I/O-Aware PIM Acceleration for Long-Sequence LLM Inference with Hybrid Sparse Attention

Xiaoyang Lu[1*], Lihan Hu[2*], Hongrui Huang[3], Peng Jiang[2], Xian-He Sun[1]
[1]Illinois Institute of Technology, Chicago, IL, USA
[2]University of Iowa, Iowa City, IA, USA
[3]Columbia University, New York, NY, USA
{xlu40, sun}@illinoistech.edu, {lihan-hu, peng-jiang}@uiowa.edu, hh3084@columbia.edu

***Abstract*—Attention mechanisms are crucial for enabling large language models (LLMs) to process long-sequence inference tasks. However, efficient LLM inference remains challenging due to the quadratic computational complexity of attention in the prefill stage and the memory-bound nature of attention in the decoding stage. Sparse attention techniques partially reduce complexity in the prefill stage, yet substantial data movement between processors and memory (external I/O) continues to limit GPU efficiency in practice. To address these challenges, we propose PILOT, a Processing-in-Memory (PIM)-based, I/O-aware software-hardware co-design that targets memory-bound attention computations. At the hardware level, PILOT strategically integrates processing units within memory to efficiently support hybrid sparse attention in the prefill stage and both dense and sparse attention in the decoding stage, reducing external I/O overhead. At the software level, a detailed I/O analysis guides tiling and scheduling strategies that optimize buffer utilization and minimize internal I/O within the PIM architecture. Finally, we establish a GPU-PILOT heterogeneous system, delivering an efficient end-to-end solution for long-sequence LLM inference. Evaluation results demonstrate that PILOT achieves up to 19.96× and 57.11× speedups over the GPU baseline for sparse attention in the prefill stage and attention in the decoding stage, respectively. The full GPU-PILOT heterogeneous system further improves end-to-end long-sequence LLM inference by up to 5.31× compared to a conventional GPU system.**

***Index Terms*—Processing-in-memory (PIM), I/O analysis, data movement, sparse attention acceleration**

I. Introduction

Large Language Models (LLMs) have significantly advanced the field of artificial intelligence, demonstrating unprecedented performance across diverse applications, including multi-turn dialogue [1], [8], extensive document analysis [16], [56], [59], multi-modal reasoning [13], [28], and automated code generation [10], [30], [38]. The remarkable capabilities of LLMs are predominantly enabled by the attention mechanism, which effectively captures complex, long-range dependencies among tokens.

However, as sequence lengths continue to expand, efficient deployment of LLM inference becomes increasingly challenging. LLM inference involves two stages: the prefill stage, where the attention mechanism processes all input tokens simultaneously, and the decoding stage, where one token is sequentially generated at each decoding step. The standard attention mechanism requires each token to attend to every other token, incurring quadratic computational and memory complexity during the prefill stage. In contrast, the decoding stage, despite its linear complexity, is significantly constrained by its inherently low arithmetic intensity [34]. Together, the substantial latencies introduced by both the prefill and decoding stages pose significant challenges to efficiently deploying LLMs in long-sequence scenarios [36].

To alleviate the computational complexity of attention mechanisms during long-sequence inference, researchers have introduced sparse attention techniques [3], [6], [49], [55], which restrict attention computations to strategically selected subsets of tokens using predefined hybrid sparsity patterns. Widely adopted patterns include sliding-window, random, and global attention. Although these sparse techniques successfully reduce computational complexity from quadratic to nearly linear in the prefill stage, and reduce computation if sparse patterns are also applied during decoding, these reductions do not fully translate into improved performance on modern GPUs. The primary reason is that, under reduced computational complexity, the efficiency of sparse attention operations remains constrained by substantial data movement between GPU compute cores and off-chip memory, and, for the inherently memory-bound decoding stage [34], [53], [54], reducing computation only makes its data-movement bottleneck even worse to achieving efficient long-sequence LLM inference.

Processing-in-Memory (PIM) architectures offer a compelling solution for accelerating memory-bound workloads [9], [15], [33], [34], [48], [52], [58]. By integrating computational units close to memory, PIM efficiently leverages high internal memory bandwidth and significantly reduces costly data movement between the processor and memory. Consequently, PIM emerges as a promising architectural approach for efficient long-sequence LLM inference, addressing memory-bound challenges in both the prefill stage with sparse attention patterns and the decoding stage.

In this work, we propose PILOT, a PIM-based, I/O-aware software-hardware co-design specifically designed to accelerate memory-bound attention computations in long-sequence LLM inference. Prior PIM studies [15], [34] primarily focus on accelerating attention operations during the decoding stage, employing static tiling and scheduling without I/O-aware

*Both authors contributed equally to this work.

optimizations. In contrast, PILOT supports the execution of both hybrid sparse attention in the prefill stage and dense or sparse attention in the decoding stage within PIM, mitigating external I/O (data movement between the GPU and off-chip memory) overhead for memory-bound attention computations. Moreover, PILOT introduces I/O-aware tiling and scheduling strategies derived from a rigorous data-movement analysis, quantitatively modeling and minimizing internal I/O (data transfers between SRAM buffers and DRAM within PIM). Overall, PILOT provides a comprehensive and systematic solution for efficient long-sequence LLM inference.

We observe that the large Query (Q), Key (K), and Value (V) matrices involved in long-sequence attention often exceed the limited SRAM buffer capacity of each processing unit within the PIM architecture, triggering significant internal I/O within the PIM memory hierarchy. Moreover, the hybrid sparse attention patterns used during prefill complicate data reuse, further exacerbating this internal I/O challenge. Therefore, we conduct a systematic I/O analysis of both sparse attention in the prefill stage and attention computations in the decoding stage. Guided by this analysis, at the software level, we introduce I/O-aware tiling and scheduling strategies that maximize data reuse and buffer utilization, minimize internal I/O, and are tailored to the PIM architecture. At the hardware level, PILOT integrates processing units directly within the memory hierarchy, enabling efficient near-memory execution of memory-bound attention computations with hybrid sparse patterns. This design effectively mitigates external I/O bottlenecks present in conventional GPU-based systems. Furthermore, to achieve end-to-end acceleration of LLM inference, we propose a heterogeneous GPU-PIM architecture that strategically assigns memory-bound attention tasks to PILOT while leveraging GPU throughput for compute-intensive operations.

We evaluate PILOT on hybrid sparse attention models [3], [55] and DuoAttention [49], a framework that employs hybrid sparse attention in LLMs. Overall, PILOT consistently outperforms the GPU baseline across all evaluated models. By minimizing both external and internal I/O overhead, PILOT achieves speedups of up to 19.96× and 57.11× for sparse attention in the prefill stage and attention in the decoding stage, respectively. Moreover, the GPU-PILOT heterogeneous system improves end-to-end long-sequence LLM inference by up to 5.31× and 2.71× over the conventional GPU system and the state-of-the-art GPU-PIM system, AttAcc [34], respectively.

In summary, we make the following key contributions:

- We identify that employing sparse patterns in the prefill stage shifts attention computations from being compute-bound to memory-bound, imposing significant performance constraints on GPUs.
- Beyond supporting attention during decoding, we explore the potential of utilizing PIM to efficiently execute static hybrid sparse attention patterns during the prefill stage, thereby providing a comprehensive solution for mitigating costly external I/O overhead.
- We conduct a detailed I/O analysis of memory-bound attention across both inference stages and propose I/O-aware tiling and scheduling strategies to minimize internal I/O overhead within the PIM architecture.
- We propose a GPU-PILOT heterogeneous system architecture, providing a comprehensive end-to-end solution that addresses both external and internal I/O bottlenecks for accelerating long-sequence LLM inference.

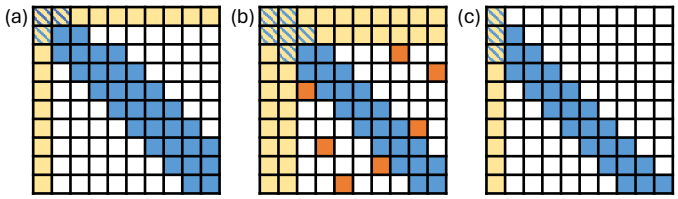


Fig. 1: Hybrid sparse attention patterns used by (a) Longformer [3], (b) BigBird [55], and (c) the streaming head in DuoAttention [49].

II. Background and Motivation

A. LLM Inference and Attention Mechanism

LLMs are transformer-based architectures composed of stacked layers, each containing a multi-head attention mechanism, a feed-forward network (FFN), and normalization components. The inference process consists of two distinct stages: prefill and decoding. In the prefill stage, the model processes the entire input sequence simultaneously to generate internal representations, while in the decoding stage, tokens are generated sequentially based on previously produced outputs. These two stages exhibit different computational behaviors.

Attention mechanisms enable LLMs to capture long-range dependencies among tokens. Each attention head involves three steps: (1) computing the product of Q and K to obtain attention scores, (2) applying a row-wise softmax to produce attention weights, and (3) multiplying these weights by V to generate the final outputs. In the prefill stage, attention primarily involves compute-intensive matrix-matrix multiplications (GEMMs) with quadratic complexity in sequence length. During decoding, each token generation performs matrix-vector multiplications (GEMVs) with low arithmetic intensity, making this stage memory-bound. Consequently, attention becomes the critical performance bottleneck in both stages for long sequences.

B. Sparse Attention Techniques

Models such as Longformer [3] and BigBird [55] employ static hybrid sparse attention patterns exclusively during the prefill stage to reduce the computational complexity of traditional attention from quadratic to linear or sub-quadratic levels. These models enable inference over significantly longer sequences. At the computational level, sparse attention in the prefill stage consists of three steps: (1) computing sparse attention scores via sampled dense-dense matrix multiplication (SDDMM) between Q and K matrices, (2) normalizing the scores with a row-wise softmax, and (3) applying the normalized scores to the V matrix through sparse matrix-matrix

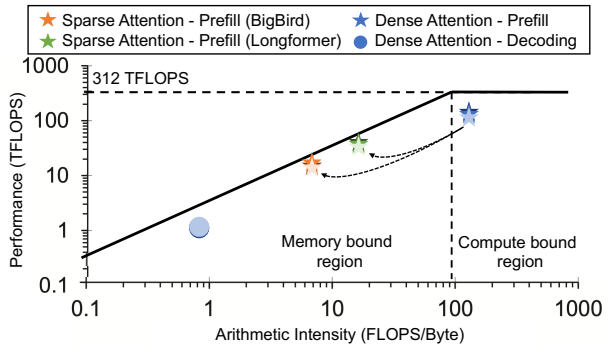


Fig. 2: Roofline analysis of attention computations in LLM inference on an NVIDIA A100 GPU. The plot shows the performance of dense attention and sparse attention for BigBird and Longformer in the prefill stage, as well as dense attention in the decoding stage. Brightness indicates batch sizes: 4 (bright), 16 (moderate), and 32 (dark).

multiplication (SpMM). These models typically follow static hybrid patterns, including sliding-window attention, random attention, and global attention, as shown in Figure 1(a-b). Sliding-window attention restricts the attention computation of each token to a local context. Random attention introduces probabilistically selected attention connections to ensure coverage across the entire sequence. Global attention designates specific tokens to attend broadly, including global rows and global columns, in which all tokens in the sequence attend to specific global tokens. This combination effectively captures broader contextual information, balancing sparsity and accuracy in the model.

Besides training models specifically to support sparse attention patterns, several frameworks have been proposed to apply hybrid sparse attention patterns directly during long-sequence inference [19], [43], [47], [49]. Among them, DuoAttention [49] extends sparse attention techniques comprehensively to both the prefill and decoding stages. DuoAttention identifies critical attention heads (retrieval heads) and applies full attention to them for comprehensive context capturing. The remaining attention heads (streaming heads [50]) employ sparse patterns, such as sliding-window and global columns, as shown in Figure 1(c). Each streaming head in the decoding stage can be regarded as a sequence of sparse matrix-vector multiplications (SpMVs) with the cached key-value patterns. This selective assignment allows DuoAttention to effectively balance efficiency and adaptability, making it particularly suitable for diverse long-sequence inference tasks.

C. Performance Characterization of Attention

While sparse attention techniques substantially reduce computational complexity, their practical effectiveness on GPUs remains challenging. To precisely characterize this issue, we perform a detailed analysis based on the roofline model [46]. Figure 2 presents the roofline analyses for dense and sparse attention computations during the prefill stage, as well as dense attention computations during the decoding stage.

For dense attention, prefill computations exhibit high arithmetic intensity due to compute-intensive GEMMs, thus efficiently utilizing GPU resources. In contrast, sparse attention computations (exemplified by BigBird [55] and Longformer [3]) significantly reduce arithmetic operations, but we observe that they also reduce arithmetic intensity sufficiently to shift execution into the memory-bound region of the roofline model. Consequently, sparse attention in the prefill stage becomes dominated by external I/O [41], [42]. Moreover, irregular memory access patterns and additional indexing overhead further reduce data reuse and degrade effective bandwidth utilization. Therefore, sparsity alone does not fully resolve performance challenges; rather, it introduces new external I/O bottlenecks that must be explicitly addressed. Similarly, attention computations in the decoding stage predominantly involve matrix-vector multiplications, inherently exhibiting low arithmetic intensity, causing these computations to remain deeply memory-bound. Together, these analyses highlight a fundamental limitation of traditional GPUs, as they are primarily optimized for compute-intensive tasks and inherently struggle with memory-bound attention computations in both inference stages.

Observation 1. *Sparse attention shifts attention computations in the prefill stage from compute-bound to memory-bound.*

D. PIM in LLM Inference

PIM architectures integrate processing units near or within memory, effectively mitigating costly external I/O overhead and exploiting massive internal memory bandwidth [2], [9], [15], [25]–[27], [33], [34], [48], [51], [52], [58]. To accelerate LLM inference specifically, recent studies [15], [34], [51] have explored heterogeneous GPU-PIM systems. Such hybrid systems leverage the complementary strengths of GPUs for compute-intensive operations and PIM for memory-bound operations, jointly enhancing the performance of LLM inference.

AttAcc [34] and NeuPIMs [15] specifically utilize HBM-based PIM to accelerate memory-bound attention operations during the decoding stage for long-sequence LLM inference, while offloading dense attention computations in the prefill stage entirely to GPUs. Both systems primarily integrate computational logic close to DRAM banks to efficiently handle memory-bound GEMV operations during decoding. However, none of them provide solutions specifically designed to accelerate memory-bound attention with hybrid sparse patterns in the prefill stage, leaving the full potential of PIM architectures for sparse attention computations unexplored. Consequently, they lack comprehensive support for memory-bound computations across both inference stages.

Observation 2. *Existing studies neglect the potential of PIM to accelerate sparse attention in prefill.*

E. Internal I/O Challenges for Attention in PIM

PIM architectures effectively mitigate external I/O overhead by placing processing units closer to memory. However, naively offloading memory-bound attention computations to

PIM without systematic optimizations introduces new performance challenges. First, hybrid sparse attention patterns, particularly in the prefill stage, exhibit irregular data access patterns and complex data reuse behaviors. Without careful tiling and scheduling explicitly considering data reuse, sparse patterns significantly increase internal I/O within PIM and degrade processing unit utilization. Second, typical PIM designs allocate limited SRAM buffer capacity to each processing unit [29], [34]. The large inputs (Q, K, V) and their even larger intermediate results involved in long-sequence attention frequently exceed this buffer capacity, necessitating unavoidable internal I/O between SRAM buffers and memory in both inference stages of LLMs. Moreover, prior PIM studies [15], [34] employ static tiling and scheduling without I/O-aware optimization, leaving significant opportunities for internal I/O reduction unexplored. Consequently, without a detailed I/O analysis that guides I/O-aware tiling and scheduling, internal I/O becomes a substantial bottleneck, diminishing the gains achieved by reducing external I/O overhead.

Observation 3. *PIM requires I/O-aware tiling and scheduling strategies to effectively mitigate internal I/O overhead.*

Leveraging these observations, we propose PILOT, a PIM-based software-hardware co-design for accelerating memory-bound attention computations in LLM inference. PILOT systematically integrates I/O analysis with tiling and scheduling strategies tailored for hybrid sparse attention in prefill and attention in decoding. PILOT differentiates itself by explicitly addressing both external and internal I/O challenges in attention computations, effectively overcoming limitations identified in existing GPU and PIM solutions. We introduce the detailed design of PILOT in the next section.

III. System and Architecture of PILOT

In this section, we first provide an overview of the heterogeneous computing system that integrates a GPU with our proposed PILOT, an HBM-based PIM accelerator for LLM inference. Figure 3 illustrates the overall GPU-PILOT system architecture. We then present the detailed hardware architecture of PILOT, including its hierarchical organization and key design components.

A. GPU-PILOT Heterogeneous System

In this work, the GPU serves as the host processor within the heterogeneous system. It executes compute-intensive operations such as dense FFNs, non-attention kernels, and GEMMs when dense attention is required for specific heads during the prefill stage, all of which benefit from the high-throughput parallel compute units of the GPU. Additionally, during runtime, the GPU orchestrates workload scheduling, prepares Q, K, and V matrices for offloading, and coordinates data transfers between the GPU and the PILOT-enabled HBM stacks. Similar to prior heterogeneous systems [15], [29], [34], this orchestration imposes negligible overhead on GPU resources. The GPU aggregates intermediate results from PILOT and seamlessly integrates them into the execution of subsequent layers.

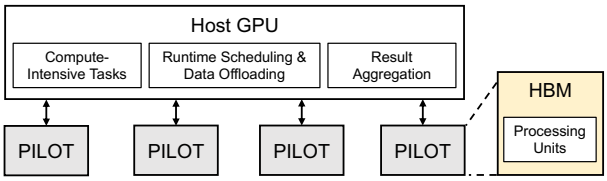


Fig. 3: Overview of the heterogeneous GPU-PILOT system.

PILOT is an HBM-based PIM accelerator designed to accelerate memory-bound attention at the operation level. In the prefill stage, it supports sparse attention, consisting of SDDMM, softmax, and SpMM operations. In the decoding stage, PILOT supports models such as Longformer [3] and BigBird [55], which employ GEMV in decoding, as well as frameworks such as DuoAttention [49], which extend sparsity patterns into decoding, resulting in SpMV operations. In this way, PILOT enables comprehensive acceleration across both inference phases. PILOT reduces external I/O overhead by executing memory-bound attention computations directly within the HBM stack, and minimizes internal I/O within the memory hierarchy through detailed I/O analysis and optimized tiling and scheduling strategies. For scalability, multiple PILOT-enabled HBM stacks can operate in parallel to process large models or multi-batch LLM inference. During execution, the GPU partitions the Q, K, and V matrices and distributes them across the appropriate PILOT-enabled HBM stacks. Each PILOT device independently executes its assigned attention tasks using an I/O-aware tiling and scheduling strategy, and the host then aggregates the outputs for integration into subsequent layers.

All PILOT-enabled HBM stacks are connected directly to the GPU [7], [58]. Each HBM stack consists of multiple DRAM dies stacked on top of a logic die and interconnected by dense through-silicon vias (TSVs). This 3D integration provides significantly higher bandwidth and lower access latency than conventional DRAM modules, which PILOT leverages to efficiently execute memory-bound attention computations. The memory controller handles both standard memory requests and PIM-specific instructions, including control, arithmetic, and data movement [21], [34].

B. PILOT Architecture

We design PILOT based on an 8-Hi HBM3 stack [39], as shown in Figure 4(a). Each HBM3 stack comprises eight vertically stacked DRAM dies interconnected by TSVs. Each DRAM die contains eight pseudo-channels (pCHs) operating independently. Each pCH is further subdivided into four bank groups (BGs), and each BG contains four banks, enabling massive internal bandwidth and parallel data access. To support the operations of attention in both inference stages while maximizing parallelism and minimizing data movement, PILOT integrates simplified compute logic across the hierarchy. As shown in Figure 4(b), its core processing components include bank-level processing modules (BPMs) and bank-group-level processing modules (BGPMs).

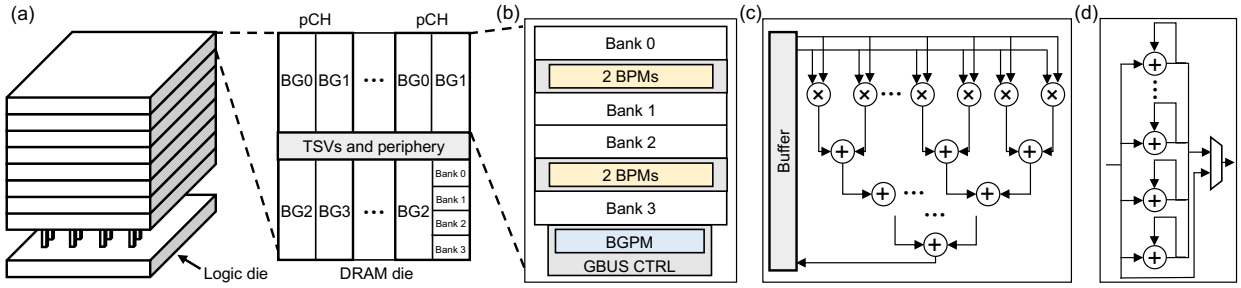


Fig. 4: The architecture of PILOT. (a) Organization of an HBM stack, (b) processing units within HBM (bank-level and BG-level modules), (c) Bank-level processing module (BPM), (d) BG-level processing module (BGPM).

BPM. Figure 4(c) depicts the architecture of a BPM. Inspired by state-of-the-art PIM accelerators [14], [34], the BPM is deployed at the bank level within HBM to enable parallel execution across multiple banks. Each BPM is attached to an individual bank and performs local MAC operations, assisted by an associated SRAM buffer for data reuse. For SDDMM, SpMM, GEMV, and SpMV operations, BPMs perform inner-product computations between two input vectors using operands fetched exclusively from their local buffers, each communicating only with its associated bank. Once operands are ready, parallel multipliers and an adder tree produce results. Additionally, to support local softmax computations, BPMs reuse their arithmetic units to approximate exponentials of attention scores and to locally accumulate row-wise exponential sums for softmax normalization.

BGPM. When the exponentials of attention scores for an entire row are distributed across multiple banks within a BG, accumulation is required to aggregate these values for softmax normalization. To support this, each BG is equipped with a BGPM containing an accumulator integrated directly into the DRAM die, as shown in Figure 4(d). Our mapping strategy ensures that reductions are confined to the BG level, avoiding the need for cross-BG or pCH-level aggregation. In cases where all attention scores for a row reside within a single bank, the BGPM is simply bypassed.

Controller. To support PIM requests, we extend the memory controller with a PIM-specific interface. PILOT augments the HBM command path with a small set of PIM commands. `PLT_CFG` initializes partitioning and mapping metadata for Q, K, and V, while `PLT_MOV` transfers these tiles between DRAM and local BPM buffers. The core compute is triggered by `PLT_MAC`, which invokes BPMs to execute MAC operations within each bank, sourcing operands exclusively from their local buffers. `PLT_EXP` reuses BPM arithmetic to approximate exponentials of attention scores, `PLT_ACC` accumulates row-wise exponentials, and `PLT_SFM` performs the softmax computation. Finally, `PLT_OUT` writes the outputs to DRAM for GPU consumption. These PIM-specific commands are issued through the standard HBM command path, consistent with prior HBM-based PIM systems [34].

C. Execution Flow

During inference, the GPU partitions Q, K, and V and maps them to the PILOT-enabled HBM stacks, where each bank independently executes its portion of the attention workload. In the prefill stage, Q and K are first tiled and scheduled based on I/O analysis, loaded into BPM buffers, and processed by BPMs to perform SDDMM. The resulting score tiles are exponentiated locally within BPMs, and the row-wise sums of exponentials are accumulated across tiles to prepare for softmax normalization. After all exponentials for a head are computed, PILOT performs SpMM using the same tiling and scheduling strategy as in SDDMM, multiplying the attention-weight tiles with the corresponding V tiles. This consistent scheduling across SDDMM and SpMM maximizes data reuse and preserves internal I/O efficiency. In the decoding stage, the execution flow is similar, with BPMs executing GEMV or SpMV operations on Q and K/V. BGPMs then perform lightweight inter-bank reductions to accumulate the row-wise exponentials required for softmax normalization, followed by SpMM with V. Finally, in both stages, the host aggregates the attention outputs from all banks and integrates them into subsequent layers.

IV. Data Mapping and I/O Analysis

To efficiently execute attention within PILOT, data must be strategically mapped to the memory hierarchy, and computations orchestrated through I/O-aware tiling and scheduling to minimize internal I/O overhead. In this section, we first describe our data mapping strategies tailored specifically for attention operations during the prefill and decoding stages. Subsequently, informed by a comprehensive I/O analysis, we propose optimized tiling and scheduling strategies designed to maximize data reuse within constrained buffer capacities while accommodating hybrid attention patterns.

A. Data Mapping

Sparse Attention in Prefill. Each sparse attention head is assigned to a distinct BG to exploit parallelism across multiple heads and eliminate inter-BG communication overhead. As shown in Figure 5(a), within each BG, hybrid sparse attention patterns are classified into two categories: *non-global rows*, which primarily follow sliding-window, global columns, and random attention patterns, and *global rows*, whose query

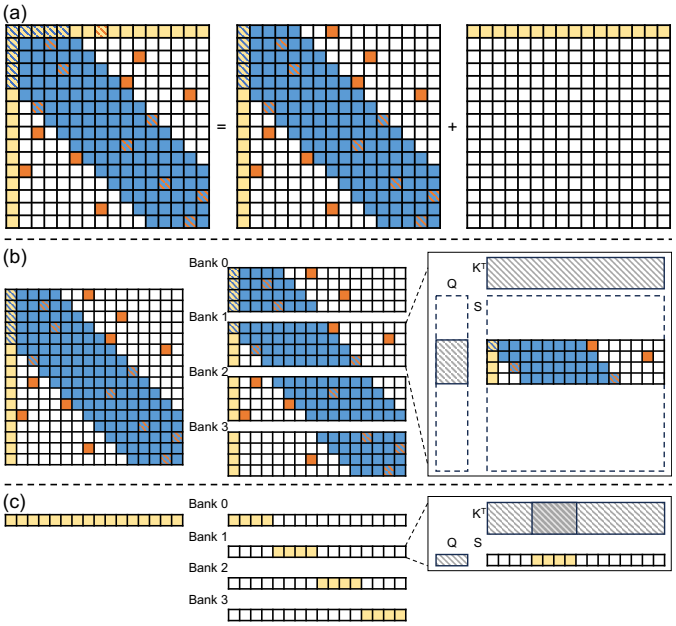


Fig. 5: Mapping strategies of Q and K for prefill with hybrid sparse attention in PILOT. (a) Hybrid sparse attention is divided into non-global and global rows. (b) Non-global rows: the Q matrix is partitioned row-wise across banks, with the K matrix replicated across all banks. (c) Global rows: the Q vectors are replicated across banks, while the K^T matrix is partitioned column-wise. Since V in attention is the transposed form of K^T, its mapping follows the reverse of K^T.

vectors attend to all positions in the sequence. Given the fundamental differences in computational scope and data demands between these categories, PILOT adopts distinct mapping strategies. Specifically, non-global rows are mapped to maximize data locality and minimize data movement among banks, whereas global rows are strategically mapped to distribute computation evenly across banks, mitigating load imbalance.

Non-Global Rows. As shown in Figure 5(b), for non-global rows of an attention head, the Q matrix is partitioned row-wise across banks, with each bank assigned a disjoint subset of rows. The K and V matrices are duplicated across all banks, ensuring that the sparse attention scores for each row can be computed entirely within its assigned bank. This eliminates remote accesses and avoids cross-bank communication. Each bank independently computes the attention scores for its rows and applies the row-wise softmax locally, ensuring that each attention row is fully resolved within a single bank.

Global Rows. As illustrated in Figure 5(c), global rows represent a special case of attention, where each query vector attends to all key and value positions. Mapping them to a single bank would cause load imbalance. To mitigate this, global rows are handled separately and distributed across banks. Specifically, the relatively small Q vectors for global rows are replicated across all banks, while the K^T and V matrices are partitioned column-wise. Each bank independently computes its assigned slice of the output, and a lightweight inter-bank reduction performed by the BGPM accumulates the row-wise exponential sums. This mapping balances workload while minimizing redundant I/O between banks.

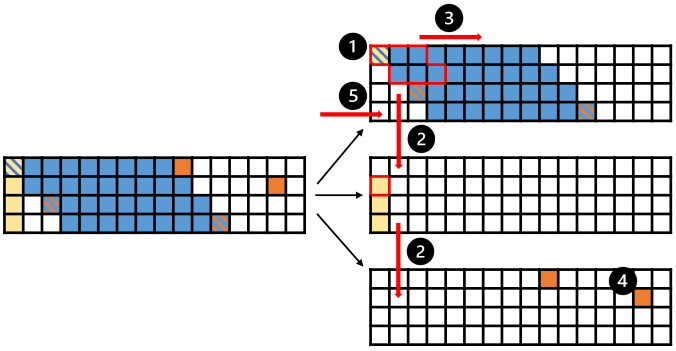


Fig. 6: Per-bank Q-stationary scheduling for SDDMM in sparse attention during the prefill stage for non-global rows.

Attention in Decoding. Attention in decoding is treated as a special case of global rows, since a single query vector attends to all key and value positions. All banks within a BG collaboratively process one attention head. The query vector is replicated to every bank, while the K^T and V matrices are partitioned column-wise. Each bank performs its local GEMV operations independently, followed by a lightweight inter-bank reduction to accumulate the row-wise exponential sum. When sparse patterns are applied in decoding (e.g., DuoAttention [49]), the same column-wise partitioning strategy is employed, ensuring consistency with the dense case.

B. I/O-Aware Tiling and Scheduling

Scheduling for Non-Global Rows in Prefill. To efficiently execute sparse attention for non-global rows during the prefill stage, PILOT adopts a per-bank Q-stationary scheduling for SDDMM. PILOT begins with the sliding-window pattern. As illustrated in Figure 6, each bank loads a Q-tile and a K-tile into its local SRAM buffer, where tile sizes are selected based on I/O analysis to maximize the compute-to-I/O ratio under limited buffer capacity (❶). Once the corresponding sliding-window computation is completed, PILOT immediately exploits opportunistic reuse of inputs: whenever Q- and K-tiles coexist in SRAM, they are reused to apply available global-column and random attention patterns, thereby avoiding redundant internal I/O (❷). After reuse, the Q-tile remains stationary while K-tiles are streamed sequentially across the sliding-window band to fully exploit the Q-tile already in the buffer (❸). When all K-tiles in the window have been processed, any remaining global-column or random patterns that can reuse the current Q-tile are executed in batches before evicting the tile (❹). Once the Q-tile has been fully reused, the system advances to the next Q-tile and repeats the process (❺). Overall, this Q-stationary scheduling for SDDMM minimizes internal I/O within the limited buffer capacity by combining I/O-aware tiling and scheduling, which naturally fuses sliding-window, global-column, and random patterns to maximize data reuse and reduce internal I/O

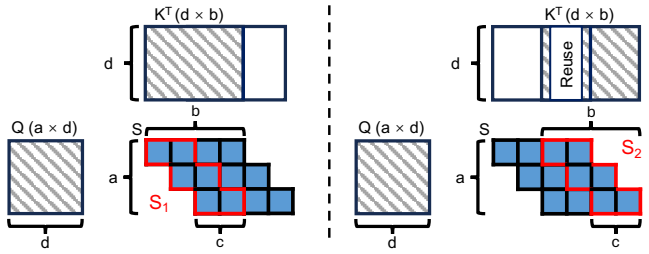


Fig. 7: I/O-aware tiling for sliding-window attention.

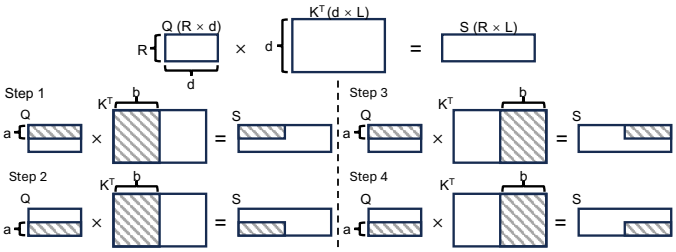


Fig. 8: Per-bank K-stationary scheduling for SDDMM in sparse attention during the prefill stage for global rows.

overhead. Similarly, for SpMM we adopt an output-stationary schedule, where partial outputs remain in the local buffer while attention-weight tiles and V tiles are streamed in tile-by-tile. This schedule can be viewed as the reverse of SDDMM and applies the same I/O-aware tile sizes to maximize reuse and minimize internal I/O. In this subsection, we primarily focus on the scheduling and tiling for SDDMM.

I/O-Optimal Tiling for Sliding-Window. Sliding-window attention is the dominant component of non-global rows, and its efficiency is largely determined by how Q and K^T are tiled and reused under the limited buffer capacity. To analyze this, we consider a Q tile of size $a \times d$ and a K^T tile of size $d \times b$, as illustrated in Figure 7. Each Q tile is reused across a sequence of subcomputations $S_1, S_2, \ldots, S_n$. Therefore, in each subcomputation S_i, the Q tile corresponds to a rows in the attention pattern, consumes b columns of K^T, and each row attends to c positions, resulting in ac attention scores.

Buffer Constraint. Let the buffer capacity be M elements. To execute a subcomputation S_i, the buffer must simultaneously store: ad elements for the Q tile, bd elements for the K^T tile, and ac elements for the attention scores of the current subcomputation. Thus, the buffer capacity constraint must be satisfied as:

$$ad + bd + ac \;\le\; M \tag{1}$$

Due to the specific characteristics of the sliding-window pattern, we have $b = (a-1) + c$. Thus, Equation 1 tightly couples the Q tile height a and the K^T tile width b with the buffer capacity M.

Compute-to-I/O Ratio. We define ρ_i as the compute-to-I/O ratio of a subcomputation S_i. Each S_i performs acd operations (MACs). For I/O between the buffer and the local bank, note that each Q tile is reused across a sequence of r subcomputations. The reuse factor r is calculated as

$$r = \frac{w}{c}, \tag{2}$$

where w is the sliding-window size. Thus, the amortized I/O cost of a Q tile per subcomputation S_i is $\frac{ad}{r}$. For any S_i with $i \neq 1$, a portion of the columns of K^T is reused between S_{i-1} and S_i due to overlap, as illustrated in Figure 7. Therefore, only cd new elements of K^T need to be loaded for S_i. The total I/O loading cost of S_i (for $i \neq 1$) is then

$$\text{I/O}_i = \frac{ad}{r} + cd. \tag{3}$$

Accordingly, the compute-to-I/O ratio of S_i (for $i \neq 1$) is

$$\rho_i = \frac{acd}{\frac{ad}{r} + cd} = \frac{ac}{\frac{a}{r} + c}. \tag{4}$$

Considering Equations 2 and 4, we further simplify ρ_i (for $i \neq 1$) as

$$\rho_i = \frac{a}{\frac{a}{w} + 1}. \tag{5}$$

Equation 5 shows that larger tile height a improves ρ_i, since w is a constant determined by the sparse attention technique itself.

I/O-Optimal Tiling Size. From the buffer constraint in Equation 1, the maximum feasible a occurs when $c = 1$, i.e., when each row in S_i attends to a single position. In this case, $b = (a-1) + c = a$, and the buffer constraint simplifies to

$$ad + ad + a \;\le\; M. \tag{6}$$

Thus, the maximum feasible a is bounded as

$$a \;\le\; \frac{M}{2d+1}. \tag{7}$$

Accordingly, the I/O-optimal tiling configuration that achieves the maximum compute-to-I/O ratio, while considering the buffer capacity M for the sliding-window pattern, is

$$a = b = \left\lfloor \frac{M}{2d+1} \right\rfloor. \tag{8}$$

Scheduling for Global Rows in Prefill. For global rows in sparse attention during the prefill stage, each query attends to all key positions. To efficiently execute such computations, PILOT adopts a per-bank K-stationary scheduling, as illustrated in Figure 8. For each bank-level computation, the Q matrix of size $R \times d$ (where R is the number of global rows) is multiplied by a K^T matrix of size $d \times L$ (where L is the total number of columns mapped to the current bank for global rows). As shown in Figure 8, a K^T tile of width b is first loaded into the local buffer. PILOT then traverses all Q tiles one by one: each Q tile of height a is loaded into SRAM and multiplied with the resident K^T tile, producing a submatrix of attention scores. The tile sizes a and b are determined by maximizing the compute-to-I/O ratio under the buffer capacity constraint M. Once all Q tiles are processed for the current K^T tile, the K^T tile is evicted and the next K^T tile is loaded. Thus, each K^T tile is loaded exactly once and reused across all global

rows, while the relatively small Q tiles (since $R \ll L$) can be efficiently reloaded as necessary.

I/O-Optimal Tiling for Global Rows. Let ρ_i denote the compute-to-I/O ratio for one subcomputation using a K^T tile. Each subcomputation performs abd MACs, where a and b represent the tile height and width of Q and K^T, respectively, as illustrated in Figure 8. The corresponding I/O cost consists of loading a Q tile (ad elements) and a fraction $\frac{1}{r}$ of a K^T tile (bd elements of K^T, is reused by r subcomputations). Therefore, the compute-to-I/O ratio ρ_i is defined as

$$\rho_i = \frac{abd}{ad + \frac{bd}{r}} = \frac{ab}{a + \frac{b}{r}}. \tag{9}$$

The reuse factor is expressed as

$$r = \frac{R}{a}. \tag{10}$$

Substituting Equation 10 into Equation 9 yields

$$\rho_i = \frac{ab}{a + \frac{ab}{R}} = \frac{bR}{R + b}. \tag{11}$$

Given the SRAM capacity M, the buffer must accommodate Q tiles (ad elements), K^T tiles (bd elements), and partial attention results (ab elements) for each subcomputation:

$$ad + bd + ab \ \leq \ M. \tag{12}$$

To maximize ρ_i, b should be as large as possible while satisfying this constraint. The upper bound occurs when the Q tile height is minimized ($a = 1$), yielding

$$d + bd + b \ \leq \ M. \tag{13}$$

Hence, the optimal K^T tile width b is determined as

$$b = \left\lfloor \frac{M - d}{d + 1} \right\rfloor. \tag{14}$$

This tiling configuration ensures that each K^T tile achieves maximal reuse across all global rows within the available buffer, effectively minimizing redundant internal I/O.

Attention in Decoding. In the decoding stage, both dense and sparse attention process only a single query vector at a time; therefore, there is no reuse of K^T tiles between GEMV-based subcomputations. PILOT adopts a scheduling and tiling strategy similar to that described for global rows. Specifically, dense decoding represents a special case in which only one query vector is processed ($R = 1$). The tiling strategy for K^T defined in Equation 14 still applies, determining the K^T tile size that fully utilizes the available buffer capacity.

C. Softmax Implementation

PILOT implements a lightweight, hardware-friendly approach to approximate the exponential function used in softmax. Inspired by [40], the exponential term is decomposed as $e^{Int+Frac} \approx 2^{Int} \times e^{Frac}$, where the integer part is approximated by a base-two exponential computed efficiently using bit-shift operations. The fractional part e^{Frac} is evaluated using a hybrid method that combines a low-order Taylor expansion [45] for $Frac < 0.5$ and a piecewise-linear approximation with a small lookup table [57] for $Frac \geq 0.5$. This hybrid approximation enables each bank to perform exponential operations locally by maximally reusing existing BPM arithmetic units, thereby achieving low-cost computation with negligible accuracy loss.

TABLE I: Architectural parameters for PILOT.

Component	Configuration
HBM	4 HBM3, 8 dies per HBM, 8 pCHs per die, 4 BGs per pCH, 4 banks per BG, 32K rows per bank, 1KB per row
BPM	666 MHz, 1 per bank, 16 FP16 adders and 16 FP16 multipliers, 2KB SRAM buffer
BGPM	666 MHz, 1 per BG, 16 FP16 adders per accumulator

V. Evaluation

A. Experimental Setup

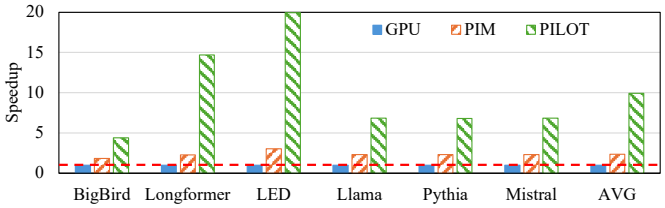
Hardware Configuration. The hardware configuration of PILOT is summarized in Table I. PILOT is built on HBM3 [39] operating at a data rate of 5.2 Gbps per pin and consists of four HBM3 stacks. To support end-to-end LLM inference, we establish a heterogeneous GPU-PILOT system by integrating PILOT with a single NVIDIA A100 GPU. While this configuration is used for prototyping, it is orthogonal to specific design choices and can be adapted to different system configurations for varying model requirements.

Baselines. We define two categories of baselines: one set for evaluating the performance of PILOT specifically on memory-bound attention computations, and another for assessing the overall end-to-end inference performance of GPU-PILOT for long-sequence LLM inference. For attention performance evaluation, we compare PILOT against two baselines: (1) GPU, an NVIDIA A100 GPU with its memory upgraded to HBM3 for fair comparison [34]; and (2) PIM, a variant of PILOT with identical hardware configurations but without the proposed I/O-aware tiling and scheduling strategies. For end-to-end evaluation, we compare GPU-PILOT against two baselines: (1) the same GPU configuration as above; and (2) AttAcc [34], which offloads attention to PIM only during decoding while handling prefill on GPUs. For fairness, we scale AttAcc to match the GPU and HBM counts of GPU-PILOT and the operating frequency of its bank-level processing units.

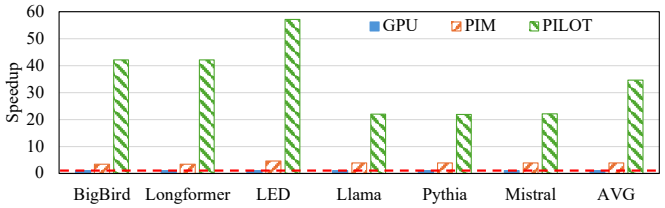
Simulation Setup. We evaluate performance using a cycle-accurate simulator built upon an established simulation methodology [34]. The simulator integrates detailed performance models for both GPU-based computations and PIM components, and all HBM timing parameters are derived from DRAMSim3 [24]. It takes system configurations and model parameters as input and reports execution time as well as data movement statistics. In addition, to evaluate the area and power of customized components, we synthesize the designs in 65 nm technology using Synopsys Design Compiler and scale the results to a 10 nm-class (1z-nm) DRAM based on the HBM3 memory technology [34], [35], [39].

TABLE II: Configuration Details of Evaluated Models.

Parameters	Batch Size	L_{in}	L_{out}	Hidden Size	Sparse Pattern
BigBird [55]	512	4096	128	12*64	192+64+64
Longformer [3]	512	4096	256	12*64	512+1+0
LED-large [3]	256	16384	2048	16*64	1024+1+0
Llama-2-7B [44]	128	4096	128	32*128	2044+4+0
Pythia-12B [4]	1024	2048	256	40*128	1022+2+0
Mistral-7B [18]	128	8192	2048	32*128	4088+8+0



(a) Sparse attention performance for different models in prefill stage.



(b) Attention performance for different models in decoding stage.

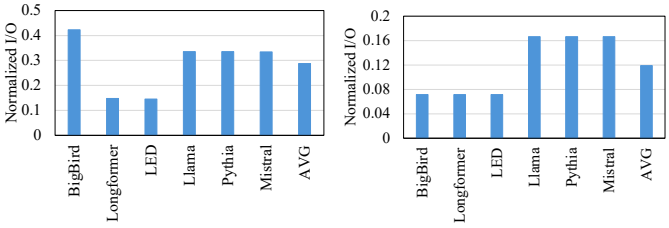
Fig. 9: Normalized speedup of GPU, PIM, and PILOT.

Benchmarks. We evaluate several models utilizing hybrid sparse attention patterns in the prefill stage, including BigBird [55], Longformer [3], and its variant LED-large [3], all specifically designed for long-sequence NLP tasks. We also evaluate DuoAttention [49], a framework that selectively applies hybrid sparse attention to specific attention heads in both prefill and decoding stages. Specifically, we implement DuoAttention on Llama-2-7B [44], Pythia-12B [4], and Mistral-7B [18], configuring 50% of attention heads to use sparse patterns. All evaluated models are executed using FP16 precision under a multi-batch inference setting. Detailed configurations are summarized in Table II, where the hidden size represents the total model dimension as $num\_heads \times head\_dim$, and the sparse pattern denotes the number of tokens used in each model, including the sliding-window size, global tokens, and random tokens (e.g., $192 + 64 + 64$).

Please note that PILOT does not modify the attention mechanism, sparsity pattern, model parameters, or numerical computation; it only optimizes data movement. Therefore, all evaluated models preserve their original accuracy.

B. Performance of PILOT

Attention Performance. Figure 9 compares the normalized attention speedup of PILOT, PIM, and GPU across different long-sequence models during both prefill and decoding stages. Speedups are normalized to the GPU baseline. We highlight three major observations. First, in the prefill stage, which executes sparse attention with hybrid sparsity patterns, PILOT consistently outperforms all baselines across all models, as shown in Figure 9(a). On average, PILOT achieves 9.93× and 4.20× speedups over GPU and PIM, respectively. Specifically, PILOT outperforms GPU by up to 19.96× (LED) and outperforms PIM by up to 6.52× (LED). By executing sparse attention during prefill and minimizing internal I/O, PILOT effectively mitigates both external and internal I/O overhead, significantly improving performance over existing solutions. Second, although attention with hybrid sparse patterns in the prefill stage is memory-bound, directly executing it on PIM limits the potential benefits of PIM. Without internal I/O optimizations, PIM provides only an average speedup of 2.36× over the GPU baseline. Third, as shown in Figure 9(b), during the decoding stage, PILOT achieves superior performance across models using dense attention as well as those employing sparse attention under the DuoAttention framework. On average, PILOT delivers 34.59× and 8.94× speedups over GPU and PIM, respectively. These results demonstrate that optimizing I/O efficiency yields greater performance gains for attention computations in the decoding stage, which inherently have low arithmetic intensity.



(a) Prefill stage (b) Decoding stage

Fig. 10: Normalized internal I/O of PILOT compared to PIM.

Internal I/O. To evaluate the effectiveness of the I/O-aware tiling and scheduling strategies of PILOT, we measure the volume of data movement (in bytes) between the local buffer of each BPM and the corresponding bank memory (internal I/O). Figure 10 presents the internal I/O of PILOT, normalized to that of PIM, across different models during attention. As shown in Figure 10(a), PILOT substantially reduces internal I/O during the prefill stage for all evaluated models, achieving on average approximately 28.71% of the internal I/O of PIM. These results demonstrate the effectiveness of the I/O-aware tiling in fully utilizing the limited buffer capacity, and the scheduling strategy in maximizing input reuse across hybrid sparse patterns. Moreover, Figure 10(b) shows that PILOT consistently achieves notable internal I/O reduction in the decoding stage as well, averaging around 11.90% of the internal I/O of PIM. These results highlight that internal I/O optimizations are essential for a PIM system, as demonstrated by PILOT, which effectively enables I/O-optimized, hardware-friendly strategies specifically tailored to its architecture.

C. Scalability of PILOT

We analyze the scalability of PILOT by showing the sparse attention speedups across varying input sequence lengths (L_{in})

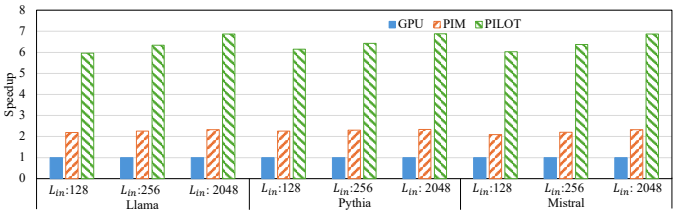


Fig. 11: Attention speedups in prefill across varying L_{in}.

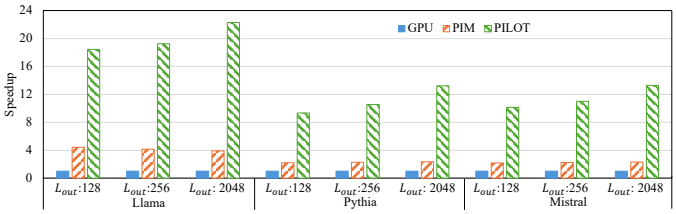


Fig. 12: Attention speedups in decoding across varying L_{out}.

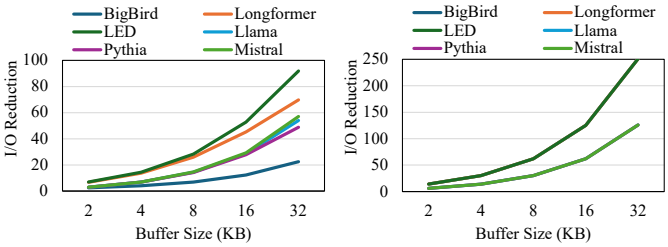
in the prefill stage, and varying output sequence lengths (L_{out}) in the decoding stage of the models with the DuoAttention framework. As illustrated in Figure 11 and Figure 12, the speedups achieved by PILOT consistently increase with longer sequence lengths. In the prefill stage, PILOT achieves up to 6.88× and 2.96× speedups over GPU and PIM, respectively, when scaling L_{in} to 2048. Similarly, in the decoding stage, PILOT achieves up to 22.30× and 5.71× speedups over GPU and PIM, respectively, when scaling L_{out} to 2048. These scalability results demonstrate the robust capability of PILOT in accelerating attention computations in both inference stages.

D. Performance with Different Buffer Sizes

To understand the generalization capability of the I/O-aware tiling and scheduling strategies, Figure 13 shows the internal I/O reduction achieved by PILOT compared to the PIM baseline when executing sparse attention in the prefill stage and attention in the decoding stage across varying buffer sizes. We make two key observations from Figure 13. First, PILOT consistently provides significant internal I/O reductions for all evaluated models. Second, these improvements become more substantial as the buffer size increases. When the buffer size reaches 32 KB, PILOT reduces internal I/O by up to 91.77× for sparse attention in prefill and up to 251.00× for attention in decoding. These results validate the importance of I/O-aware optimizations that carefully consider input dimensions and hardware constraints to fully utilize limited buffer capacities.

E. End-to-End Performance of LLM Inference

Figure 14 compares the normalized end-to-end inference execution time of GPU, AttAcc [34], the state-of-the-art GPU-PIM system for LLM inference, and GPU-PILOT across various models. We observe that GPU-PILOT outperforms GPU and AttAcc across all evaluated models. Moreover, the performance advantage of GPU-PILOT tends to increase for longer sequence lengths ($L_{in} + L_{out}$). Among models with



(a) Sparse attention in prefill (b) Attention in decoding

Fig. 13: Internal I/O reduction with varying buffer sizes.

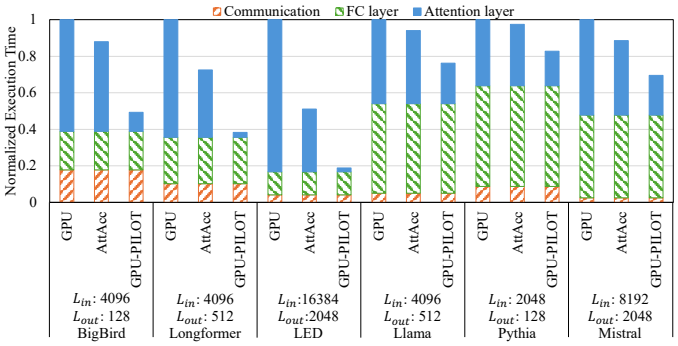


Fig. 14: Normalized execution time of various long-sequence models on GPU, AttAcc, and GPU-PILOT systems.

hybrid sparse attention patterns in the prefill stage, GPU-PILOT achieves speedups of up to 5.31× and 2.71× over GPU and AttAcc, respectively, for the LED model. Among models using DuoAttention, GPU-PILOT achieves speedups of up to 1.44× and 1.27× over GPU and AttAcc, respectively, for the Mistral model. The relatively smaller gap under DuoAttention arises because half of the attention heads in the prefill stage use dense attention and thus still execute on GPUs, limiting the acceleration opportunities for PILOT. Overall, compared to the state-of-the-art AttAcc, the performance improvements primarily arise from the efficient execution of hybrid sparse attention during the prefill stage and from the I/O-aware optimizations, which mitigate internal I/O overhead in both inference stages.

F. Overhead

The total area overhead of processing units integrated within an HBM3 stack for PILOT is 14.87 mm^2 per DRAM die, corresponding to 12.29% of a 121 mm^2 standard HBM3 [35], [39]. Specifically, each BPM introduces an area overhead of 0.107 mm^2 and consumes 4.10 mW of power. Additionally, each BGPM contributes an area overhead of 0.036 mm^2 and consumes 0.17 mW of power.

VI. Related Works

I/O analysis provides a theoretical foundation for analyzing data movement across memory hierarchies and offers insights into data reuse and dependency patterns. It has been successfully applied to optimizing FFT [17], matrix multiplica-

tion [20], CNN [5], and long-sequence dense attention [32]. In contrast to prior studies, we provide I/O analysis for long-sequence hybrid sparse attention and leverage it to guide optimized execution strategies in PIM architectures.

Recent research has explored a variety of architectures and co-design techniques to accelerate LLM inference. For example, A^3 [11], SpAtten [45], Sanger [31], ELSA [12], and DOTA [37] employ approximation or pruning strategies to reduce attention computation using dedicated hardware. However, these works primarily focus on reducing computational complexity, while the memory bottleneck remains a key limiting factor in long-sequence inference. Moreover, several studies have explored PIM techniques to accelerate transformer-based LLMs, including AttAcc [34] and NeuPIMs [15], which offload attention in the decoding stage to PIM; PIM-DL [22] and Pyramid [51], which replace GEMM computations with lookup-table-based operations to improve in-memory efficiency; and SpecPIM [23], which applies PIM to accelerate speculative inference. Unlike these studies, our work targets the memory bottleneck through I/O-aware PIM acceleration of attention mechanisms.

VII. Conclusion

In this paper, we propose PILOT, a PIM-based, I/O-aware software-hardware co-design for accelerating memory-bound attention computations during long-sequence LLM inference. At the software level, we conduct a comprehensive I/O analysis and propose optimized tiling and scheduling strategies explicitly tailored to hybrid sparse attention in the prefill stage and attention computations in the decoding stage. At the hardware level, PILOT strategically integrates computational units within the memory hierarchy. Extensive evaluations demonstrate that PILOT significantly improves performance and substantially reduces internal I/O overhead for memory-bound attention operations. Our detailed I/O analysis lays a strong foundation for future architectures designed to efficiently handle workloads with irregular data access patterns.

Acknowledgment

We thank the anonymous reviewers for their constructive feedback. This work was supported in part by the National Science Foundation under Grants CNS-2310422, CNS-2152497, and CNS-2310423.

References

[1] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat *et al.*, "Gpt-4 technical report," *arXiv preprint arXiv:2303.08774*, 2023.

[2] J. Ahn, S. Yoo, O. Mutlu, and K. Choi, "A scalable processing-in-memory accelerator for parallel graph processing," in *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2015, pp. 105–117.

[3] I. Beltagy, M. E. Peters, and A. Cohan, "Longformer: The long-document transformer," *arXiv preprint arXiv:2004.05150*, 2020.

[4] S. Biderman, H. Schoelkopf, Q. G. Anthony, H. Bradley, K. O'Brien, E. Hallahan, M. A. Khan, S. Purohit, U. S. Prashanth, E. Raff *et al.*, "Pythia: A suite for analyzing large language models across training and scaling," in *International Conference on Machine Learning*. PMLR, 2023, pp. 2397–2430.

[5] X. Chen, Y. Han, and Y. Wang, "Communication lower bound in convolution accelerators," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2020, pp. 529–541.

[6] R. Child, S. Gray, A. Radford, and I. Sutskever, "Generating long sequences with sparse transformers," *arXiv preprint arXiv:1904.10509*, 2019.

[7] 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.

[8] H. Duan, J. Wei, C. Wang, H. Liu, Y. Fang, S. Zhang, D. Lin, and K. Chen, "Botchat: Evaluating llms' capabilities of having multi-turn dialogues," *arXiv preprint arXiv:2310.13650*, 2023.

[9] Y. Gu, A. Khadem, S. Umesh, N. Liang, X. Servot, O. Mutlu, R. Iyer, and R. Das, "Pim is all you need: A cxl-enabled gpu-free system for large language model inference," in *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, 2025, pp. 862–881.

[10] D. Guo, Q. Zhu, D. Yang, Z. Xie, K. Dong, W. Zhang, G. Chen, X. Bi, Y. Wu, Y. Li *et al.*, "Deepseek-coder: When the large language model meets programming–the rise of code intelligence," *arXiv preprint arXiv:2401.14196*, 2024.

[11] T. J. Ham, S. J. Jung, S. Kim, Y. H. Oh, Y. Park, Y. Song, J.-H. Park, S. Lee, K. Park, J. W. Lee *et al.*, "A^3: Accelerating attention mechanisms in neural networks with approximation," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2020, pp. 328–341.

[12] T. J. Ham, Y. Lee, S. H. Seo, S. Kim, H. Choi, S. J. Jung, and J. W. Lee, "Elsa: Hardware-software co-design for efficient, lightweight self-attention mechanism in neural networks," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2021, pp. 692–705.

[13] Y. Hao, J. Gu, H. W. Wang, L. Li, Z. Yang, L. Wang, and Y. Cheng, "Can mllms reason in multimodality? emma: An enhanced multimodal reasoning benchmark," *arXiv preprint arXiv:2501.05444*, 2025.

[14] M. He, C. Song, I. Kim, C. Jeong, S. Kim, I. Park, M. Thottethodi, and T. Vijaykumar, "Newton: A dram-maker's accelerator-in-memory (aim) architecture for machine learning," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2020, pp. 372–385.

[15] G. Heo, S. Lee, J. Cho, H. Choi, S. Lee, H. Ham, G. Kim, D. Mahajan, and J. Park, "Neupims: Npu-pim heterogeneous acceleration for batched llm inferencing," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, 2024, pp. 722–737.

[16] L. Huang, S. Cao, N. Parulian, H. Ji, and L. Wang, "Efficient attentions for long document summarization," *arXiv preprint arXiv:2104.02112*, 2021.

[17] H. Jia-Wei and H.-T. Kung, "I/o complexity: The red-blue pebble game," in *Proceedings of the thirteenth annual ACM symposium on Theory of computing*, 1981, pp. 326–333.

[18] A. Q. Jiang, A. Sablayrolles, A. Mensch, C. Bamford, D. S. Chaplot, D. de Las Casas, F. Bressand, G. Lengyel, G. Lample, L. Saulnier, L. R. Lavaud, M. Lachaux, P. Stock, T. L. Scao, T. Lavril, T. Wang, T. Lacroix, and W. E. Sayed, "Mistral 7b," *CoRR*, vol. abs/2310.06825, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2310.06825

[19] H. Jiang, Y. Li, C. Zhang, Q. Wu, X. Luo, S. Ahn, Z. Han, A. H. Abdi, D. Li, C.-Y. Lin *et al.*, "Minference 1.0: Accelerating pre-filling for long-context llms via dynamic sparse attention," *Advances in Neural Information Processing Systems*, vol. 37, pp. 52 481–52 515, 2024.

[20] G. Kwasniewski, M. Kabić, M. Besta, J. VandeVondele, R. Solcà, and T. Hoefler, "Red-blue pebbling revisited: near optimal parallel matrix-matrix multiplication," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2019, pp. 1–22.

[21] S. Lee, S.-h. Kang, J. Lee, H. Kim, E. Lee, S. Seo, H. Yoon, S. Lee, K. Lim, H. Shin *et al.*, "Hardware architecture and software stack for pim based on commercial dram technology: Industrial product," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2021, pp. 43–56.

[22] C. Li, Z. Zhou, Y. Wang, F. Yang, T. Cao, M. Yang, Y. Liang, and G. Sun, "Pim-dl: Expanding the applicability of commodity dram-pims for deep learning via algorithm-system co-optimization," in *Proceedings of the 29th ACM International Conference on Architectural Support for*

Programming Languages and Operating Systems, Volume 2, 2024, pp. 879–896.

[23] C. Li, Z. Zhou, S. Zheng, J. Zhang, Y. Liang, and G. Sun, "Specpim: Accelerating speculative inference on pim-enabled system via architecture-dataflow co-exploration," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, 2024, pp. 950–965.

[24] S. Li, Z. Yang, D. Reddy, A. Srivastava, and B. Jacob, "Dramsim3: A cycle-accurate, thermal-capable dram simulator," *IEEE Computer Architecture Letters*, vol. 19, no. 2, pp. 106–109, 2020.

[25] S. Li, C. Hu, R. Langston, L. Deng, Y. Xie, J. Ahn, and O. Mutlu, "ipim: Programmable in-memory image processing accelerator using near-bank architecture," in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2018, pp. 12–23.

[26] Y. Li, Z. Chen, A. Arunkumar, and J. Ahn, "Fafnir: Accelerating sparse gathering by using efficient near-memory intelligent reduction," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2021, pp. 635–648.

[27] Y. Li, Z. Chen, O. Mutlu, and J. Ahn, "Chameleon: Versatile and practical near-dram acceleration architecture for large memory systems," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2020, pp. 1161–1175.

[28] J. Lin, H. Yin, W. Ping, P. Molchanov, M. Shoeybi, and S. Han, "Vila: On pre-training for visual language models," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2024, pp. 26 689–26 699.

[29] C. Liu, H. Liu, D. Chen, Y. Huang, Y. Zhang, W. Xiao, X. Liao, and H. Jin, "Heterrag: Heterogeneous processing-in-memory acceleration for retrieval-augmented generation," in *Proceedings of the 52nd Annual International Symposium on Computer Architecture*, 2025, pp. 884–898.

[30] A. Lozhkov, R. Li, L. B. Allal, F. Cassano, J. Lamy-Poirier, N. Tazi, A. Tang, D. Pykhtar, J. Liu, Y. Wei *et al.*, "Starcoder 2 and the stack v2: The next generation," *arXiv preprint arXiv:2402.19173*, 2024.

[31] L. Lu, Y. Jin, H. Bi, Z. Luo, P. Li, T. Wang, and Y. Liang, "Sanger: A co-design framework for enabling sparse attention using reconfigurable architecture," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, 2021, pp. 977–991.

[32] X. Lu, B. Long, X. Chen, Y. Han, and X.-H. Sun, "I/o analysis is all you need: An i/o analysis for long- sequence attention," in *Proceedings of the 31st ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, 2026.

[33] O. Mutlu, "Intelligent architectures for intelligent computing systems," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2021, pp. 318–323.

[34] J. Park, J. Choi, K. Kyung, M. J. Kim, Y. Kwon, N. S. Kim, and J. H. Ahn, "Attacc! unleashing the power of pim for batched transformer-based generative model inference," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, 2024, pp. 103–119.

[35] M.-J. Park, J. Lee, K. Cho, J. Park, J. Moon, S.-H. Lee, T.-K. Kim, S. Oh, S. Choi, Y. Choi *et al.*, "A 192-gb 12-high 896-gb/s hbm3 dram with a tsv auto-calibration scheme and machine-learning-based layout optimization," *IEEE Journal of Solid-State Circuits*, vol. 58, no. 1, pp. 256–269, 2022.

[36] P. Patel, E. Choukse, C. Zhang, A. Shah, Í. Goiri, S. Maleki, and R. Bianchini, "Splitwise: Efficient generative llm inference using phase splitting," in *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2024, pp. 118–132.

[37] Z. Qu, L. Liu, F. Tu, Z. Chen, Y. Ding, and Y. Xie, "Dota: detect and omit weak attentions for scalable transformer acceleration," in *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2022, pp. 14–26.

[38] B. Roziere, J. Gehring, F. Gloeckle, S. Sootla, I. Gat, X. E. Tan, Y. Adi, J. Liu, R. Sauvestre, T. Remez *et al.*, "Code llama: Open foundation models for code," *arXiv preprint arXiv:2308.12950*, 2023.

[39] Y. Ryu, S.-G. Ahn, J. H. Lee, J. Park, Y. K. Kim, H. Kim, Y. G. Song, H.-W. Cho, S. Cho, S. H. Song *et al.*, "A 16 gb 1024 gb/s hbm3 dram with source-synchronized bus design and on-die error control scheme for enhanced ras features," *IEEE Journal of Solid-State Circuits*, vol. 58, no. 4, pp. 1051–1061, 2023.

[40] J. R. Stevens, R. Venkatesan, S. Dai, B. Khailany, and A. Raghunathan, "Softermax: Hardware/software co-design of an efficient softmax for transformers," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2021, pp. 469–474.

[41] X.-H. Sun and X. Lu, "The memory-bounded speedup model and its impacts in computing," *Journal of Computer Science and Technology*, vol. 38, no. 1, pp. 64–79, 2023.

[42] X.-H. Sun and L. M. Ni, "Scalable problems and memory-bounded speedup," *Journal of parallel and distributed computing*, vol. 19, no. 1, pp. 27–37, 1993.

[43] H. Tang, Y. Lin, J. Lin, Q. Han, S. Hong, Y. Yao, and G. Wang, "Razorattention: Efficient kv cache compression through retrieval heads," *arXiv preprint arXiv:2407.15891*, 2024.

[44] 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.

[45] H. Wang, Z. Zhang, and S. Han, "Spatten: Efficient sparse attention architecture with cascade token and head pruning," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2021, pp. 97–110.

[46] S. Williams, A. Waterman, and D. Patterson, "Roofline: an insightful visual performance model for multicore architectures," *Communications of the ACM*, vol. 52, no. 4, pp. 65–76, 2009.

[47] W. Wu, Y. Wang, G. Xiao, H. Peng, and Y. Fu, "Retrieval head mechanistically explains long-context factuality," *arXiv preprint arXiv:2404.15574*, 2024.

[48] Y. Wu, Z. Wang, and W. D. Lu, "Pim gpt a hybrid process in memory accelerator for autoregressive transformers," *npj Unconventional Computing*, vol. 1, no. 1, p. 4, 2024.

[49] G. Xiao, J. Tang, J. Zuo, J. Guo, S. Yang, H. Tang, Y. Fu, and S. Han, "Duoattention: Efficient long-context llm inference with retrieval and streaming heads," *arXiv preprint arXiv:2410.10819*, 2024.

[50] G. Xiao, Y. Tian, B. Chen, S. Han, and M. Lewis, "Efficient streaming language models with attention sinks," *arXiv preprint arXiv:2309.17453*, 2023.

[51] L. Yan, X. Lu, X. Chen, Y. Han, and X.-H. Sun, "Pyramid: Accelerating llm inference with cross-level processing-in-memory," *IEEE Computer Architecture Letters*, 2025.

[52] L. Yan, M. Zhang, R. Wang, X. Chen, X. Zou, X. Lu, Y. Han, and X.-H. Sun, "Copim: a concurrency-aware pim workload offloading architecture for graph applications," in *2021 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*. IEEE, 2021, pp. 1–6.

[53] S. Yang, J. Guo, H. Tang, Q. Hu, G. Xiao, J. Tang, Y. Lin, Z. Liu, Y. Lu, and S. Han, "Lserve: Efficient long-sequence llm serving with unified sparse attention," *arXiv preprint arXiv:2502.14866*, 2025.

[54] Z. Yuan, Y. Shang, Y. Zhou, Z. Dong, Z. Zhou, C. Xue, B. Wu, Z. Li, Q. Gu, Y. J. Lee *et al.*, "Llm inference unveiled: Survey and roofline model insights," *arXiv preprint arXiv:2402.16363*, 2024.

[55] M. Zaheer, G. Guruganesh, K. A. Dubey, J. Ainslie, C. Alberti, S. Ontanon, P. Pham, A. Ravula, Q. Wang, L. Yang *et al.*, "Big bird: Transformers for longer sequences," *Advances in neural information processing systems*, vol. 33, pp. 17 283–17 297, 2020.

[56] T. Zhang, F. Ladhak, E. Durmus, P. Liang, K. McKeown, and T. B. Hashimoto, "Benchmarking large language models for news summarization," *Transactions of the Association for Computational Linguistics*, vol. 12, pp. 39–57, 2024.

[57] J. Zhao, P. Zeng, G. Shen, Q. Chen, and M. Guo, "Hardware–software co-design enabling static and dynamic sparse attention mechanisms," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 43, no. 9, pp. 2783–2796, 2024.

[58] M. Zhou, W. Xu, J. Kang, and T. Rosing, "Transpim: A memory-based acceleration via software-hardware co-design for transformer," in *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2022, pp. 1071–1085.

[59] A. Zou, W. Yu, H. Zhang, K. Ma, D. Cai, Z. Zhang, H. Zhao, and D. Yu, "Docbench: A benchmark for evaluating llm-based document reading systems," *arXiv preprint arXiv:2407.10701*, 2024.

Appendix: Artifact Description/Artifact Evaluation

Artifact Description (AD)

I. Overview of Contributions and Artifacts

A. Paper's Main Contributions

C_1 We identify that employing sparse patterns in the prefill stage shifts attention computations from being compute-bound to memory-bound, imposing significant performance constraints on GPUs.

C_2 Beyond supporting attention during decoding, we explore the potential of utilizing PIM to efficiently execute static hybrid sparse attention patterns during the prefill stage, thereby providing a comprehensive solution for mitigating costly external I/O overhead.

C_3 We conduct a detailed I/O analysis of memory-bound attention across both inference stages and propose I/O-aware tiling and scheduling strategies to minimize internal I/O overhead within the PIM architecture.

C_4 We propose a GPU-PILOT heterogeneous system architecture, providing a comprehensive end-to-end solution that addresses both external and internal I/O bottlenecks for accelerating long-sequence LLM inference.

B. Computational Artifacts

A_1 https://doi.org/10.5281/zenodo.18645044

Artifact ID	Contributions Supported	Related Paper Elements
A_1	C_1, C_2, C_3, C_4	Tables 1 Figures 9-14

II. Artifact Identification

A. Computational Artifact A_1

Relation To Contributions

This artifact is used to reproduce the Artifact A1 supports the evaluation of all key contributions of the paper:

- C_1: Characterization of memory-bound behavior of sparse attention
- C_2: PIM for hybrid sparse attention execution
- C_3: I/O-aware optimization strategies
- C_4: End-to-end GPU-PILOT heterogeneous framework

Expected Results

By running the provided scripts, users should be able to reproduce:

- Sparse attention speedups during prefill stage
- Attention acceleration during decoding stage
- Internal I/O reduction trends under different buffer sizes
- End-to-end inference performance comparison between GPU, AttAcc, and GPU-PILOT systems

The reproduced results should follow the same performance trends reported in the paper.

Expected Reproduction Time (in Minutes)

Artifact Setup (incl. Inputs)

Hardware: Two AMD EPYC 7763 CPUs (64 Cores, 128 Threads) @2.45 GHz, 2 TB RAM, four HBM3 stacks operating at a data rate of 5.2 Gbps per pin and NVIDIA A100 GPU

Software: Ubuntu 22.04.5 LTS

Datasets / Inputs: enwik-8 https://mattmahoney.net/dc/textdata.html

Installation and Deployment: CUDA 12.8, Python, and PyTorch 1.13.0

Artifact Execution

Run single experiment: python pim_simulator.py
Run full evaluation results: bash run_all_exps.sh

Artifact Analysis (incl. Outputs)

The simulator generates:

- Execution latency of different implementations for both prefill and decoding stages
- Memory traffic statistics
- Internal I/O metrics