LibraPIM: Dynamic Load Rebalancing to Maximize Utilization in PIM-Assisted LLM Inference Systems

Hyeongjun Cho
Sungkyunkwan University
Suwon, Korea
cho0624ck@skku.edu

Yoonho Jang
Sungkyunkwan University
Suwon, Korea
jyh8807@skku.edu

Hyungi Kim
Sungkyunkwan University
Suwon, Korea
khg718@skku.edu

Seongwook Kim
Sungkyunkwan University
Suwon, Korea
su8939@skku.edu

Keewon Kwon
Sungkyunkwan University
Suwon, Korea
keewkwon@skku.edu

Gwangsun Kim
POSTECH
Suwon, Korea
g.kim@postech.ac.kr

Seokin Hong
Sungkyunkwan University
Suwon, Korea
seokin@skku.edu

Abstract

Large language models (LLMs) require inference systems that can handle both compute- and memory-intensive workloads. GPUs and NPUs (referred to as xPUs) efficiently process compute-intensive layers, while Processing-In-Memory (PIM) architectures are well suited for memory-bound stages. To exploit this complementary relationship, recent LLM inference systems have adopted heterogeneous architectures integrating xPUs with PIM units. However, this integration poses several challenges. Tight execution dependencies between the two devices limit concurrency, as PIM often must wait for data produced by the xPUs. Moreover, batch size and sequence length affect the computational load on PIM and the accelerator differently, leading to execution imbalance across devices.

To address these challenges, we propose **LibraPIM**, a novel PIM framework that orchestrates workload rebalancing and concurrent execution between compute accelerators and in-memory compute units, enabling efficient and scalable LLM inference. LibraPIM addresses the above challenges through two key techniques: **Dynamic Batch Offloading (DBO)**, which adaptively redirects portions of the workload to the underutilized device based on runtime profiling, and **Dual-Path Execution (DEX)**, which enables concurrent PIM operations and memory accesses through sub-bank partitioning with minimal hardware overhead. Together, these techniques improve resource utilization across heterogeneous devices, thereby increasing system throughput in LLM inference. Our experimental results demonstrate that LibraPIM achieves **6.2×** average speedup over the baseline PIM-enabled system and delivers a **2.1×** average speedup compared to a state-of-the-art approach.

Keywords

LLM inference, Processing-In-Memory, Load balancing

1 Introduction

Transformer-based large language models (LLMs) [5, 15, 52, 53, 59] have been widely adopted across various applications, including natural language processing [25, 55] and other multimodal tasks [39, 51, 54]. LLM inference typically relies on compute accelerators, such as GPUs [47, 48] and NPUs [8, 18, 19, 26, 32, 45], which exploit massive parallelism to accelerate computationally intensive layers. To improve accelerator utilization and overall throughput, LLM inference systems often employ a batching scheme [9, 16, 57], where multiple requests are aggregated into a single matrix–matrix multiplication (GEMM). This approach enables efficient processing of QKV generation, output projection, and feed-forward layers.

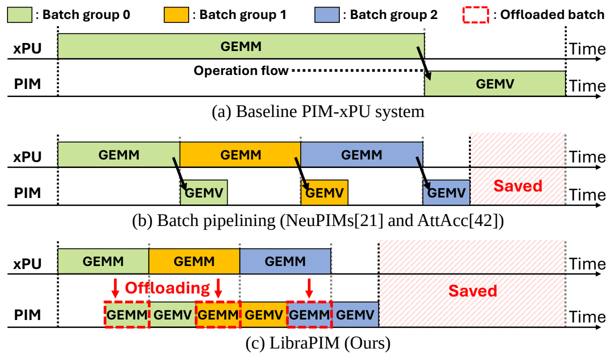


Figure 1: Execution timeline of LLM inference in (a) baseline, (b) batch-pipelined, and (c) LibraPIM-enabled systems. LibraPIM reduces idle time through dynamic task offloading and concurrent execution.

Unlike other transformer operations, however, the multi-head attention (MHA) layer cannot benefit from the batching, as each token generates its own query, key, and value vectors. This prevents aggregation across tokens, leaving the computation in the form of matrix–vector multiplications (GEMVs), which are inherently memory-bound [1, 41, 44, 60]. As a result, MHA becomes a persistent performance bottleneck in LLM inference, posing a major challenge in efficiently accelerating transformer workloads.

A promising direction to address this challenge is the use of Processing-In-Memory (PIM) devices [2, 3, 12, 13, 27–29, 38, 46, 50, 56]. Recent DRAM-based PIM architectures integrate lightweight processing units within each memory bank, exploiting high internal bandwidth to perform GEMV [20, 36, 61]. While PIM effectively addresses memory-bound operations, it lacks the computing capability to perform GEMM. In contrast, accelerators such as GPUs and NPUs (xPUs) offer high arithmetic throughput but remain limited by memory-bound operations. To exploit these complementary strengths, modern inference systems combine PIM and xPUs into a heterogeneous architecture [21, 44, 49]: GEMV layers (e.g., MHA) execute on PIM, while GEMM layers run on xPUs, achieving high compute throughput while alleviating memory bottlenecks.

Although heterogeneous systems that integrate PIM and xPUs offer theoretical benefits, they often fail to fully utilize both devices in practice, primarily due to two fundamental limitations: static workload allocation and serial execution dependencies across layers. Figure 1a illustrates static workload allocation in a conventional PIM-xPU heterogeneous systems, where GEMM operations execute entirely on the xPUs before GEMV can begin on the PIM. This strict sequencing enforces device serialization: one device remains idle while waiting for the other to finish, leading to underutilization and execution stalls.

To mitigate this inefficiency, recent work [21, 44] proposes batch pipelining, which overlaps the execution of independent batch groups to improve concurrency [9, 16, 57]. As shown in Figure 1b, this technique assigns different batch groups to each device in a staggered manner, allowing partial overlap between GEMM and GEMV executions. While this approach increases parallelism between compute- and memory-bound operations, *it remains vulnerable to load imbalance between PIM and xPUs, as it still relies on static workload allocation.* As illustrated, if GEMM takes longer to execute than GEMV (or vice versa), one device finishes early and stays idle.

This paper addresses the problem of imbalanced device utilization in heterogeneous PIM-xPU architectures for LLM inference. To tackle this challenge, we propose **LibraPIM**, a novel PIM framework that dynamically rebalances workloads between xPUs and in-memory compute units. LibraPIM achieves this goal through two key mechanisms: **Dynamic Batch Offloading (DBO)** and **Dual-Path EXecution (DEX)**.

The ***DBO*** mechanism adaptively redistributes workloads between PIM and xPUs to balance execution latencies. Unlike prior approaches that statically assign GEMM to xPUs and GEMV to PIM, *DBO* enables flexible task-to-device mapping, allowing PIM to execute a portion of GEMM operations and xPUs to handle GEMV when beneficial. To this end, *DBO* employs a runtime scheduler that estimates the execution time of each device based on model-specific parameters such as batch size, sequence length, and hidden dimension size. The scheduler identifies the device with longer projected latency and selectively offloads part of its workload to the other device. For instance, when xPUs are projected to take longer for GEMM, a portion of GEMM can be reassigned to PIM, as shown in Figure 1c. Conversely, if GEMV execution on PIM incurs longer latency, it can be offloaded to xPUs.

The ***DEX*** mechanism enables concurrent execution of PIM and xPU operations by decoupling their DRAM access paths. In a conventional PIM architectures, memory requests from the xPU can block PIM execution due to contention for the same DRAM bank. *DEX* resolves this by introducing a lightweight modification that partitions each bank into two sub-banks using slicing transistors. This structural separation allows normal memory accesses and PIM

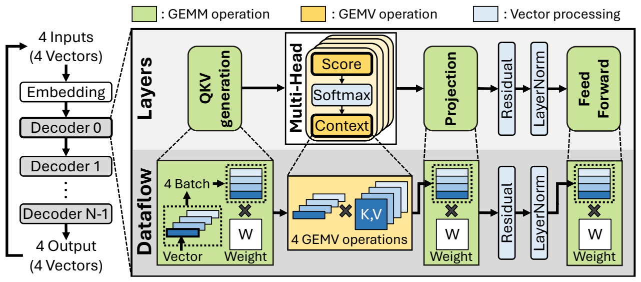


Figure 2: Computation flow of transformer-based LLM inference, illustrating how GEMM, GEMV, and vector processing operations are distributed across the decoder layer.

operations to proceed in parallel within the same bank. By overlapping xPU memory access with PIM computation, *DEX* eliminates inter-device interference and accelerates batch offloading, leading to improved system throughput without significant hardware changes.

Our evaluation demonstrates that LibraPIM effectively improves device utilization via its two key mechanisms. At a sequence length of 4096, LibraPIM achieves 38% PIM utilization and 77% NPU utilization, which are significantly higher than those of the state-of-the-art architectures (6% and 13% in AttAcc [44], and 21% and 62% in NeuPIMs [21]). This utilization improvement translates into an average of 6.2× speedup over a conventional PIM-xNPU architecture. LibraPIM also outperforms state-of-the-art architectures such as AttAcc and NeuPIMs by 4.4× and 2.1× on average, respectively.

The key contributions of this paper are summarized as follows:

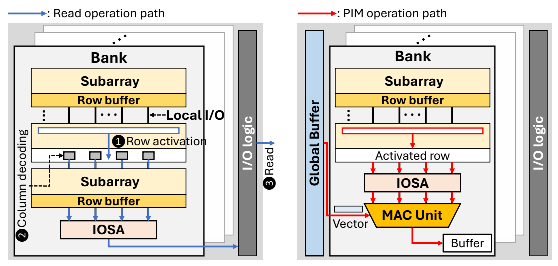
- We analyze the causes of device utilization imbalance in PIM–xPU architectures for LLM inference. In particular, we characterize the impact of batch size and sequence length on the imbalance and expose the limitations of batch pipelining.
- We propose **LibraPIM**, a novel framework that dynamically balances workloads between PIM and xPUs. LibraPIM introduces *Dynamic Batch Offloading*, a scheduling mechanism that adaptively offloads portions of GEMM and GEMV operations between devices based on predicted execution latency.
- LibraPIM employs *Dual-Path Execution* mechanism, a lightweight DRAM architectural enhancement that partitions each bank into sub-banks to physically decouple conventional memory accesses from PIM operations, enabling true concurrent execution with minimal hardware modifications.

2 Background

2.1 Batched LLM Inference

Figure 2 illustrates the token generation process in a transformer-based LLM. During the token generation, the transformer decoder takes the embedded input token as a vector and iteratively generates output tokens through its decoders. The output token is then fed back into the decoder to produce the next token.

The decoder consists mainly of memory-intensive GEMV (General Matrix–Vector multiplication) operations, which impose significant pressure on memory bandwidth [27]. To alleviate this, recent LLM inference systems often employ batched inference approach



(a) DRAM organization & Read operation (b) PIM architecture

Figure 3: DRAM organization and its read operation flow (left) and its extension to a PIM architecture (right).

that processes multiple input vectors simultaneously [9, 16, 57]. With batching, components such as Query, Key, and Value generation (QKV), projection, and the feed-forward network (FFN) share weights across input vectors, converting GEMV into GEMM (General Matrix–Matrix multiplication), which can be efficiently executed on GPUs or NPUs (collectively referred to as xPUs).

However, as shown in Figure 2, multi-head attention (MHA) remains inherently memory-intensive. This is because, unlike other layers, MHA cannot share query, key, and value vectors across input tokens, as each token generates its own unique set for self-attention computations (yellow box in Figure 2). As a result, each input vector requires a separate GEMV operation, leading to sequential processing. This increases memory bandwidth demands and creates a significant bottleneck in the LLM inference.

2.2 DRAM-based PIM Architectures

Processing-In-Memory (PIM) technology provides an effective solution for handling memory-intensive tasks [2, 50, 56]. To provide architectural context, Figure 3 illustrates a standard DRAM organization and its read operation (left), alongside a representative PIM design, AiM [20], implemented in an HBM2 configuration [24] (right). The figure shows how data is read from a conventional DRAM bank and how PIM extends this pathway to support near-data computation using local compute units.

A DRAM die consists of multiple banks that can operate independently. Each bank is further divided into multiple subarrays, which are structured as two-dimensional memory cell arrays. Each subarray contains a row buffer that amplifies data for reliable read or write. During a read operation, the bank first selects the subarray containing the requested data and activates the corresponding row (1KB size of data), loading it into the row buffer (❶). Next, the row buffer transfers data (256-bit), based on the specified column address, through the Local I/O path to the IOSA (❷). Finally, the IOSA amplifies the received data and transfers it to the I/O logic, which is connected to the external data bus (❸).

Recent PIM architectures integrate Multiply-and-Accumulate (MAC) units directly within memory banks [10, 20, 36], as shown in Figure 3b. This implementation reads data from the subarray like a normal read operation, enabling rapid in-memory computation by utilizing bank-level parallelism. While the PIM device can efficiently execute memory-intensive operations such as GEMV, it faces challenges in accelerating compute-intensive workloads due to the limited number of MAC units and lower internal memory clock speeds. To address this challenge, recent research has focused on using PIM devices alongside xPUs in LLM inference systems.

These heterogeneous PIM-xPU systems aim to leverage the complementary strengths of both processing components for improving overall system performance in LLM inference.

2.3 Limitation of Heterogeneous PIM-xPU Systems: Device Underutilization

The heterogeneous PIM-xPU systems accelerate LLM inference by distributing workloads according to their computational characteristics. In Figure 2, compute-intensive operations (green blocks) are executed on processing units (xPUs), while memory-intensive operations (yellow blocks) are executed on PIM devices. Figure 4 illustrates the overview of device-level operations and data flows. In (a), xPUs read weight matrices from DRAM equipped with PIM devices and perform MAC operations on input vectors for the GEMM stage. In (b), PIM devices receive the Q, K, and V vectors produced by the xPUs and perform the GEMV operation between the query and key/value matrices.

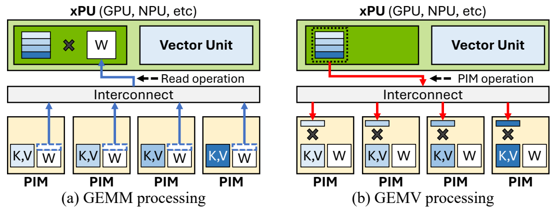
While the workload partitioning across computing devices leverages each device's strengths in a heterogeneous system, conventional heterogeneous systems still struggle to fully utilize all devices during LLM inference due to serial dependencies. As shown in Figure 5a, each device begins its assigned workload only after the other completes its task. This results in idle periods for one device while the other is active, reducing overall system utilization.

2.4 Recent Attempt: Batch Pipelining

To address this limitation, prior works introduce a batch pipelining technique that enables xPUs and PIM devices to execute concurrently by overlapping computation across independent batch groups [21, 44]. As illustrated in Figure 5b, the entire input batch is divided into smaller batch groups (e.g., A and B), which are processed in a staggered manner. While xPUs handle the QKV and FFN computations for group B, the MHA layer for group A is simultaneously executed on the PIM. Once xPUs complete group B and move to group A, the PIM begins processing MHA for group B. This interleaved execution reduces idle time for both devices by decoupling their execution timelines, thereby improving resource utilization compared to the serialized execution shown in Figure 5a.

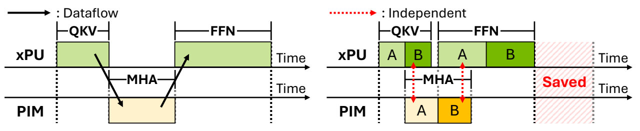
3 Motivation

This section identifies two challenges in the heterogeneous PIM–xPU systems with batch pipelining. First, variations in input parameters create execution imbalances between PIM and xPU devices, resulting in severe resource underutilization. Second, existing batch pipelining architectures, particularly those requiring



(a) GEMM processing (b) GEMV processing

Figure 4: Execution flow of LLM inference in a heterogeneous PIM-xPU system, where (a) GEMM operations are processed by xPUs and (b) GEMV operations are processed by PIM units.



(a) Conventional heterogeneous system (b) Concept of batch pipelining

Figure 5: Execution timelines of LLM inference on heterogeneous systems. (a) shows serialized execution due to inter-device dependency; (b) shows batch pipelining with partial overlap across batch groups A and B.

DRAM modifications, introduce significant hardware overhead. We analyze this overhead and its impact on system performance.

3.1 Unbalanced Execution Time

Workload Scaling with Batch Size and Sequence Length: Table 1 summarizes the time complexity of key operations of the transformer-based LLM model as a function of batch size (B) and sequence length (L). During inference, the batch size (B) and sequence length (L) are dynamically determined by the input. In contrast, model-specific parameters such as the hidden dimension (d_{model}) and the feed-forward dimension (d_{ff}) are fixed by the model configuration. The computational cost of xPU workloads, including QKV generation, projection, and FFN, scales linearly with both B and L. In comparison, the computation required for the MHA layer scales quadratically with L, making it the dominant workload at longer sequence lengths. This difference is clearly illustrated in Figure 6, which shows that the FLOPS associated with MHA (executed on PIM) increase much more rapidly than those of xPU-side operations. Furthermore, during generation, MHA becomes even more expensive as the number of output tokens (n) increases, since new K and V vectors must be generated for each token.

Static Allocation: In heterogeneous systems, it allocates workloads to specific hardware units based on their computational strength. While this strategy is effective in theory, it falls short in practice due to real-world LLM inference workloads' dynamic and diverse nature. Variations in batch size and sequence length continuously shift the ratio of workload demands about xPUs and PIM, leading to imbalanced load distribution. Therefore, one device may remain idle for extended periods while the other becomes a bottleneck, significantly degrading system efficiency. Figure 7 illustrates the idle time for each device in an NPU+PIM system (see Section 6.1 for experimental details). PIM exhibits significantly extended idle periods for small batches and short sequences, more than 80% of total processing time. Conversely, under large batches and extended sequences, NPU actives only 10–20% of the total processing time, while PIM is fully operated. This imbalance reduces overall system utilization during LLM inference.

Table 1: Time complexity of LLM layers.

Layer	Summarization	Generation
QKV Gen.	$O(B \cdot L \cdot {d_{model}}^2)$	$O(B \cdot {d_{model}}^2)$
MHA	$O(B \cdot L^2 \cdot d_{model})$	$O(B \cdot (L+n) \cdot d_{model})$
Projection	$O(B \cdot L \cdot {d_{model}}^2)$	$O(B \cdot {d_{model}}^2)$
FFN	$O(B \cdot L \cdot d_{model} \cdot d_{ff})$	$O(B \cdot d_{model} \cdot d_{ff})$

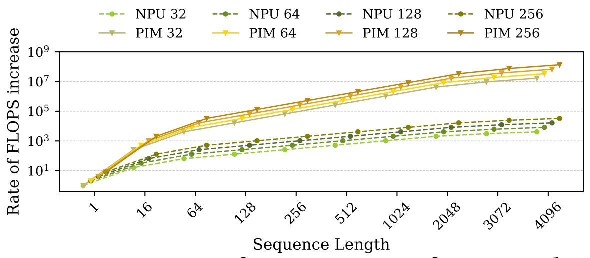


Figure 6: Comparison of FLOPS increase for PIM and NPU workloads with increasing sequence length. MHA on PIM scales more rapidly than NPU-side operations, leading to workload imbalance at longer sequence lengths.

> **Limitation ①: During LLM inference, heterogeneous PIM-xPU systems experience significant underutilization of computing resources due to variations in batch size and sequence length.**

3.2 Architectural Overhead in Batch Pipelining

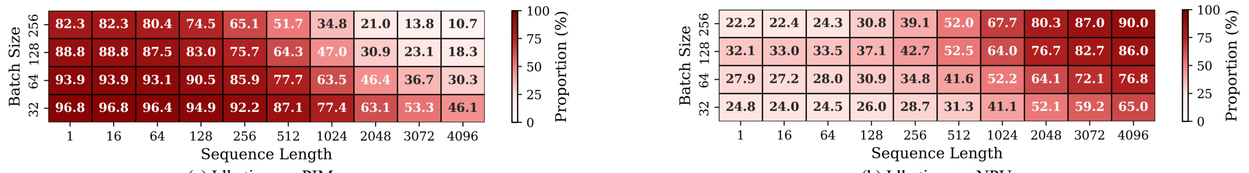
Although several prior works attempt to adopt batch pipelining, they face fundamental limitations due to hardware conflict and costly architectural modifications. In the ideal case (Figure 8a), DRAM read and PIM operation for different batch groups are executed in parallel, maximizing resource overlap and device utilization. However, when (Figure 8b), current PIM architectures enforce exclusive access to shared resources such as the row buffer and Local I/O lines. As a result, either the system performs PIM or serves read requests, but not both in parallel. This frequent resource conflict as seen in the Figure 8b limits the efficiency of batch pipelining.

Several prior works have proposed batch pipelining, but these often rely on substantial hardware modifications. For example, AttAcc integrates hierarchical MAC units at bank and bank-group levels to accelerate GEMV processing. However, it still suffers from limited parallelism, as these units contend for shared resources during DRAM read operations. NeuPIMs attempts to overcome this by doubling the row buffer, enabling DRAM reads and PIM operations to proceed concurrently [21, 44]. Despite these efforts, such modifications come with a high cost; doubling the row buffer increases DRAM mat area by up to 38% [7, 43] and adds considerable routing complexity across multiple metal layers. These approaches ultimately sacrifice practicality and scalability, highlighting the need for a lightweight solution that achieves meaningful overlap without major architectural redesigns.

> **Limitation ②: Achieving effective batch pipelining requires either substantial hardware modifications or exclusive access scheduling between DRAM and PIM operations.**

3.3 Underutilization in Batch-Pipelining Architecture

Even with batch pipelining, heterogeneous systems experience residual underutilization due to delays in consecutive operations, especially when PIM execution is involved. When DRAM accesses, and PIM operations are restricted to exclusive execution, PIM utilization declines significantly. Figure 9 shows the utilization of both the PIM and NPU under ideal pipelining (dotted box) and exclusive execution (yellow box) across varying sequence lengths for small (B=32) and large (B=256) batch sizes. In small batch scenarios,



(a) Idle time on PIM (b) Idle time on NPU

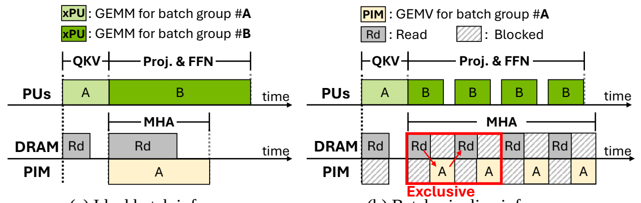
Figure 7: Proportion of idle time for PIM and NPU under varying batch sizes and sequence lengths during GPT3 175B inference.

DRAM read operations occupy a larger proportion of the workload, making the system more sensitive to serialization and reducing utilization. As sequence length increases, the MHA layer becomes the dominant component, leading to a sharp drop in PIM utilization, particularly when the sequence length exceeds 512, due to its high computational and bandwidth demands.

4 LibraPIM

4.1 Overview

This section introduces LibraPIM, a novel PIM architecture and system designed to reduce idle time by dynamically balancing workloads between xPUs and PIM devices in batch-pipelined LLM inference. Figure 10 provides an overview of **D**ynamic **B**atch **O**ffloading **(DBO)** in LibraPIM system. At the host level, LibraPIM includes a novel scheduler that identifies the dominant workload and adaptively determines the offloaded batch size to balance processing time. The scheduler estimates workload latency based on batch size and sequence length, referring to a table updated after each generation stage to reflect runtime variations. This dynamic adjustment ensures consistently balanced utilization across devices. In addition, LibraPIM is built on a flexible framework that supports various LLM models, providing broad applicability and scalability.


xPU: GEMM for batch group #A
xPU: GEMM for batch group #B
QKV
Proj. & FFN
PUs
A
B
time
MHA
DRAM
Rd
PIM
A
PIM: GEMV for batch group #A
Rd: Read
Blocked
Exclusive

(a) Ideal batch inference (b) Batch-pipeline inference

Figure 8: Exclusive execution of batch pipeline inference.

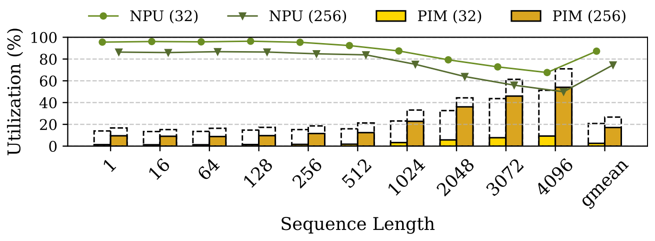


Figure 9: NPU & PIM utilization with batch pipelining. The number in brackets indicates the batch size.

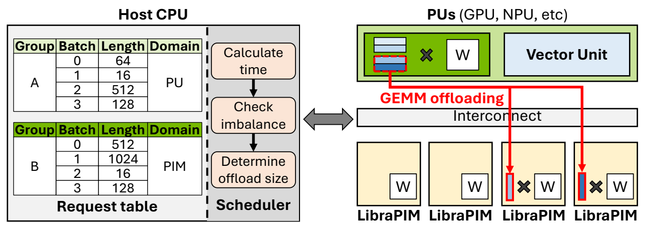


Figure 10: Overview of LibraPIM system.

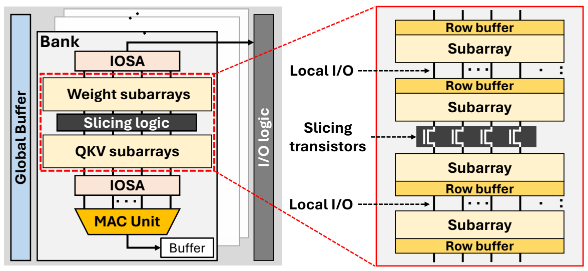


Figure 11: LibraPIM architecture.

4.2 Bank-Sliced PIM Architecture

To avoid the delay caused by exclusive execution when offloading during PIM operation, we modified the bank structure slightly. Figure 11 illustrates the LibraPIM bank architecture, which enables parallel PIM and read operations by partitioning each bank into two independent sections. As shown in the figure, the Local I/O path is divided into two parts using slicing logic, which consists of additional transistors and one-per-bit, effectively isolating the Local I/O path when enabled. This design allows each partition to read data independently without resource conflicts. We refer to this mechanism as **D**ual-Path **EX**ecution (DEX), and further discuss the impact of slicing logic on the signal-level in Section 5.

Following the division of the bank, each partition is assigned a different workload during LLM inference. One partition performs a standard read operation to transfer weight data for xPUs, while the other processes GEMV operations for the MHA layer via PIM operations. Consequently, one partition stores weight parameters in its subarrays, while the other stores K, V data. We incorporate an additional IOSA, the final amplifier in the DRAM hierarchy, to ensure reliable data reads across both partitions. Since IOSA is significantly smaller than the row buffer (256-bit for IOSA vs. 1KB per row buffer), it introduces only a 0.002% area overhead per bank, as calculated in our evaluation. Moreover, unlike other logic, all banks share the I/O logic and a global buffer, which supplies vector data to the banks for PIM operations. The global buffer can receive vectors from within the bank or I/O logic.

4.3 Operation of Dual-Path Execution

Intra-bank: To facilitate the understanding of DEX, we illustrate its detailed operation within a LibraPIM bank in Figure 12. First, the slicing logic divides the bank into two partitions to prevent resource conflicts. *Each partition independently selects a subarray and activates a row according to its specific task*—one activates a weight row for xPU access, while the other activates a K,V row for

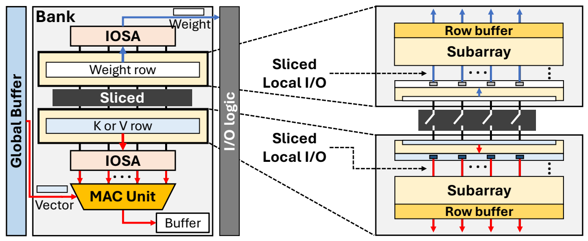


Figure 12: Detail of DEX in LibraPIM bank.

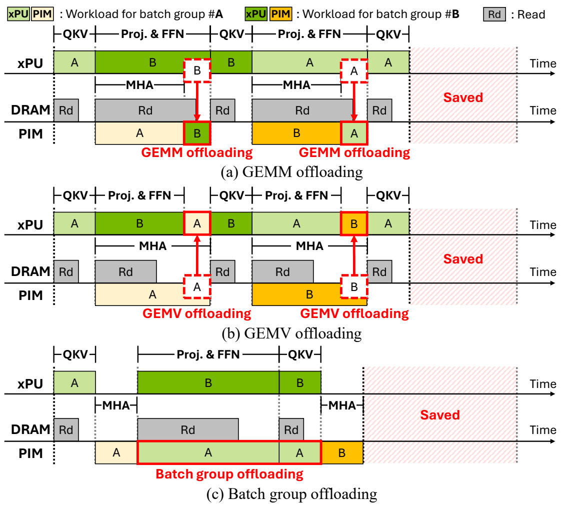


Figure 13: Concept of DBO with timing diagrams. The red box represents offloading a partial batch.

PIM operations. The activated data from each row buffer is then transferred through slicing Local I/O paths and IOSAs based on column indices. Finally, the weight data is routed via I/O logic to the xPUs, whereas the other partition's data flows into the MAC unit for GEMV computation with vector data from the global buffer.

Inter-bank: While LibraPIM enables DEX in multiple banks simultaneously, the shared data bus restricts read data transfers to one bank at a time. This constraint stems from the fixed size of the I/O logic, which is matched with a bandwidth of standard memory interconnects and cannot support concurrent data reads from multiple banks. Instead, LibraPIM uses DEX to read continuous data under this constraint. LibraPIM activates rows in multiple banks in advance to maintain high bus utilization under this constraint. Once the read from the currently active bank is completed, the system begins reading from another bank already activated via DEX. This interleaved access pattern overlaps the precharge and activation latencies across banks, ensuring continuous data delivery and maximizing data bus utilization during PIM operation.

4.4 Dynamic Batch Offloading

Key Idea: To mitigate both utilization and performance degradation caused by static allocation and workload imbalance (Section 3.1), we propose the DBO mechanism. DBO dynamically redistributes portions of batch groups across devices when a noticeable execution time gap is observed, utilizing DEX mechanism.

Figure 13a illustrates the scenario where xPU execution exceeds that of PIM. DBO offloads a portion of the xPU's batch group to LibraPIM. Since all batches share the same weight parameters, weights fetched by the xPU simultaneously serve as operands for PIM, enabling overlapped GEMM without additional data transfer. Figure 13b shows the opposite scenario, where PIM execution exceeds xPU. DBO transfers K,V vectors from selected batches to the xPU, enabling simultaneous attention computation on the xPU and GEMV on the PIM, effectively balancing execution times.

In contrast to the above two cases, partial offloading becomes inefficient when the batch group size is too small due to the overhead of offloading and memory reads. In such scenarios, shown in Figure 13c, LibraPIM offloads an entire batch group to the PIM. Both devices then process equivalent workloads in parallel. Since GEMM operations on small batches remain memory-intensive, PIM can complete their workloads nearly as fast as xPU's operation, achieving little synchronization overhead.

Hardware Operation for DBO: Figure 14 illustrates the detailed hardware mechanisms enabling DBO within a LibraPIM bank. Each DRAM bank is partitioned by slicing logic into two regions: the weight partition storing model weights accessed by the xPU, and the KV partition holding K,V matrices directly supplying operands to LibraPIM's MAC units for GEMV.

Upon initiating the computation phase, LibraPIM bank activates the necessary rows from both partitions: weights from the weight partition and input vectors from the KV partition. As depicted in Figure 14a, activated weight rows are sequentially transferred to the global buffer. Then, as Figure 14b illustrates, the global buffer simultaneously propagates these weights to both the xPU and LibraPIM MAC units, enabling concurrent xPU-based GEMM and LibraPIM-based GEMM.

When PIM's MHA execution time exceeds the xPU, input vectors selected from xPU for GEMV offloading are independent and unique per batch. Consequently, these vectors bypass the global buffer and are streamed directly through I/O logic.

4.5 Scheduling

We designed the DBO scheduler to determine the offloading batch size accurately. This scheduler is implemented in software and runs on the host. It predicts the processing time of each device and dynamically adjusts the offloading batch size accordingly. The host executes the scheduling process at the beginning of each embedding layer, which is the first layer during generation, as some batches may be completed while others still need processing. The system maintains a balanced workload distribution by continuously re-evaluating the offloading batch size.

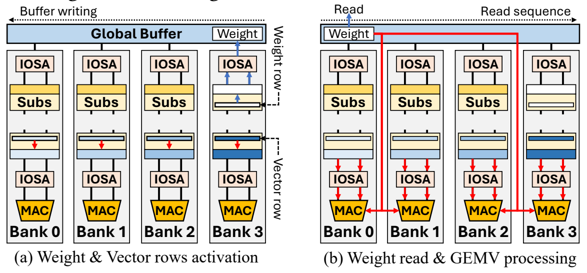


Figure 14: Batch offloading when processing large FFN.

Algorithm 1 Dynamic Batch Offloading Algorithm

```
1:  **Input:** Longest sequence length $L$, Size of batch group $B$
2:  $m, n$ : batch size for offloading
3:  $Channel_{PIM}$ : # of PIM channels
4:                                              ▷ Processing time ← (# of Bytes)/FLOPS
5:  $tQKV_{xPU} \leftarrow (K_{QKV} \cdot d^2_{model} \cdot B)/FLOPS_{xPU}$
6:  $tProj_{xPU} \leftarrow (K_{Proj} \cdot d^2_{model} \cdot B)/FLOPS_{xPU}$
7:  $tFFN_{xPU} \leftarrow (K_{FFN} \cdot d_{model} \cdot d_{ff} \cdot B)/FLOPS_{xPU}$
8:  $tMHA_{PIM} \leftarrow (K_{MHA} \cdot d_{model} \cdot B \cdot L)/FLOPS_{PIM}$
9:                                              ▷ Too small batch case
10: **if** $B < Channel_{PIM}$ **then**
11:     Batch_group_offload($B$)
12:     **return**
13:                                             ▷ Longer FFN case
14: **else if** $tQKV_{xPU} + tProj_{xPU} + tFFN_{xPU} > tMHA_{PIM}$ **then**
15:     $tFFN_{xPU} \leftarrow (K_{FFN} \cdot d_{model} \cdot d_{ff} \cdot (B - m))/FLOPS_{xPU}$
16:     $tFFN_{PIM} \leftarrow (K_{FFN} \cdot d_{model} \cdot d_{ff} \cdot m)/FLOPS_{xPU}$
17:     $m \leftarrow$ Offloading_FFN_size($tQKV_{xPU}, tProj_{xPU}, tFFN_{xPU}$
                $tMHA_{PIM}, tFFN_{PIM}$)
18:     Offload_FFN($m$)
19:     **return**
20:                                             ▷ Longer MHA case
21: **else if** $tQKV_{xPU} + tProj_{xPU} + tFFN_{xPU} < tMHA_{PIM}$ **then**
22:     $tMHA_{xPU} \leftarrow (K_{MHA} \cdot d_{model} \cdot n \cdot L)/FLOPS_{xPU\_GEMV}$
23:     $tMHA_{PIM} \leftarrow (K_{MHA} \cdot d_{model} \cdot (B - n) \cdot L)/FLOPS_{PIM}$
24:     $n \leftarrow$ Offloading_MHA_size($tQKV_{xPU}, tProj_{xPU}, tFFN_{xPU}$
                $tMHA_{xPU}, tMHA_{PIM}$)
25:     Offload_MHA($n$)
26:     **return**
27: **else**
28:     **return**
29: **end if**
```

The scheduler determines the offloading batch size using Algorithm 1 based on the FLOPs of each device. At the start of each generation stage, the scheduler receives several parameters, including input parameters and model parameters. Input parameters, such as sequence length and batch size, are determined by the inference task, while model parameters are fixed based on the model architecture. In this context, K_x is a linear constant that accounts for the ratio of the hidden layer per model (e.g., $K = 16$ in GPT3's FFN). Also, since the processing times among batches are different due to sequence length, the scheduler predicts processing time with the longest sequence which includes input length and generated tokens in each batch group as a dominant parameter.

In the first step, the scheduler calculates the processing time for each layer based on the allocated device throughput (lines 5–8). Since LibraPIM enables parallel execution, memory read latency, including vector processing time, is disregarded, which introduces minimal overhead in the overall execution. Next, the scheduler checks whether the batch group size is smaller than the number of available PIM channels, each of which can independently process a batch. If this condition holds, the entire batch group is offloaded to the PIM for parallel execution. If not, the scheduler compares the processing times of the devices to identify the one with the longer processing time and determines which workload to offload (lines 14 and 21). If offloading is needed, the scheduler recomputes the processing time for both devices and adjusts the offloading batch size (lines 15–17 and 22–24). Finally, it identifies the optimal batch size for offloading and executes the batch transfer (lines 17 and 25).

$$tQKV_{xPU} + tProj_{xPU} + tFFN_{xPU} = tMHA_{PIM} + tFFN_{PIM} \quad (1)$$

In lines 17 and 24, the offloading functions compare the processing times of the xPU and LibraPIM to determine the optimal offloading batch size by formulating an equation. Equation (1) calculates m, representing the batch size to be offloaded to the PIM device for FFN processing. The left-hand side of the equation represents the xPU's processing time, while the right-hand side accounts for the PIM, including the offloaded FFN workload. The scheduler determines the appropriate m by balancing processing times.

$$tQKV_{xPU} + tProj_{xPU} + tFFN_{xPU} + tMHA_{xPU} = tMHA_{PIM} \quad (2)$$

Similar to equation (1), equation (2) calculates n, representing the batch size to be offloaded to the xPUs for MHA processing. In this case, the left-hand side of the equation consists of four terms, as xPUs handle four different operations sequentially. Additionally, since xPUs cannot fully utilize their throughput in GEMV operations, the scheduler accounts for the FLOPs of xPUs required for GEMV when computing $tMHA_{xPU}$. By incorporating these factors, the scheduler ensures high utilization across devices.

5 Implementation

This section details the design of LibraPIM at both the hardware and system levels. We analyze the impact of integrating bank slicing logic, particularly considering timing constraints. Based on these findings, we present the realistic operation flow considering DRAM timing constraints and outline the mapping method for K, V, and weight data. Additionally, we introduce a dynamic partitioning technique that enables flexible bank partition size into equal halves and customized ratios to accommodate varying K, V, and weight sizes, ensuring adaptability across different workloads.

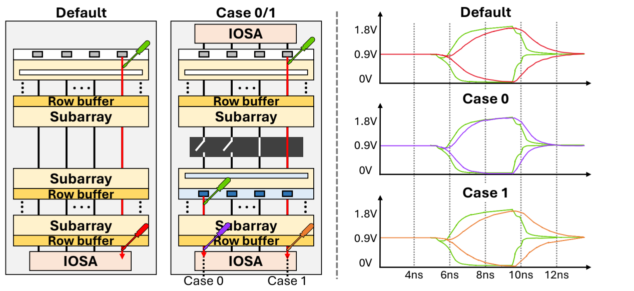


Figure 15: Circuit simulation waveform of segregation logic.

5.1 Circuit-level Implementation

To evaluate the impact of bank slicing logic at the circuit level, we designed a simplified LibraPIM bank model using Cadence Virtuoso [6]. Figure 15 illustrates the methodology and waveforms for three test cases. We use the original bank organization as a baseline and compare it against two modified cases: one with the partitioned bank and the other with the bank partitions connected through slicing logic. To assess the impact of bank slicing logic on latency, we measure the voltage propagation delay from the row buffer to the IOSA, which directly influences the tCCD timing constraint—the key timing unit for consecutive reads or PIM operations. We also evaluate the critical path by measuring the delay from the farthest subarray to the IOSA across all cases. As shown in the figure, the slicing logic introduces minimal impact on the tCCD constraint. In the sliced case (case 0), data transfer time is reduced due to the shortened path. In connected case (case 1), latency slightly increases, but the additional delay remains within acceptable bounds under modern DRAM constraints. Overall, the segregation logic introduces only a 2% timing overhead in the worst case, confirming its negligible impact on system.

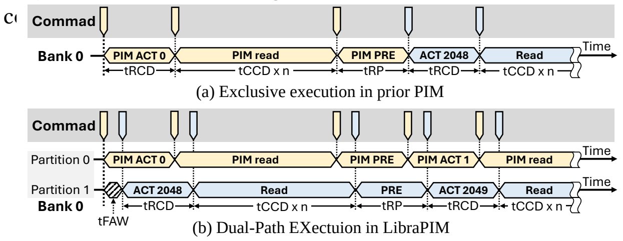
5.2 Operation Timing

Figure 16 presents a time diagram comparing a common PIM bank and a LibraPIM bank, considering DRAM timing constraints when executing PIM and read operations. We assume that all banks perform the same PIM operation in response to a single PIM command, enabling simultaneous execution— a common technique in PIM devices [3, 20, 27, 36]. As shown in (a), conventional PIM devices execute commands sequentially, waiting to complete the previous operation before proceeding. In contrast, LibraPIM enables parallel execution of PIM and read operations, effectively overlapping delayed execution time for improved efficiency. Additionally, due to tFAW constraints preventing voltage drops caused by consecutive row activations, LibraPIM schedules commands with tFAW intervals. However, tFAW contributes little to the total operation time by overlapping with consecutive tCCD. Moreover, since commercial PIM products are actively working to mitigate tFAW limitations to enable all-bank PIM operations, there is potential to further reduce tFAW overhead in LibraPIM's design [20, 36].

5.3 Data Mapping

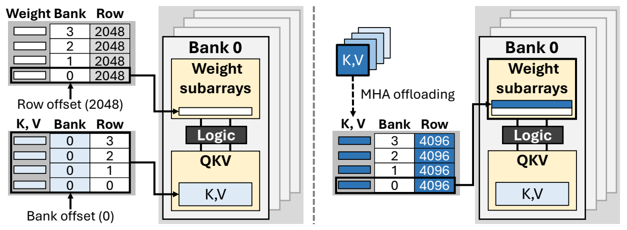
LibraPIM maps data based on their characteristics to enable fast data access in each operation, as illustrated in Figure 17. Weights are interleaved from the channel to the bank to read weight data fast, allowing data to be read across the entire channel in parallel and quickly reconstructed into the original matrix. However, since each MAC unit within a bank requires dedicated K, V data, each K, V matrix is gathered in allocated bank. This mapping strategy is implemented by setting an offset address for each operation, as shown in (a) of the figure. The system efficiently manages data across PIM devices and xPUs by leveraging these offset-based mappings.

In the case of MHA offloading, the K, V data allocated for offloading is stored in the weight partition to enable parallel execution. Since a bank cannot access two different rows within the same partition simultaneously, the K, V data for offloading is stored in the weight partition and accessed using parallel execution during MHA processing, as illustrated in (b) of Figure 17. For the same reason as weight mapping, xPUs can quickly receive offloaded K, V data by applying the same mapping method used for weights but with a different row offset. Additionally, since MHA offloading requires only K, V data and not weight data, the bank can avoid resource c



(b) Dual-Path EXectuion in LibraPIM

Figure 16: Timing diagram of normal PIM and LibraPIM interface when processing PIM and read operations.



(a) Weight & K,V mapping (b) K,V mapping for MHA offloading

Figure 17: Weight and KV mapping in LibraPIM.

5.4 Discussion

Address mapping: As discussed in the previous section, weight and K, V data require different data mapping schemes, necessitating distinct address mappings. In a conventional system, changing the mapping in real-time is infeasible since mapping method is determined by BIOS. Fortunately, several studies have explored techniques for enabling dynamic address mapping [11, 14, 58]. Furthermore, recent researches propose novel memory management techniques that can support both PIM and normal memory mapping [35, 61]. As memory management techniques are orthogonal to our approach, we can incorporate dynamic address mapping by adopting them to enhance flexibility and efficiency.

Fine-grained partitioning: Going further from dividing the bank into two parts, LibraPIM can dynamically partition the bank optionally by grouping multiple subarrays and combining them to form two partitions of different sizes for supporting operations with varying sizes of weight and K, V data. Figure 18 illustrates LibraPIM's fine-grained partitioning architecture, which enables flexible memory allocation. LibraPIM achieves this by deploying multiple segregation logic units among subarrays and selectively connecting them to construct two partitions according to a specified ratio (n: m in the figure). This partitioning ratio can be configured before inference begins, allowing for adaptive memory management in edge cases, such as scenarios with short sequence lengths and small batch sizes, where fixed partitioning may be inefficient.

6 Evaluation

6.1 Methodology

We evaluate the effectiveness of LibraPIM by comparing it against three baseline configurations: (1) NPU+PIM based on the AiM architecture [20], and two state-of-the-art heterogeneous systems, AttAcc [44] and NeuPIMs [21]. Each baseline adopts its respective optimization strategies: AttAcc applies head-level pipelining and FFN co-processing, NeuPIMs leverages head-level pipelining and batch grouping, and LibraPIM adopts DEX and DBO with batch pipelining. NPU+AiM does not apply any optimization techniques beyond standard execution.

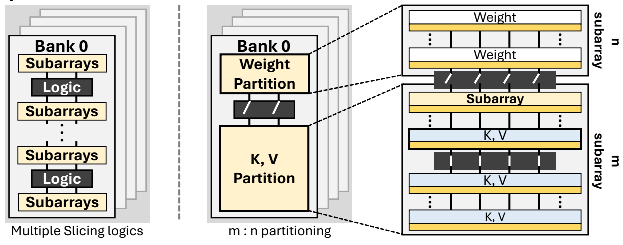
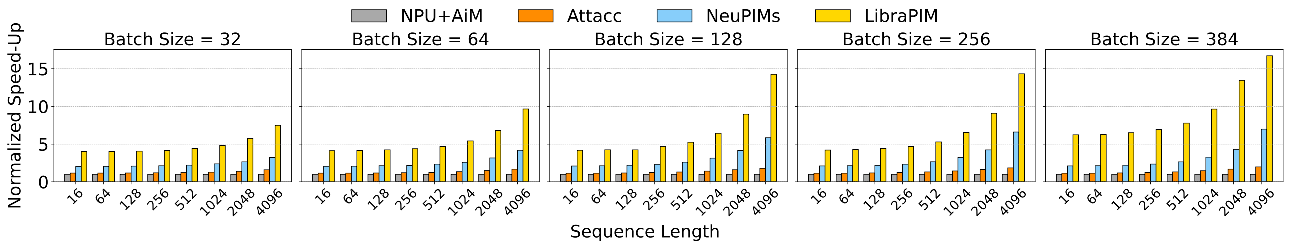
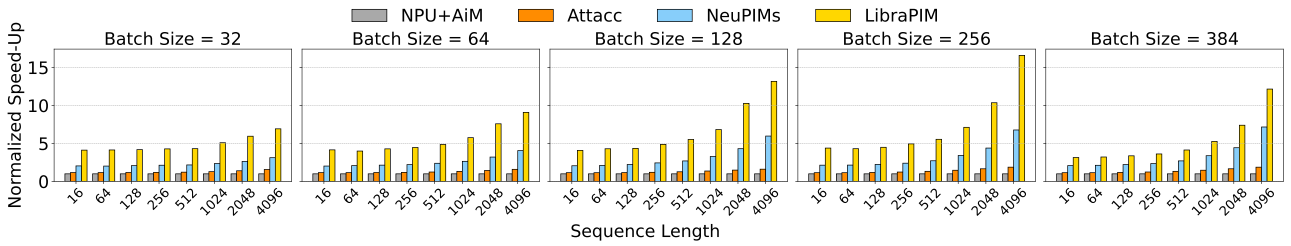


Figure 18: Fine-grained partitioning in LibraPIM.



(a) GPT3-175B



(b) Llama3-70B

Figure 19: Performance comparison of prior works and LibraPIM in GPT3-175B and Llama3-70B

Table 2: Configuration of simulated system

Hardware		
NPU	Systolic Arrays / Size	8 / 64 x 64
	Vector Units / Size	8 / 128 x 1
	Memory Type / Chips	HBM2 / 8
PIM	Type	HBM2
	Channel / Capacity	64 / 16 GB
	Bank / Bankgroup	4 / 4
	# of MAC units	16 units per bank
	tCL-tRCD-tRP	14-14-14
LLM Model	**# of Decoders**	**# of Heads**
GPT3-175B	96	96
Llama3-70B	80	64

We evaluate two widely used large language models in datacenter inference, GPT3 175B[5] and Llama3-70B[15], using input sequences sampled from the Alpaca dataset. We vary the input sequence length from 16 to 4096 and the batch size from 32 to 384. For performance evaluation, we measure token generation speed in each system and report their results relative to the NPU+AiM.

Our simulation environment builds upon the NeuPIMs heterogeneous system simulator [22], extended with the ONNXim NPU simulator [17, 23] and a modified version of DRAMsim3 [40]. We enhance DRAMsim3 to support PIM execution modeling and benchmark integration. To ensure fair comparisons, all systems adopt identical hardware and memory configurations. We summarize configuration of the simulator in Table 2. We further analyze system-level energy consumption, PIM utilization, and power breakdowns. The DRAM energy model follows JEDEC standards and incorporates read and PIM operations. Area overheads are estimated using CACTI 7.0 at 22nm process technology [4, 42].

6.2 Performance

Figure 19 compares LibraPIM against several baselines across two LLM workloads, with all results normalized to the NPU+AiM system. While all heterogeneous systems outperform the NPU+AiM baseline, LibraPIM consistently achieves the highest speedups, averaging 6.2x, 4.4x, and 2.1x over NPU+AiM, AttAcc, and NeuPIMs, respectively. We breakdown the results by batch size and sequence length since they significantly influence the performance.

Small batch size & short sequence length: In this case, the workload is dominated by NPU execution, but the NPU operates in a memory-bound region, resulting in long execution times with low computational efficiency. Meanwhile, the overlap between NPU and PIM computations remains narrow, as PIM has minimal workload. Under these conditions, LibraPIM achieves consistent performance gains primarily through batch group offloading via DBO, which offloads the entire group's FFN operations to PIM, enabling otherwise idle PIM resources to contribute to computation.

Large Batch Case: As the batch size increases, the execution time of the PIM increases more rapidly than that of the NPU. While the NPU continues to benefit from batching efficiency, the growing PIM workload results in the execution times of both devices becoming increasingly similar. This leads to a longer period during which NPU and PIM computations can proceed in parallel, allowing more effective use of DEX. In this setting, LibraPIM further applies DBO to offload FFN workloads to PIM. For instance, in the GPT3-175B workload with a batch size of 384, up to 64 batches—the number of available PIM channels—can be offloaded in parallel. When the PIM becomes more heavily loaded, LibraPIM shifts to MHA offloading to maintain parallelism. This combination of DEX and DBO contributes to performance improvements in large-batch configurations by maintaining high utilization across devices.

Long Sequence Case: With longer sequences, MHA computation increases quadratically and eventually exceeds the NPU's execution time. While short sequences are dominated by NPU, longer sequences shift the dominant workload to PIM. As the execution times of both devices become more comparable, the system benefits more from DEX. Even when PIM becomes the dominant component, LibraPIM maintains performance improvements by applying DBO to offload a portion of the MHA workload to the NPU. As a result, while LibraPIM and NeuPIMs show similar performance under GPT3-175B with batch size 256—where execution times are already balanced—LibraPIM outperforms at batch size 384, due to additional gains from MHA offloading.

Comparison with Prior Works: AttAcc employs FFN co-processing, which consistently outperforms NPU+AiM. However, its serialized processing flow between PIM and NPU imposes inherent limitations on overall speedup. In contrast, LibraPIM and NeuPIMs adopt batch pipelining to enable inter-group parallelism, which is particularly advantageous in batch inference scenarios. LibraPIM further extends this benefit through DEX, which prevents PIM operation blocking during overlapping NPU-PIM execution phases, thereby improving PIM utilization across all cases. Moreover, LibraPIM leverages DBO to mitigate workload imbalance, which can lead to full-group stalls.

6.3 Device Utilization

To evaluate system efficiency, we measure both PIM and NPU utilization with a batch size of 128. As shown in Figure 20a, PIM utilization remains near zero for AttAcc and NeuPIMs at short sequence lengths, limited by minimal MHA workload and waiting for NPU's workload to end. In contrast, LibraPIM achieves over 25% utilization even for short inputs, thanks to DBO, which enables parallel operation in the whole system. As MHA workload increases with sequence length, PIM utilization improves across all designs. Nevertheless, LibraPIM consistently leads, reaching up to 38% at the longest sequence. In Figure 20b, NPU utilization shows the opposite trend: high for short sequences and decreasing as squences grow longer. AttAcc declines sharply, while NeuPIM and LibraPIM maintain higher utilization by mitigating sequential dependencies via batch-grouping. LibraPIM sustains the highest utilization, aided by DBO, and achieves 77% utilization at the longest sequence length.

6.4 Sensitivity Analysis

We vary the configuration and measure performance to evaluate LibraPIM's scalability across different system configurations. Figure 21 shows the normalized performance under various NPU configurations, using a sequence length 512 and normalized to the 32×32 NPU without DBO. In small batch cases, performance gains remain modest across different systolic array sizes, as the NPU cannot fully utilize larger compute units due to limited parallelism, regardless of the PIM architecture. Additionally, the benefit of DBO is minimal for small batch sizes, as the workload is too small to enable meaningful offloading. In contrast, for batch size 256, we observe notable improvements in the 32×32 and 64×64 configurations. However, for larger NPUs (e.g., 128×128 and 256×256), this size is insufficient for offloading, regarding as a small batch size to them.

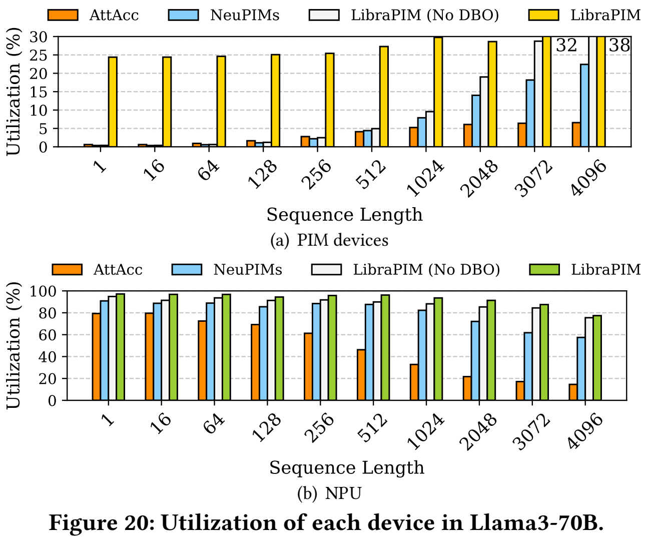


Figure 20: Utilization of each device in Llama3-70B.

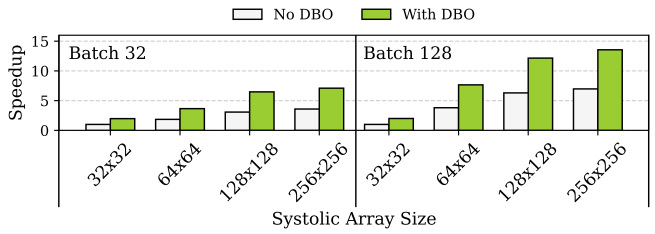


Figure 21: Performance comparison by NPU configuration.

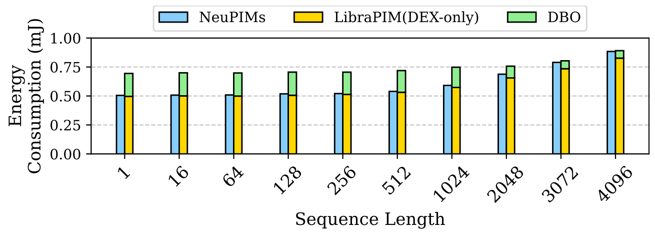


Figure 22: Energy consumption in Llama3-70B.

6.5 Area and Power Overheads

Table 3 summarizes the additional components introduced in LibraPIM, based on the AiM architecture's bank, along with their respective area overheads. The segregation logic, implemented using a simple transistor, adds a negligible overhead in the bank. Including an extra IOSA in each bank contributes only 0.002% overhead, as IOSA operates on decoded column data rather than the full subarray row size. Additionally, a dedicated column predecoder/decoder for PIM operations—added to enable parallel execution—introduces a modest overhead of 0.03%. LibraPIM achieves a minimal total area overhead of just 0.032%, demonstrating its hardware efficiency.

We present the energy consumption of LibraPIM in Figure 22 and compare it against NeuPIMs under the batch size of 128. Without DBO, LibraPIM exhibits similar energy usage due to the comparable computational workload required by MHA processing. However, with DBO enabled, LibraPIM shows higher energy consumption for shorter sequences, as DBO increases the number of PIM operations. This overhead diminishes as the sequence length grows since DBO reduces PIM activity in longer sequences. As a result, the total energy consumption of LibraPIM gradually aligns with that of NeuPIMs, with an average overhead of approximately 20%.

Table 3: Area overhead in LibraPIM bank.

Logic	Rate
Column predecoder & decoder	0.030%
Segregation logic	Negligible
IOSA	0.002%
Total	**0.032%**

7 Related Works

Processing-In-Memory devices: Building on the advantages of PIM technology, leading memory manufacturers have developed their own PIM devices. AiM incorporates multipliers and adder trees within each bank to accelerate GEMV processing [20, 36]. Similarly, HBM-PIM integrates multipliers and adders within each bank; however, these units operate independently, enabling support for general arithmetic operations. Both manufacturers have successfully fabricated their PIM architectures as commercial products [30, 31, 33, 34, 37]. UPMEM is a prototype PIM device that integrates a general-purpose processing unit directly within the memory die [10]. Since UPMEM's core is based on the RISC-V architecture, users can develop their applications using UPMEM's APIs, enabling programmable PIM environment.

Memory management in PIM: PIM operations require bank-centric data access, conflicting with conventional channel-interleaved address mappings. To address this, PIM-MMU introduces a hardware/software co-designed memory controller that remaps data into PIM-friendly regions by translating standard addresses into bank-consecutive ones [35]. It also offers user-level APIs for easy data management. Similarly, UM-PIM classifies memory pages as PIM or normal based on the physical address's MSB during address translation [61]. It then aggregates PIM-designated pages within specific banks and supports virtual address generation to unify data access across standard and PIM operations.

8 Conclusion

Heterogeneous PIM-xPU systems combine xPUs (i.e., GPUs or NPUs) and PIM devices to leverage their strengths for LLM inference. However, these systems often suffer from imbalanced processing times across devices, leading to underutilization and performance bottlenecks due to shared memory resources. We propose LibraPIM, a novel framework that addresses workload imbalance and improves utilization of computing resources through dynamic workload offloading. The underlying architecture, based on bank slicing, decouples DRAM access from PIM operations, enabling dual-path execution and allowing both operations to proceed concurrently without interference. This architecture resolves resource conflicts between DRAM access and PIM operations, enabling true parallelism. LibraPIM improves PIM utilization by up to 38% and overall performance by 2.1x compared to the state-of-the-art PIM-NPU system.

References

[1] Saurabh Agarwal, Bilge Acun, Basil Hosmer, Mostafa Elhoushi, Yejin Lee, Shivaram Venkataraman, Dimitris Papailiopoulos, and Carole-Jean Wu. 2024. CHAI: Clustered Head Attention for Efficient LLM Inference. arXiv:2403.08058 [cs.LG] https://arxiv.org/abs/2403.08058

[2] Junwhan Ahn, Sungpack Hong, Sungjoo Yoo, Onur Mutlu, and Kiyoung Choi. 2015. A scalable processing-in-memory accelerator for parallel graph processing. In *Proceedings of the 42nd Annual International Symposium on Computer Architecture* (Portland, Oregon) *(ISCA '15)*. Association for Computing Machinery, New York, NY, USA, 105–117. doi:10.1145/2749469.2750386

[3] Daehyeon Baek, Soojin Hwang, and Jaehyuk Huh. 2024. pSyncPIM: Partially Synchronous Execution of Sparse Matrix Operations for All-Bank PIM Architectures. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*. 354–367. doi:10.1109/ISCA59077.2024.00034

[4] Rajeev Balasubramonian, Andrew B. Kahng, Naveen Muralimanohar, Ali Shafiee, and Vaishnav Srinivas. 2017. CACTI 7: New Tools for Interconnect Exploration in Innovative Off-Chip Memories. *ACM Trans. Archit. Code Optim.* 14, 2, Article 14 (jun 2017), 25 pages. doi:10.1145/3085572

[5] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (Eds.), Vol. 33. Curran Associates, Inc., 1877–1901. https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf

[6] Cadence. [n. d.]. Cadence Virtuoso. https://www.cadence.com/en_US/home/tools/custom-ic-analog-rf-design/layout-design/virtuoso-layout-suite.html.

[7] Niladrish Chatterjee, Mike O'Connor, Donghyuk Lee, Daniel R. Johnson, Stephen W. Keckler, Minsoo Rhu, and William J. Dally. 2017. Architecting an Energy-Efficient DRAM System for GPUs. In *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 73–84. doi:10.1109/HPCA.2017.58

[8] Yu-Hsin Chen, Tushar Krishna, Joel S. Emer, and Vivienne Sze. 2017. Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks. *IEEE Journal of Solid-State Circuits* 52, 1 (2017), 127–138. doi:10.1109/JSSC.2016.2616357

[9] Zhoujun Cheng, Jungo Kasai, and Tao Yu. 2023. Batch Prompting: Efficient Inference with Large Language Model APIs. arXiv:2301.08721 [cs.CL] https://arxiv.org/abs/2301.08721

[10] F. Devaux. 2019. The true Processing In Memory accelerator. In *2019 IEEE Hot Chips 31 Symposium (HCS)*. IEEE Computer Society, Los Alamitos, CA, USA, 1–24. doi:10.1109/HOTCHIPS.2019.8875680

[11] Alexandar Devic, Siddhartha Balakrishna Rai, Anand Sivasubramaniam, Ameen Akel, Sean Eilert, and Justin Eno. 2022. To PIM or not for emerging general purpose processing in DDR memory systems. In *Proceedings of the 49th Annual International Symposium on Computer Architecture* (New York, New York) *(ISCA '22)*. Association for Computing Machinery, New York, NY, USA, 231–244. doi:10.1145/3470496.3527431

[12] João Dinis Ferreira, Gabriel Falcao, Juan Gómez-Luna, Mohammed Alser, Lois Orosa, Mohammad Sadrosadati, Jeremie S. Kim, Geraldo F. Oliveira, Taha Shahroodi, Anant Nori, and Onur Mutlu. 2022. pLUTo: Enabling Massively Parallel Computation in DRAM via Lookup Tables. In *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 900–919. doi:10.1109/MICRO56248.2022.00067

[13] Mingyu Gao, Jing Pu, Xuan Yang, Mark Horowitz, and Christos Kozyrakis. 2017. Tetris: Scalable and efficient neural network acceleration with 3d memory. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*. 751–764.

[14] Mohsen Ghasempour, Aamer Jaleel, Jim D. Garside, and Mikel Luján. 2016. DReAM: Dynamic Re-arrangement of Address Mapping to Improve the Performance of DRAMs. In *Proceedings of the Second International Symposium on Memory Systems* (Alexandria, VA, USA) *(MEMSYS '16)*. Association for Computing Machinery, New York, NY, USA, 362–373. doi:10.1145/2989081.2989102

[15] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurelien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Roziere, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, Danny Wyatt, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego

Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Francisco Guzmán, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Govind Thattai, Graeme Nail, Gregoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel Kloumann, Ishan Misra, Ivan Evtimov, Jack Zhang, Jade Copet, Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Mahadeokar, Jeet Shah, Jelmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu, Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak, Jongsoo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Junteng Jia, Kalyan Vasuden Alwala, Karthik Prasad, Kartikeya Upasani, Kate Plawiak, Ke Li, Kenneth Heafield, Kevin Stone, Khalid El-Arini, Krithika Iyer, Kshitiz Malik, Kuenley Chiu, Kunal Bhalla, Kushal Lakhotia, Lauren Rantala-Yeary, Laurens van der Maaten, Lawrence Chen, Liang Tan, Liz Jenkins, Louis Martin, Lovish Madaan, Lubo Malo, Lukas Blecher, Lukas Landzaat, Luke de Oliveira, Madeline Muzzi, Mahesh Pasupuleti, Mannat Singh, Manohar Paluri, Marcin Kardas, Maria Tsimpoukelli, Mathew Oldham, Mathieu Rita, Maya Pavlova, Melanie Kambadur, Mike Lewis, Min Si, Mitesh Kumar Singh, Mona Hassan, Naman Goyal, Narjes Torabi, Nikolay Bashlykov, Nikolay Bogoychev, Niladri Chatterji, Ning Zhang, Olivier Duchenne, Onur Çelebi, Patrick Alrassy, Pengchuan Zhang, Pengwei Li, Petar Vasic, Peter Weng, Prajjwal Bhargava, Pratik Dubal, Praveen Krishnan, Punit Singh Koura, Puxin Xu, Qing He, Qingxiao Dong, Ragavan Srinivasan, Raj Ganapathy, Ramon Calderer, Ricardo Silveira Cabral, Robert Stojnic, Roberta Raileanu, Rohan Maheswari, Rohit Girdhar, Rohit Patel, Romain Sauvestre, Ronnie Polidoro, Roshan Sumbaly, Ross Taylor, Ruan Silva, Rui Hou, Rui Wang, Saghar Hosseini, Sahana Chennabasappa, Sanjay Singh, Sean Bell, Seohyun Sonia Kim, Sergey Edunov, Shaoliang Nie, Sharan Narang, Sharath Raparthy, Sheng Shen, Shengye Wan, Shruti Bhosale, Shun Zhang, Simon Vandenhende, Soumya Batra, Spencer Whitman, Sten Sootla, Stephane Collot, Suchin Gururangan, Sydney Borodinsky, Tamar Herman, Tara Fowler, Tarek Sheasha, Thomas Georgiou, Thomas Scialom, Tobias Speckbacher, Todor Mihaylov, Tong Xiao, Ujjwal Karn, Vedanuj Goswami, Vibhor Gupta, Vignesh Ramanathan, Viktor Kerkez, Vincent Gonguet, Virginie Do, Vish Vogeti, Vítor Albiero, Vladan Petrovic, Weiwei Chu, Wenhan Xiong, Wenyin Fu, Whitney Meers, Xavier Martinet, Xiaodong Wang, Xiaofang Wang, Xiaoqing Ellen Tan, Xide Xia, Xinfeng Xie, Xuchao Jia, Xuewei Wang, Yaelle Goldschlag, Yashesh Gaur, Yasmine Babaei, Yi Wen, Yiwen Song, Yuchen Zhang, Yue Li, Yuning Mao, Zacharie Delpierre Coudert, Zheng Yan, Zhengxing Chen, Zoe Papakipos, Aaditya Singh, Aayushi Srivastava, Abha Jain, Adam Kelsey, Adam Shajnfeld, Adithya Gangidi, Adolfo Victoria, Ahuva Goldstand, Ajay Menon, Ajay Sharma, Alex Boesenberg, Alexei Baevski, Allie Feinstein, Amanda Kallet, Amit Sangani, Amos Teo, Anam Yunus, Andrei Lupu, Andres Alvarado, Andrew Caples, Andrew Gu, Andrew Ho, Andrew Poulton, Andrew Ryan, Ankit Ramchandani, Annie Dong, Annie Franco, Anuj Goyal, Aparajita Saraf, Arkabandhu Chowdhury, Ashley Gabriel, Ashwin Bharambe, Assaf Eisenman, Azadeh Yazdan, Beau James, Ben Maurer, Benjamin Leonhardi, Bernie Huang, Beth Loyd, Beto De Paola, Bhargavi Paranjape, Bing Liu, Bo Wu, Boyu Ni, Braden Hancock, Bram Wasti, Brandon Spence, Brani Stojkovic, Brian Gamido, Britt Montalvo, Carl Parker, Carly Burton, Catalina Mejia, Ce Liu, Changhan Wang, Changkyu Kim, Chao Zhou, Chester Hu, Ching-Hsiang Chu, Chris Cai, Chris Tindal, Christoph Feichtenhofer, Cynthia Gao, Damon Civin, Dana Beaty, Daniel Kreymer, Daniel Li, David Adkins, David Xu, Davide Testuggine, Delia David, Devi Parikh, Diana Liskovich, Didem Foss, Dingkang Wang, Duc Le, Dustin Holland, Edward Dowling, Eissa Jamil, Elaine Montgomery, Eleonora Presani, Emily Hahn, Emily Wood, Eric-Tuan Le, Erik Brinkman, Esteban Arcaute, Evan Dunbar, Evan Smothers, Fei Sun, Felix Kreuk, Feng Tian, Filippos Kokkinos, Firat Ozgenel, Francesco Caggioni, Frank Kanayet, Frank Seide, Gabriela Medina Florez, Gabriella Schwarz, Gada Badeer, Georgia Swee, Gil Halpern, Grant Herman, Grigory Sizov, Guangyi, Zhang, Guna Lakshminarayanan, Hakan Inan, Hamid Shojanazeri, Han Zou, Hannah Wang, Hanwen Zha, Haroun Habeeb, Harrison Rudolph, Helen Suk, Henry Aspegren, Hunter Goldman, Hongyuan Zhan, Ibrahim Damlaj, Igor Molybog, Igor Tufanov, Ilias Leontiadis, Irina-Elena Veliche, Itai Gat, Jake Weissman, James Geboski, James Kohli, Janice Lam, Japhet Asher, Jean-Baptiste Gaya, Jeff Marcus, Jeff Tang, Jennifer Chan, Jenny Zhen, Jeremy Reizenstein, Jeremy Teboul, Jessica Zhong, Jian Jin, Jingyi Yang, Joe Cummings, Jon Carvill, Jon Shepard, Jonathan McPhie, Jonathan Torres, Josh Ginsburg, Junjie Wang, Kai Wu, Kam Hou U, Karan Saxena, Kartikay Khandelwal, Katayoun Zand, Kathy Matosich, Kaushik Veeraraghavan, Kelly Michelena, Keqian Li, Kiran Jagadeesh, Kun Huang, Kunal Chawla, Kyle Huang, Lailin Chen, Lakshya Garg, Lavender A, Leandro Silva, Lee Bell, Lei Zhang, Liangpeng Guo, Licheng Yu, Liron Moshkovich, Luca Wehrstedt, Madian Khabsa, Manav Avalani, Manish Bhatt, Martynas Mankus, Matan Hasson, Matthew Lennie, Matthias Reso, Maxim Groshev, Maxim Naumov, Maya Lathi, Meghan Keneally, Miao Liu, Michael L. Seltzer, Michal Valko, Michelle Restrepo, Mihir Patel, Mik Vyatskov, Mikayel Samvelyan, Mike Clark, Mike Macey, Mike Wang, Miquel Jubert Hermoso, Mo Metanat, Mohammad Rastegari, Munish Bansal, Nandhini Santhanam, Natascha Parks, Natasha White, Navyata Bawa, Nayan Singhal, Nick Egebo, Nicolas Usunier, Nikhil Mehta, Nikolay Pavlovich Laptev, Ning Dong, Norman Cheng, Oleg Chernoguz, Olivia Hart, Omkar Salpekar, Ozlem Kalinli, Parkin Kent, Parth Parekh, Paul Saab, Pavan Balaji, Pedro Rittner, Philip Bontrager, Pierre Roux, Piotr Dollar, Polina Zvyagina, Prashant Ratanchandani, Pritish Yuvraj, Qian Liang, Rachad Alao, Rachel Rodriguez, Rafi Ayub, Raghotham Murthy, Raghu Nayani, Rahul Mitra, Rangaprabhu Parthasarathy, Raymond Li, Rebekkah Hogan, Robin Battey, Rocky Wang, Russ Howes, Ruty Rinott, Sachin Mehta, Sachin Siby, Sai Jayesh Bondu, Samyak Datta, Sara Chugh, Sara Hunt, Sargun Dhillon, Sasha Sidorov, Satadru Pan, Saurabh Mahajan, Saurabh Verma, Seiji Yamamoto, Sharadh Ramaswamy, Shaun Lindsay, Shaun Lindsay, Sheng Feng, Shenghao Lin, Shengxin Cindy Zha, Shishir Patil, Shiva Shankar, Shuqiang Zhang, Shuqiang Zhang, Sinong Wang, Sneha Agarwal, Soji Sajuyigbe, Soumith Chintala, Stephanie Max, Stephen Chen, Steve Kehoe, Steve Satterfield, Sudarshan Govindaprasad, Sumit Gupta, Summer Deng, Sungmin Cho, Sunny Virk, Suraj Subramanian, Sy Choudhury, Sydney Goldman, Tal Remez, Tamar Glaser, Tamara Best, Thilo Koehler, Thomas Robinson, Tianhe Li, Tianjun Zhang, Tim Matthews, Timothy Chou, Tzook Shaked, Varun Vontimitta, Victoria Ajayi, Victoria Montanez, Vijai Mohan, Vinay Satish Kumar, Vishal Mangla, Vlad Ionescu, Vlad Poenaru, Vlad Tiberiu Mihailescu, Vladimir Ivanov, Wei Li, Wenchen Wang, Wenwen Jiang, Wes Bouaziz, Will Constable, Xiaocheng Tang, Xiaojian Wu, Xiaolan Wang, Xilun Wu, Xinbo Gao, Yaniv Kleinman, Yanjun Chen, Ye Hu, Ye Jia, Ye Qi, Yenda Li, Yilin Zhang, Ying Zhang, Yossi Adi, Youngjin Nam, Yu, Wang, Yu Zhao, Yuchen Hao, Yundi Qian, Yunlu Li, Yuzi He, Zach Rait, Zachary DeVito, Zef Rosnbrick, Zhaoduo Wen, Zhenyu Yang, Zhiwei Zhao, and Zhiyu Ma. 2024. The Llama 3 Herd of Models. arXiv:2407.21783 [cs.AI] https://arxiv.org/abs/2407.21783

[16] Ozgur Guldogan, Jackson Kunde, Kangwook Lee, and Ramtin Pedarsani. 2024. Multi-Bin Batching for Increasing LLM Inference Throughput. arXiv:2412.04504 [cs.CL] https://arxiv.org/abs/2412.04504

[17] Hyungkyu Ham, Wonhyuk Yang, Yunseon Shin, Okkyun Woo, Guseul Heo, Sangyeop Lee, Jongse Park, and Gwangsun Kim. 2024. ONNXim: A Fast, Cycle-level Multi-core NPU Simulator. *arXiv preprint arXiv:2406.08051* (2024).

[18] Tae Jun Ham, Sung Jun Jung, Seonghak Kim, Young H Oh, Yeonhong Park, Yoonho Song, Jung-Hun Park, Sanghee Lee, Kyoung Park, Jae W Lee, et al. 2020. Aˆ 3: Accelerating attention mechanisms in neural networks with approximation. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 328–341.

[19] Tae Jun Ham, Yejin Lee, Seong Hoon Seo, Soosung Kim, Hyunji Choi, Sung Jun Jung, and Jae W Lee. 2021. 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, 692–705.

[20] Mingxuan He, Choungki Song, Ilkon Kim, Chunseok Jeong, Seho Kim, Il Park, Mithuna Thottethodi, and T. N. Vijaykumar. 2020. Newton: A DRAM-maker's Accelerator-in-Memory (AiM) Architecture for Machine Learning. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 372–385. doi:10.1109/MICRO50266.2020.00040

[21] Guseul Heo, Sangyeop Lee, Jaehong Cho, Hyunmin Choi, Sanghyeon Lee, Hyungkyu Ham, Gwangsun Kim, Divya Mahajan, and Jongse Park. 2024. 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* (La Jolla, CA, USA) *(ASPLOS '24)*. Association for Computing Machinery, New York, NY, USA, 722–737. doi:10.1145/3620666.3651380

[22] Guseul Heo, Sangyeop Lee, Jaehong Cho, Hyunmin Choi, Sanghyeon Lee, Hyungkyu Ham, Gwangsun Kim, Divya Mahajan, and Jongse Park. 2024. NeuPIMs Simulator. https://github.com/casys-kaist/NeuPIMs.

[23] Yunseon Shin Okkyun Woo Guseul Heo Sangyeop Lee Jongse Park Gwangsun Kim Hyungkyu Ham, Wonhyuk Yang. 2024. ONNXim: A Fast, Cycle-level Multi-core NPU Simulator. arXiv:2406.08051 [cs.AR] https://arxiv.org/abs/2406.08051

[24] JEDEC. 2016. High Bandwidth Memory DRAM (HBM1, HBM2). https://www.jedec.org/sites/default/files/docs/JESD235D.pdf.

[25] Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. 2024. Mixtral of experts. *arXiv preprint arXiv:2401.04088* (2024).

[26] Norm Jouppi, George Kurian, Sheng Li, Peter Ma, Rahul Nagarajan, Lifeng Nai, Nishant Patil, Suvinay Subramanian, Andy Swing, Brian Towles, Clifford Young, Xiang Zhou, Zongwei Zhou, and David A Patterson. 2023. TPU v4: An Optically Reconfigurable Supercomputer for Machine Learning with Hardware Support for Embeddings. In *Proceedings of the 50th Annual International Symposium on Computer Architecture* (Orlando, FL, USA) *(ISCA '23)*. Association for Computing Machinery, New York, NY, USA, Article 82, 14 pages. doi:10.1145/3579371.3589350

[27] Hongju Kal, Chanyoung Yoo, and Won Woo Ro. 2023. AESPA: Asynchronous Execution Scheme to Exploit Bank-Level Parallelism of Processing-in-Memory. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture* (Toronto, ON, Canada) *(MICRO '23)*. Association for Computing Machinery, New York, NY, USA, 815–827. doi:10.1145/3613424.3614314

[28] Shinhaeng Kang, Sukhan Lee, Byeongho Kim, Hweesoo Kim, Kyomin Sohn, Nam Sung Kim, and Eojin Lee. 2022. An FPGA-based RNN-T Inference Accelerator with PIM-HBM. In *Proceedings of the 2022 ACM/SIGDA International Symposium*

on Field-Programmable Gate Arrays (Virtual Event, USA) *(FPGA '22)*. Association for Computing Machinery, New York, NY, USA, 146–152. doi:10.1145/3490422.3502355

[29] Duckhwan Kim, Jaeha Kung, Sek Chai, Sudhakar Yalamanchili, and Saibal Mukhopadhyay. 2016. Neurocube: a programmable digital neuromorphic architecture with high-density 3D memory. *SIGARCH Comput. Archit. News* 44, 3 (jun 2016), 380–392. doi:10.1145/3007787.3001178

[30] Jin Hyun Kim, Shin-Haeng Kang, Sukhan Lee, Hyeonsu Kim, Yuhwan Ro, Seungwon Lee, David Wang, Jihyun Choi, Jinin So, YeonGon Cho, JoonHo Song, Jeonghyeon Cho, Kyomin Sohn, and Nam Sung Kim. 2022. Aquabolt-XL HBM2-PIM, LPDDR5-PIM With In-Memory Processing, and AXDIMM With Acceleration Buffer. *IEEE Micro* 42, 3 (2022), 20–30. doi:10.1109/MM.2022.3164651

[31] Jin Hyun Kim, Shin-haeng Kang, Sukhan Lee, Hyeonsu Kim, Woongjae Song, Yuhwan Ro, Seungwon Lee, David Wang, Hyunsung Shin, Bengseng Phuah, Jihyun Choi, Jinin So, YeonGon Cho, JoonHo Song, Jangseok Choi, Jeonghyeon Cho, Kyomin Sohn, Youngsoo Sohn, Kwangil Park, and Nam Sung Kim. 2021. Aquabolt-XL: Samsung HBM2-PIM with in-memory processing for ML accelerators and beyond. In *2021 IEEE Hot Chips 33 Symposium (HCS)*. 1–26. doi:10.1109/HCS52781.2021.9567191

[32] Seongwook Kim, Gwangeun Byeon, Sihyung Kim, Hyungjin Kim, and Seokin Hong. 2023. Conveyor: Towards Asynchronous Dataflow in Systolic Array to Exploit Unstructured Sparsity. In *2023 IEEE 41st International Conference on Computer Design (ICCD)*. 423–431. doi:10.1109/ICCD58817.2023.00070

[33] Daehan Kwon, Seongju Lee, Kyuyoung Kim, Sanghoon Oh, Joonhong Park, Gi-Moon Hong, Dongyoon Ka, Kyudong Hwang, Jeongje Park, Kyeongpil Kang, Jungyeon Kim, Junyeol Jeon, Nahsung Kim, Yongkee Kwon, Vladimir Kornijcuk, Woojae Shin, Jongsoon Won, Minkyu Lee, Hyunha Joo, Haerang Choi, Guhyun Kim, Byeongju An, Jaewook Lee, Donguc Ko, Younggun Jun, Ilwoong Kim, Choungki Song, Ilkon Kim, Chanwook Park, Seho Kim, Chunseok Jeong, Euicheol Lim, Dongkyun Kim, Jieun Jang, Il Park, Junhyun Chun, and Joohwan Cho. 2023. A 1ynm 1.25V 8Gb 16Gb/s/Pin GDDR6-Based Accelerator-in-Memory Supporting 1TFLOPS MAC Operation and Various Activation Functions for Deep Learning Application. *IEEE Journal of Solid-State Circuits* 58, 1 (2023), 291–302. doi:10.1109/JSSC.2022.3200718

[34] Young-Cheon Kwon, Suk Han Lee, Jaehoon Lee, Sang-Hyuk Kwon, Je Min Ryu, Jong-Pil Son, O Seongil, Hak-Soo Yu, Haesuk Lee, Soo Young Kim, Youngmin Cho, Jin Guk Kim, Jongyoon Choi, Hyun-Sung Shin, Jin Kim, BengSeng Phuah, HyoungMin Kim, Myeong Jun Song, Ahn Choi, Daeho Kim, SooYoung Kim, Eun-Bong Kim, David Wang, Shinhaeng Kang, Yuhwan Ro, Seungwoo Seo, JoonHo Song, Jaeyoun Youn, Kyomin Sohn, and Nam Sung Kim. 2021. 25.4 A 20nm 6GB Function-In-Memory DRAM, Based on HBM2 with a 1.2TFLOPS Programmable Computing Unit Using Bank-Level Parallelism, for Machine Learning Applications. In *2021 IEEE International Solid-State Circuits Conference (ISSCC)*, Vol. 64. 350–352. doi:10.1109/ISSCC42613.2021.9365862

[35] Dongjae Lee, Bongjoon Hyun, Taehun Kim, and Minsoo Rhu. 2024. PIM-MMU: A Memory Management Unit for Accelerating Data Transfers in Commercial PIM Systems. In *2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 627–642. doi:10.1109/MICRO61859.2024.00053

[36] Sukhan Lee, Shin-haeng Kang, Jaehoon Lee, Hyeonsu Kim, Eojin Lee, Seungwoo Seo, Hosang Yoon, Seungwon Lee, Kyounghwan Lim, Hyunsung Shin, Jinhyun Kim, O Seongil, Anand Iyer, David Wang, Kyomin Sohn, and Nam Sung Kim. 2021. 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)*. 43–56. doi:10.1109/ISCA52012.2021.00013

[37] Seongju Lee, Kyuyoung Kim, Sanghoon Oh, Joonhong Park, Gimoon Hong, Dongyoon Ka, Kyudong Hwang, Jeongje Park, Kyeongpil Kang, Jungyeon Kim, Junyeol Jeon, Nahsung Kim, Yongkee Kwon, Kornijcuk Vladimir, Woojae Shin, Jongsoon Won, Minkyu Lee, Hyunha Joo, Haerang Choi, Jaewook Lee, Donguc Ko, Younggun Jun, Keewon Cho, Ilwoong Kim, Choungki Song, Chunseok Jeong, Daehan Kwon, Jieun Jang, Il Park, Junhyun Chun, and Joohwan Cho. 2022. A 1ynm 1.25V 8Gb, 16Gb/s/pin GDDR6-based Accelerator-in-Memory supporting 1TFLOPS MAC Operation and Various Activation Functions for Deep-Learning Applications. In *2022 IEEE International Solid-State Circuits Conference (ISSCC)*, Vol. 65. 1–3. doi:10.1109/ISSCC42614.2022.9731711

[38] Cong Li, Zhe Zhou, Yang Wang, Fan Yang, Ting Cao, Mao Yang, Yun Liang, and Guangyu Sun. 2024. 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* (La Jolla, CA, USA) *(ASPLOS '24)*. Association for Computing Machinery, New York, NY, USA, 879–896. doi:10.1145/3620665.3640376

[39] Junnan Li, Ramprasaath Selvaraju, Akhilesh Gotmare, Shafiq Joty, Caiming Xiong, and Steven Chu Hong Hoi. 2021. Align before fuse: Vision and language representation learning with momentum distillation. *Advances in neural information processing systems* 34 (2021), 9694–9705.

[40] Shang Li, Zhiyuan Yang, Dhiraj Reddy, Ankur Srivastava, and Bruce Jacob. 2020. DRAMsim3: A Cycle-Accurate, Thermal-Capable DRAM Simulator. *IEEE Computer Architecture Letters* 19, 2 (2020), 106–109. doi:10.1109/LCA.2020.2973991

[41] Yi Lu, Xin Zhou, Wei He, Jun Zhao, Tao Ji, Tao Gui, Qi Zhang, and Xuanjing Huang. 2024. LongHeads: Multi-Head Attention is Secretly a Long Context Processor. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (Eds.). Association for Computational Linguistics, Miami, Florida, USA, 7136–7148. doi:10.18653/v1/2024.findings-emnlp.417

[42] Vaishnav Srinivas Naveen Muralimanohar, Ali Shafiee. 2017. CACTI 7.0. https://github.com/HewlettPackard/cacti.

[43] Mike O'Connor, Niladrish Chatterjee, Donghyuk Lee, John Wilson, Aditya Agrawal, Stephen W Keckler, and William J Dally. 2017. Fine-grained DRAM: Energy-efficient DRAM for extreme bandwidth systems. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 41–54.

[44] Jaehyun Park, Jaewan Choi, Kwanhee Kyung, Michael Jaemin Kim, Yongsuk Kwon, Nam Sung Kim, and Jung Ho Ahn. 2024. AttAcc! Unleashing the Power of PIM for Batched Transformer-based Generative Model Inference *(ASPLOS '24)*. Association for Computing Machinery, New York, NY, USA, 103–119. doi:10.1145/3620665.3640422

[45] Pratyush Patel, Esha Choukse, Chaojie Zhang, Aashaka Shah, Íñigo Goiri, Saeed Maleki, and Ricardo Bianchini. 2024. Splitwise: Efficient Generative LLM Inference Using Phase Splitting. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*. 118–132. doi:10.1109/ISCA59077.2024.00019

[46] Akshay Krishna Ramanathan, Gurpreet S Kalsi, Srivatsa Srinivasa, Tarun Makesh Chandran, Kamlesh R Pillai, Om J Omer, Vijaykrishnan Narayanan, and Sreenivas Subramoney. 2020. Look-Up Table based Energy Efficient Processing in Cache Support for Neural Network Acceleration. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 88–101. doi:10.1109/MICRO50266.2020.00020

[47] Stephen Jones Nick Stam Ronny Krashinsky, Olivier Giroux and Sridhar Ramaswamy. 2020. NVIDIA Ampere Architecture In-Depth. https://developer.nvidia.com/blog/nvidia-ampere-architecture-in-depth/.

[48] Stephen Jones Nick Stam Ronny Krashinsky, Olivier Giroux and Sridhar Ramaswamy. 2023. NVIDIA H100 Tensor Core GPU. https://resources.nvidia.com/en-us-tensor-core/nvidia-tensor-core-gpu-datasheet.

[49] Minseok Seo, Xuan Truong Nguyen, Seok Joong Hwang, Yongkee Kwon, Guhyun Kim, Chanwook Park, Ilkon Kim, Jaehan Park, Jeongbin Kim, Woojae Shin, Jongsoon Won, Haerang Choi, Kyuyoung Kim, Daehan Kwon, Chunseok Jeong, Sangheon Lee, Yongseok Choi, Wooseok Byun, Seungcheol Baek, Hyuk-Jae Lee, and John Kim. 2024. IANUS: Integrated Accelerator based on NPU-PIM Unified Memory System *(ASPLOS '24)*. Association for Computing Machinery, New York, NY, USA, 545–560. doi:10.1145/3620666.3651324

[50] Vivek Seshadri, Donghyuk Lee, Thomas Mullins, Hasan Hassan, Amirali Boroumand, Jeremie Kim, Michael A. Kozuch, Onur Mutlu, Phillip B. Gibbons, and Todd C. Mowry. 2017. Ambit: in-memory accelerator for bulk bitwise operations using commodity DRAM technology. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture* (Cambridge, Massachusetts) *(MICRO-50 '17)*. Association for Computing Machinery, New York, NY, USA, 273–287. doi:10.1145/3123939.3124544

[51] Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, Katie Millican, et al. 2023. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805* (2023).

[52] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023. Llama 2: Open Foundation and Fine-Tuned Chat Models. arXiv:2307.09288 [cs.CL] https://arxiv.org/abs/2307.09288

[53] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems*, I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.), Vol. 30. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

[54] Peng Wang, An Yang, Rui Men, Junyang Lin, Shuai Bai, Zhikang Li, Jianxin Ma, Chang Zhou, Jingren Zhou, and Hongxia Yang. 2022. Ofa: Unifying architectures, tasks, and modalities through a simple sequence-to-sequence learning framework. In *International conference on machine learning*. ICML, 23318–23340.

[55] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems* 35 (2022), 24824–24837.

[56] Amir Yazdanbakhsh, Choungki Song, Jacob Sacks, Pejman Lotfi-Kamran, Hadi Esmaeilzadeh, and Nam Sung Kim. 2018. In-DRAM near-data approximate acceleration for GPUs. In *Proceedings of the 27th International Conference on Parallel Architectures and Compilation Techniques* (Limassol, Cyprus) *(PACT '18)*. Association for Computing Machinery, New York, NY, USA, Article 34, 14 pages. doi:10.1145/3243176.3243188

[57] Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. 2022. Orca: A Distributed Serving System for Transformer-Based Generative Models. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. USENIX Association, Carlsbad, CA, 521–538. https://www.usenix.org/conference/osdi22/presentation/yu

[58] Jialiang Zhang, Michael Swift, and Jing (Jane) Li. 2022. Software-defined address mapping: a case on 3D memory. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems* (Lausanne, Switzerland) *(ASPLOS '22)*. Association for Computing Machinery, New York, NY, USA, 70–83. doi:10.1145/3503222.3507774

[59] Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, Todor Mihaylov, Myle Ott, Sam Shleifer, Kurt Shuster, Daniel Simig, Punit Singh Koura, Anjali Sridhar, Tianlu Wang, and Luke Zettlemoyer. 2022. OPT: Open Pre-trained Transformer Language Models. arXiv:2205.01068 [cs.CL] https://arxiv.org/abs/2205.01068

[60] Tianyi Zhang, Jonah Yi, Bowen Yao, Zhaozhuo Xu, and Anshumali Shrivastava. 2024. NoMAD-Attention: Efficient LLM Inference on CPUs Through Multiply-add-free Attention. In *Advances in Neural Information Processing Systems*, A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (Eds.), Vol. 37. Curran Associates, Inc., 112706–112730. https://proceedings.neurips.cc/paper_files/paper/2024/file/ccda3c632cc8590ee60ca5ba226a4c30-Paper-Conference.pdf

[61] Yilong Zhao, Mingyu Gao, Fangxin Liu, Yiwei Hu, Zongwu Wang, Han Lin, Ji Li, He Xian, Hanlin Dong, Tao Yang, Naifeng Jing, Xiaoyao Liang, and Li Jiang. 2024. UM-PIM: DRAM-based PIM with Uniform Shared Memory Space. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*. 644–659. doi:10.1109/ISCA59077.2024.00053