achieves significant throughput improvements over baselines. Experiments show that FlashGEMM delivers an average throughput improvement of 7.11×, 6.98×, 5.84× and 5.44× over Cannon, MeshGEMM, SUMMA-BD and SUMMA-AG, respectively. Additionally, on Gemma2-2B, Llama3-8B, and DeepSeek-671B, FlashGEMM achieves average speedups of 6.39×, 8.66×, and 3.97× over Cannon. This indicates that FlashGEMM achieves greater speedup over other GEMM algorithms on large dense models than on small dense or MoE models.

C. Ablation Study

GEMM Breakdown. Fig. 9(a) shows the latency breakdown and total execution time of GEMM with different matrix dimensions. The latency breakdown shows contributions from computation, NoC communication, and memory access, representing performance without computation-communication overlap. With the optimal tiling factor, FlashGEMM achieves computation-communication overlap and delivers 1.12∼1.66× speedup for GEMM with varying matrix dimensions.

Impact of Communication Optimization. Fig. 9 shows the effect of communication optimizations on the throughput of Llama3-8B. SUMMA-SAG denotes replacing the naive ring all-gather with the skipped ring all-gather. FlashGEMM builds on SUMMA-SAG and dynamically selects between output- and weight-stationary dataflows based on matrix dimensions. Experiments show that FlashGEMM and SUMMA-SAG deliver 7.44× and 1.14× average throughput improvements over SUMMA-AG, respectively. The improvements demonstrate the effectiveness of communication optimization in FlashGEMM.

D. Comparison with NVIDIA H100

Fig. 10 compares the performance of Llama3-8B for the 3D-stacked LLM accelerator and NVIDIA H100 simulation results. With FlashGEMM, the 3D-stacked LLM accelerator achieves an average of 1.57× faster inference and 1.46× higher throughput than the H100. However, it demonstrates limitations in TTFT performance. The reason lies in that TTFT performance is primarily determined by compute power. The computation resources of 3D-stacked accelerators is approximately half that of the H100 (Section IV-A).

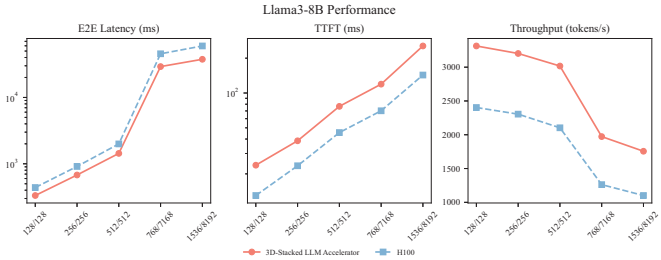


Fig. 10 Performance comparison between Calculon H100 and 3D-stacked LLM accelerators.

V. Conclusion

In this paper, we propose FlashGEMM, an efficient matrix multiplication for 3D-stacked LLM accelerators with mesh topology. FlashGEMM addresses the inefficiency and poor scalability of previous distributed GEMM algorithms with a two-level tiling strategy and optimized dataflows. FlashGEMM reduces TTFT by up to 1.50× and improves throughput by up to 7.11× over state-of-the-art GEMM baselines. Furthermore, despite utilizing only 51.74% of the computational power of H100's FP8, with FlashGEMM, the 3D-stacked LLM accelerator achieves 1.57× lower latency than H100.

Acknowledgment

This research was partially supported by AI Chip Center for Emerging Smart Systems (ACCESS, supported by the InnoHK initiative of the Innovation and Technology Commission of the Hong Kong Special Administrative Region Government), and Research Grants Council of Hong Kong SAR (16213824 & 16212825).

References

[1] A. Liu, B. Feng, B. Xue, B. Wang, B. Wu, C. Lu, C. Zhao, C. Deng, C. Zhang, C. Ruan *et al.*, "DeepSeek-V3 Technical Report," *arXiv preprint arXiv:2412.19437*, 2024.

[2] OpenAI, "OpenAI o3 and o4-mini System Card," *https://cdn.openai.com/pdf/2221c875-02dc-4789-800b-e7758f3722c1/o3-and-o4-mini-system-card.pdf*, 2025.

[3] K. Team, Y. Bai, Y. Bao, G. Chen, J. Chen, N. Chen, R. Chen, Y. Chen, Y. Chen, Y. Chen *et al.*, "Kimi K2: Open Agentic Intelligence," *arXiv preprint arXiv:2507.20534*, 2025.

[4] K. Hong, G. Dai, J. Xu, Q. Mao, X. Li, J. Liu, K. Chen, Y. Dong, and Y. Wang, "FlashDecoding++: Faster Large Language Model Inference with Asynchronization, Flat GEMM Optimization, and Heuristics," *Machine Learning and Systems (MLSys)*, vol. 6, pp. 148–161, 2024.

[5] H. Liao, J. Tu, J. Xia, H. Liu, X. Zhou, H. Yuan, and Y. Hu, "Ascend: a Scalable and Unified Architecture for Ubiquitous Deep Neural Network Computing: Industry track paper," in *IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2021, pp. 789–801.

[6] J. Choquette, W. Gandhi, O. Giroux, N. Stam, and R. Krashinsky, "NVIDIA A100 Tensor Core GPU: Performance and Innovation," *IEEE Micro*, vol. 41, no. 2, pp. 29–35, 2021.

[7] J. Choquette, "NVIDIA Hopper H100 GPU: Scaling Performance," *IEEE Micro*, vol. 43, no. 3, pp. 9–17, 2023.

[8] N. P. Jouppi, G. Kurian, S. Li, P. Ma, R. Nagarajan, L. Nai, N. Patil, S. Subramanian, A. Swing, B. Towles *et al.*, "TPU v4: An Optically Reconfigurable Supercomputer for Machine Learning with Hardware Support for Embeddings," *IEEE/ACM International Symposium on Computer Architecture (ISCA)*, 2023.

[9] A. Smith, G. H. Loh, M. J. Schulte, M. Ignatowski, S. Naffziger, M. Mantor, M. F. N. Kalyanasundharam, V. Alla, N. Malaya, J. L. Greathouse *et al.*, "Realizing the AMD Exascale Heterogeneous Processor Vision: Industry Product," in *IEEE/ACM International Symposium on Computer Architecture (ISCA)*. IEEE, 2024, pp. 876–889.

[10] S. Lie, "Cerebras Architecture Deep Dive: First Look Inside the Hardware/Software Co-Design for Deep Learning," *IEEE Micro*, vol. 43, no. 3, pp. 18–30, 2023.

[11] E. Talpes, D. D. Sarma, D. Williams, S. Arora, T. Kunjan, B. Floering, A. Jalote, C. Hsiong, C. Poorna, V. Samant *et al.*, "The Microarchitecture of DOJO, Tesla's Exa-Scale Computer," *IEEE Micro*, vol. 43, no. 3, pp. 31–39, 2023.

[12] J. H. Lau, "Recent Advances and Trends in Cu–Cu Hybrid Bonding," *IEEE Transactions on Components, Packaging and Manufacturing Technology (TCPMT)*, vol. 13, no. 3, pp. 399–425, 2023.

[13] D. Ingerly, S. Amin, L. Aryasomayajula, A. Balankutty, D. Borst, A. Chandra, K. Cheemalapati, C. Cook, R. Criss, K. Enamul *et al.*, "Foveros: 3D Integration and the Use of Face-to-Face Chip Stacking for Logic Devices," in *IEEE International Electron Devices Meeting (IEDM)*. IEEE, 2019, pp. 19–6.

[14] M.-F. Chen, F.-C. Chen, W.-C. Chiou, and C. Doug, "System on Integrated Chips (SoIC (TM)) for 3D Heterogeneous Integration," in *IEEE Electronic Components and Technology Conference (ECTC)*. IEEE, 2019, pp. 594–599.

[15] C. Bai, X. Fan, Z. Zhu, W. Zhang, and Y. Xie, "AccelStack: A Cost-Driven Analysis of 3D-Stacked LLM Accelerators," in *2025 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 2025, pp. 1–9.

[16] B. Fujun, J. Xiping, W. Song, Y. Bing, T. Jie, Z. Fengguo, W. Chunjuan, W. Fan, L. Xiaodong, Y. Guoqing *et al.*, "A Stacked Embedded DRAM Array for LPDDR4/4X using Hybrid Bonding 3D Integration with 34GB/s/1Gb 0.88 pJ/b Logic-to-memory Interface," in *IEEE International Electron Devices Meeting (IEDM)*. IEEE, 2020, pp. 6–6.

[17] J. Lee, K. Cho, C. K. Lee, Y. Lee, J.-H. Park, S.-H. Oh, Y. Ju, C. Jeong, H. S. Cho, J. Lee *et al.*, "A 48GB 16-high 1280GB/s HBM3E DRAM with All-around Power TSV and a 6-phase RDQS Scheme for TSV Area Optimization," in *IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 67. IEEE, 2024, pp. 238–240.

[18] P. Huang, C. Lu, W. Wei, C. Chiu, K. Ting, C. Hu, C. Tsai, S. Hou, W. Chiou, C. Wang *et al.*, "Wafer Level System Integration of the Fifth Generation CoWoS®-S with High Performance Si Interposer at 2500 mm2," in *IEEE Electronic Components and Technology Conference (ECTC)*. IEEE, 2021, pp. 101–104.

[19] "JEDEC Standard: High Bandwidth Memory (HBM3) DRAM, JEDEC Solid State Technology Association," 2023, https://www.jedec.org/standards-documents/docs/jesd238a.

[20] M. D. Schatz, R. A. van de Geijn, and J. Poulson, "Parallel Matrix Multiplication: A Systematic Journey," *SIAM J. Sci. Comput.*, pp. C748–C781, Jan. 2016.

[21] H. Nam, G. Gerogiannis, and J. Torrellas, "MeshSlice: Efficient 2D Tensor Parallelism for Distributed DNN Training," in *IEEE/ACM International Symposium on Computer Architecture (ISCA)*, 2025, pp. 821–834.

[22] R. A. Van De Geijn and J. Watts, "SUMMA: Scalable Universal Matrix Multiplication Algorithm," *Concurrency: Practice and Experience*, vol. 9, no. 4, pp. 255–274, 1997.

[23] C. He, Y. Huang, P. Mu, Z. Miao, J. Xue, L. Ma, F. Yang, and L. Mai, "WaferLLM: Large Language Model Inference at Wafer Scale," *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2025.

[24] J. Ainslie, J. Lee-Thorp, M. De Jong, Y. Zemlyanskiy, F. Lebron, and S. Sanghai, "GQA: Training Generalized Multi-query Transformer Models from Multi-head Checkpoints," *arXiv preprint arXiv:2305.13245*, 2023.

[25] N. Shazeer, "GLU Variants Improve Transformer," *arXiv preprint arXiv:2002.05202*, 2020.

[26] J. Li, B. Zhang, Y. Sun, W. Xing, and Y. Cheng, "Cool3D: Cost-Optimized and Efficient Liquid Cooling for 3D Integrated Circuits," in *IEEE/ACM Proceedings Design, Automation and Test in Eurpoe (DATE)*. IEEE, 2025, pp. 1–7.

[27] K. Sakuma, R. Yu, N. Polomoff, A. Kumar, S. Raghavan, M. Belyansky, A. Jain, R. Bonam, Y. Sulehria, H. Hsu *et al.*, "D2W and W2W Hybrid bonding system with below 2.5 micron pitch for 3D chiplet AI applications," in *IEEE International Electron Devices Meeting (IEDM)*. IEEE, 2024, pp. 1–4.

[28] R. Pope, S. Douglas, A. Chowdhery, J. Devlin, J. Bradbury, J. Heek, K. Xiao, S. Agrawal, and J. Dean, "Efficiently Scaling Transformer Inference," *Machine Learning and Systems (MLSys)*, vol. 5, pp. 606–624, 2023.

[29] A. Weingram, Y. Li, H. Qi, D. Ng, L. Dai, and X. Lu, "xCCL: A Survey of Industry-Led Collective Communication Libraries for Deep Learning," *Journal of Computer Science and Technology*, vol. 38, no. 1, pp. 166–195, 2023.

[30] J. Chen, S. Li, R. Guo, J. Yuan, and T. Hoefler, "AutoDDL: Automatic Distributed Deep Learning with Near-Optimal Bandwidth Cost," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, vol. 35, no. 8, pp. 1331–1344, 2024.

[31] M. Isaev, N. McDonald, L. Dennison, and R. Vuduc, "Calculon: a Methodology and Tool for High-Level Codesign of Systems and Large Language Models," in *IEEE/ACM International Conference for High Performance Computing, Networking, Storage, and Analysis (SC)*, 2023, pp. 1–14.

[32] A. Samajdar, Y. Zhu, P. Whatmough, M. Mattina, and T. Krishna, "Scale-Sim: Systolic CNN Accelerator Simulator," *arXiv preprint arXiv:1811.02883*, 2018.

[33] N. Jiang, D. U. Becker, G. Michelogiannakis, J. Balfour, B. Towles, D. E. Shaw, J. Kim, and W. J. Dally, "A Detailed and Flexible Cycle-Accurate Network-on-Chip Simulator," in *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2013, pp. 86–96.

[34] T. Mesnard, C. Hardin, and et al., "Gemma: Open Models Based on Gemini Research and Technology," *arXiv preprint arXiv:2403.08295*, 2024.

[35] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar *et al.*, "Llama: Open and Efficient Foundation Language Models," *arXiv preprint arXiv:2302.13971*, 2023.

[36] H. Liu, C. Bai, P. Xu, L. Yin, X. Yu, H.-L. Zhen, M. Yuan, T.-Y. Ho, and B. Yu, "LLMShare: Optimizing LLM Inference Serving with Hardware Architecture Exploration," *ACM/IEEE Design Automation Conference (DAC)*, 2025.