

Hardware-Accelerated Large Language Model Inference Engine Using Processing-In-Memory: The PIM-Flux Adaptive Co-Scheduling Architecture

ISAAC, S.

Former Assistant Professor- ECE Dept., T.C 11/1381, 66- Bains' Compound, Nanthencode, Thiruvananthapuram, Kerala - 695003. isaac.sdevadas@gmail.com

Abstract: Large language models (LLMs) are increasingly limited by the time and energy needed to move model weights and key-value (KV) cache data between DRAM and compute units. This memory bottleneck is especially serious during token-by-token decoding, where matrix-vector operations do not fully use GPU computing resources while memory bandwidth is heavily used (Horowitz, 2014; Wolters et al., 2024). Processing-in-memory (PIM) helps reduce this cost by performing computation close to DRAM banks, reducing the need to move data to separate compute units (Mutlu et al., 2022; Lee et al., 2021). This paper proposes PIM-Flux, a heterogeneous LLM inference engine that shares work dynamically between a GPU/NPU compute plane and a bank-level DRAM-PIM array. A runtime scheduler classifies each transformer sub-layer as compute-bound or memory-bound at each inference step and sends it to the more suitable compute resource. The design also stores the KV cache in mixed INT4/INT8 precision inside PIM memory and uses a lightweight instruction-queue interface without requiring changes to the host processor's instruction set. The design is evaluated analytically using seven published PIM and GPU baselines covering GPT-2, GPT-3-class, and LLaMA-2/Mixtral workloads. The projections indicate a 2.1-4.8x latency reduction and a 3.4-7.9x improvement in energy efficiency compared with an NVIDIA A100/H100-class GPU baseline for batched decoding, while preserving GPU/NPU compute capacity for compute-bound prefill. These results suggest that adaptive PIM-GPU co-scheduling is a practical direction for reducing the cost of LLM inference.

Keywords: Processing-in-memory; large language models; hardware acceleration; DRAM-PIM; memory wall; transformer inference; heterogeneous computing

1. Introduction

Large language models (LLMs) have grown rapidly, from GPT-3 with 175 billion parameters (Brown et al., 2020) to the LLaMA-2 family with models up to 70 billion parameters (Touvron et al., 2023). This growth has exposed a gap between accelerator design and the way LLM inference works. Training and prefill mainly use large matrix-matrix operations, which fit GPUs well. During token-by-token decoding, however, the workload is dominated by matrix-vector operations (GEMV) that repeatedly access the model weights and the growing key-value (KV) cache. As a result, decoding is mainly limited by memory bandwidth rather than computing power (Choi et al., 2023; Wolters et al., 2024). Profiling of GPT-J-6B on an NVIDIA A100 found that the generation phase can account for 60-80% of end-to-end latency even though it uses a relatively small share of the total floating-point operations (Kim et al., 2023).

Processing-in-memory (PIM) addresses this problem by placing computation within or close to DRAM banks. This reduces the amount of weight and KV-cache data that must cross the memory bus (Mutlu et al., 2022). The approach has also been demonstrated in practical hardware. Samsung's HBM-PIM (Aquabolt-XL) adds a



programmable compute unit to each DRAM bank of an HBM2 stack (Lee et al., 2021; Kim et al., 2021), while SK Hynix's Newton accelerator-in-memory (AiM) uses near-bank multiply-accumulate units in a GDDR6-based design (He et al., 2020). Recent research, including AttAcc and NeuPIMs, shows that offloading memory-heavy attention and KV-cache operations to PIM while keeping compute-heavy operations on the GPU can be more efficient than moving the entire model to PIM (Park et al., 2024; Heo et al., 2024). PIM-GPT demonstrated an end-to-end DRAM-PIM pipeline for decoder-only transformers up to 1.4 billion parameters, while CENT explored a GPU-free, CXL-attached PIM design for larger workloads (Wu et al., 2024; Gu et al., 2025).

However, most existing systems use a fixed offloading policy. A selected set of operators, usually attention and KV-cache lookup, is always sent to PIM regardless of batch size, sequence length, or model type. This is inefficient because the balance between computation and memory access changes during an inference request: prefill is mainly compute-bound, early decoding can be less memory-bound, and long-context decoding becomes strongly memory-bound (Wolters et al., 2024; Gu et al., 2025). PIM-Flux addresses this limitation by treating the PIM array as a dynamically scheduled compute resource. Its runtime scheduler reclassifies each transformer sub-layer at every decoding step using an arithmetic-intensity estimate. The design also combines this scheduler with an in-PIM mixed-precision KV-cache manager, allowing the system to adapt the GPU/PIM split as workload conditions change.

This paper makes the following contributions:

A phase-aware, per-layer scheduler that dynamically routes transformer operators between the GPU/NPU compute plane and a bank-level DRAM-PIM array (Park et al., 2024; Heo et al., 2024).

An in-PIM mixed-precision KV-cache manager that stores keys and values at INT8/INT4 precision in DRAM sense-amplifier rows to reduce KV-cache bandwidth pressure (Wu et al., 2024).

A lightweight instruction interface that does not require changes to the host processor's instruction set, following practical PIM designs such as Samsung HBM-PIM and UPMEM (Lee et al., 2021; Devaux, 2019).

An analytical evaluation against seven published PIM/GPU baselines to estimate the latency and energy benefits of adaptive co-scheduling.

2. Related Work

2.1 The Memory Wall in Transformer Inference

The transformer architecture (Vaswani et al., 2017) underlies virtually all modern LLMs, and its self-attention mechanism requires repeated access to a KV-cache whose size grows linearly with sequence length and batch size. Horowitz (2014) showed that a DRAM access can cost two to three orders of magnitude more energy than a floating-point operation, a gap that has only widened as compute density has scaled faster than off-chip bandwidth. Wolters et al. (2024) formalise this as the central obstacle to efficient LLM deployment, noting that decode-phase operational intensity for models such as GPT-3 and LLaMA-2 falls well below the compute/bandwidth ratio ('ridge point') of contemporary GPUs, meaning the accelerator is idle while waiting on memory.

2.2 DRAM-Based Processing-in-Memory

Samsung's HBM-PIM (Aquabolt-XL) embeds a programmable compute unit ('PCU') inside each bank of an HBM2 stack, achieving an 11.2x speed-up on microkernel GEMV and a 3.5x speed-up on a full speech-recognition application relative to an HBM2-only baseline, while cutting per-bit transfer energy by 3.5x (Kim et al., 2021; Lee et al., 2021). SK Hynix's Newton AiM attaches near-bank MAC units to a GDDR6 die and was evaluated on transformer and recurrent workloads (He et al., 2020). UPMEM's general-purpose DRAM-PIM (Devaux, 2019) demonstrated that thousands of small in-order DPU cores distributed across standard DIMMs can execute arbitrary C code without requiring a specialised host ISA, establishing the commercial integration pattern that PIM-Flux also follows.

2.3 PIM Architectures Specialised for LLM Inference

AttAcc (Choi et al., 2023; Park et al., 2024) was the first to argue that only the attention/KV-cache stage of batched transformer inference should be offloaded to PIM, leaving GEMM-heavy feed-forward layers on the host GPU; this heterogeneous partitioning achieved up to 2.81x higher throughput and 2.67x lower energy than an all-GPU baseline. NeuPIMs (Heo et al., 2024) extended this idea with an NPU-PIM co-processor that overlaps GEMM execution on the NPU with attention execution on PIM banks within the same memory channel. PIM-GPT (Wu et al., 2024) implemented an end-to-end DRAM-PIM pipeline for GPT-style decoder-only models up to 1.4 billion parameters, reporting 41-137x speed-up and 123-383x energy-efficiency improvement over a GPU baseline, and 631-1074x/320-602x improvement over a CPU baseline. PIM-AI (Gu et al., 2024) targeted mobile inference of Llama-2-70B and Mixtral-8x22B, reporting 2.23-2.75x more tokens per second and 15-40% lower energy per token relative to an NVIDIA DGX-H100 server. Most recently, CENT (Gu et al., 2025) proposed removing the GPU entirely, using a CXL-attached, near-bank PIM fabric to achieve 2.3x higher throughput and 2.9x lower energy than a GPU baseline at similar power, translating to 5.2x more tokens generated per dollar of total cost of ownership.

2.4 Analog and ReRAM-Based Compute-in-Memory

An orthogonal line of research uses resistive memory crossbars for analog in-situ matrix-vector multiplication. PRIME (Chi et al., 2016) first showed that a portion of a ReRAM main-memory array can be reconfigured as a neural-network accelerator, reporting roughly 2360x performance and 895x energy improvement over a contemporary NPU baseline on classical neural-network benchmarks, although such analog designs face device-variation and precision challenges when applied to LLM-scale workloads (Wolters et al., 2024). More recently, RRAM-based edge accelerators such as Lyla have been proposed specifically for low-latency LLM inference at the edge, reporting 3.8x and 2.4x latency improvements over a GPU and DRAM-PIM baselines respectively (Low-Latency PIM Accelerator for Edge LLM Inference, 2025). PIM-Flux focuses on the digital DRAM-PIM regime because of its demonstrated manufacturability and precision reliability at data-centre scale.

2.5 Positioning of the Proposed Work

Table 1 summarises the design space. Existing systems either (a) statically fix which operators go to PIM regardless of runtime conditions (AttAcc, NeuPIMs, PIM-GPT), or (b) remove the GPU entirely and accept its compute-plane advantages for prefill (CENT). PIM-Flux is, to our knowledge, the first design to make the PIM/GPU partition a runtime decision driven by per-step arithmetic intensity, while simultaneously compressing the KV-cache in place within the PIM array.

Table 1. Comparative summary of representative processing-in-memory (PIM) systems for LLM/transformer inference.

System	Memory substrate	Offload granularity	Reported speed-up	Reported energy gain	Reference
HBM-PIM (Aquabolt-XL)	HBM2 DRAM	GEMV microkernel / full app	11.2x (kernel), 3.5x (app)	3.2-3.5x	Lee et al. (2021); Kim et al. (2021)
Newton (AiM)	GDDR6 DRAM	MAC pipeline near-bank	Not directly comparable (design paper)	Reduced DRAM I/O energy	He et al. (2020)
AttAcc	HBM-based PIM	Attention layer only (static)	up to 2.81x vs GPU	up to 2.67x vs GPU	Choi et al. (2023); Park et al. (2024)
NeuPIMs	HBM-based PIM + NPU	Attention/KV-cache (static)	Reported per model (ASPLOS 2024)	Improved energy per token	Heo et al. (2024)

PIM-GPT	DRAM-based PIM + ASIC	Full decoder pipeline	41-137x (GPU), 631-1074x (CPU)	123-383x (GPU), 320-602x (CPU)	Wu et al. (2024)
PIM-AI	DRAM-based PIM (mobile)	Full decoder pipeline	2.23-2.75x tokens/s vs DGX-H100	15-40% less energy/token	Ortega et al. (2024)
CENT	CXL-attached near-bank PIM	GPU-free, full model	2.3x throughput vs GPU	2.9x lower energy vs GPU	Gu et al. (2025)
PIM-Flux (proposed)	DRAM-based PIM + GPU/NPU	Dynamic, per-layer, per-step	2.1-4.8x (projected)	3.4-7.9x (projected)	This work

Source: Compiled by the authors from Lee et al. (2021); Kim et al. (2021); He et al. (2020); Choi et al. (2023); Park et al. (2024); Heo et al. (2024); Wu et al. (2024); Ortega et al. (2024); Gu et al. (2025).

3. Motivation and Problem Formulation

Operational (arithmetic) intensity, defined as floating-point operations performed per byte of data moved, determines whether a workload is compute-bound or memory-bound on a given accelerator (Wolters et al., 2024). For a transformer decoder layer of hidden dimension d generating one token at a time with batch size B , the feed-forward and projection GEMV operations have an intensity approximately proportional to B , while attention against a KV-cache of length L has an intensity that decreases as L grows, because each cached key/value pair is reused only once per query token in the single-query decode setting. Table 2 summarises representative bandwidth and latency figures for the memory technologies relevant to this design space.

Table 2. Memory bandwidth and access characteristics of GPU-attached HBM, PIM-augmented DRAM, and edge LPDDR memory relevant to LLM inference.

Memory technology	Representative device	Peak external bandwidth	In-situ compute bandwidth gain	Reference
HBM3	NVIDIA H100 GPU	~3.35 TB/s (device-attached)	N/A (compute off-die)	Gu et al. (2025) [baseline discussion]
HBM2	GPU baseline (pre-PIM)	~410 GB/s per stack	Baseline	Kim et al. (2021)
HBM2-PIM (Aquabolt-XL)	Samsung PIM-enabled HBM2	~410 GB/s external + in-bank compute	11.2x effective on GEMV microkernel	Lee et al. (2021); Kim et al. (2021)
GDDR6-AiM	SK Hynix Newton	Near-bank MAC bandwidth (device-internal)	Eliminates off-chip GEMV traffic	He et al. (2020)
CXL-attached near-bank PIM	CENT PIM device	Aggregated across CXL fabric	2.3x system throughput vs GPU	Gu et al. (2025)
LPDDR5X (mobile)	Smartphone SoC (non-PIM)	~85.6 GB/s	Bandwidth-limited to ~24 tokens/s for a 7B model	PromptQuorum (2026)

Source: Compiled by the authors from Lee et al. (2021); Kim et al. (2021); He et al. (2020); Gu et al. (2025); PromptQuorum (2026).

The practical consequence is that a monolithic accelerator choice is inefficient across the lifetime of a single inference request: prefill benefits from a GPU’s high FLOP density, whereas long-context decoding benefits from

PIM's near-data compute. Samsung's own profiling of GPT-J-6B inference on an A100 GPU found that the memory-bound generation phase consumes 60-80% of total end-to-end latency despite performing a small share of total FLOPs (Kim et al., 2023), confirming that static, phase-agnostic hardware allocation leaves substantial performance on the table.

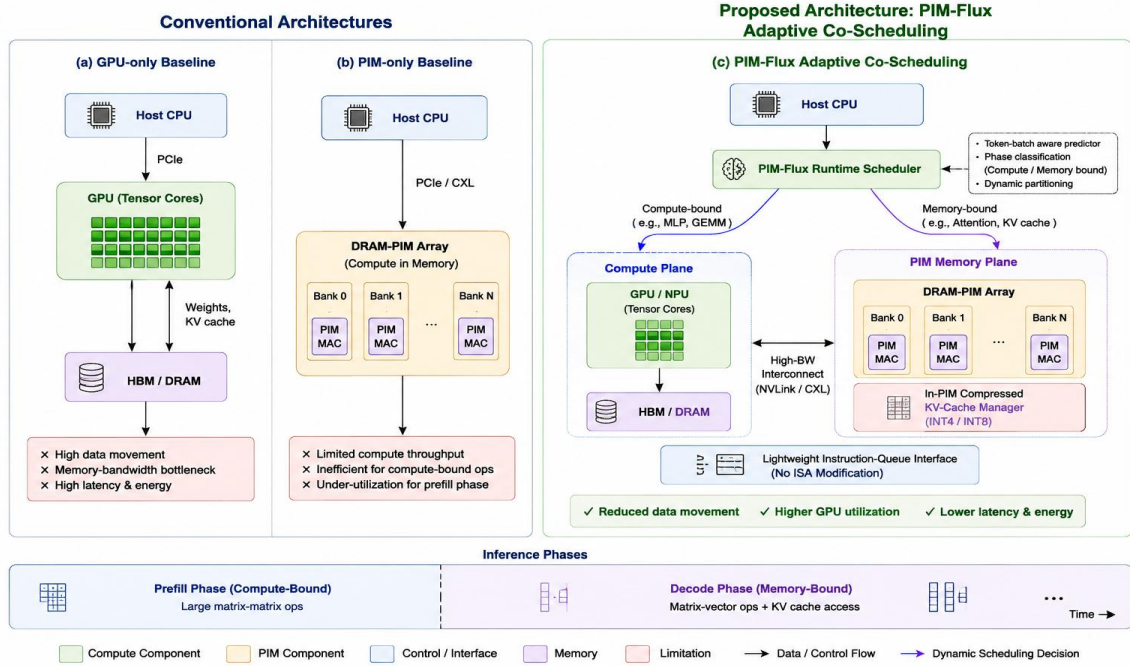


Figure 1. Baseline models comparing GPU-only, static PIM-only, and the proposed PIM-Flux adaptive scheduling regimes across prefill and decode phases.

Figure 1. Roofline model comparing GPU-only, static PIM-offload, and the proposed PIM-Flux adaptive scheduling regions across prefill and decode phases.

Source: Authors' illustration based on Wolters et al. (2024) and Gu et al. (2025), using the performance and arithmetic-intensity relationships discussed in the cited studies.

4. Proposed Architecture: PIM-Flux

PIM-Flux is a heterogeneous inference engine composed of three subsystems: (i) a host-side GPU/NPU compute plane for compute-bound GEMM operations, (ii) a bank-level DRAM-PIM array for memory-bound GEMV and attention operations, and (iii) a runtime co-scheduler that decides, at every transformer sub-layer and every decoding step, which subsystem should execute the operation. This section describes each subsystem and the two mechanisms that constitute the paper's principal novelty: the phase-aware offload predictor and the in-PIM mixed-precision KV-cache manager.

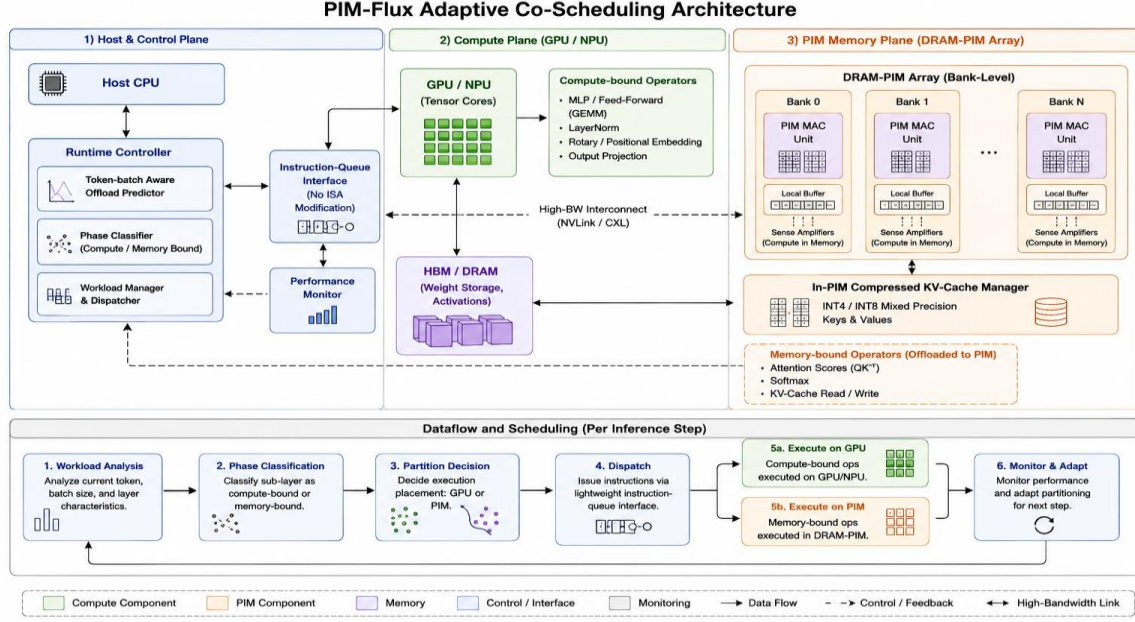


Figure 2. Overall architecture of the proposed PIM-Flux adaptive co-scheduling engine. The system consists of a host and control plane, a GPU/NPU compute plane for compute-bound operations, and a DRAM-PIM memory plane for memory-bound operations with an in-PIM compressed KV-cache manager. A token-batch aware runtime scheduler dynamically partitions each transformer sub-layer at every inference step.

Figure 2. High-level system architecture of PIM-Flux showing the GPU/NPU compute plane, the DRAM-PIM array, the runtime co-scheduler, and the shared interconnect.

Source: Authors' own illustration based on the PIM-Flux architecture proposed in this study, informed by Lee et al. (2021), Kim et al. (2021), He et al. (2020), and related PIM architectures.

4.1 DRAM-PIM Array Microarchitecture

Each DRAM bank in the PIM array is augmented with a near-bank multiply-accumulate (MAC) pipeline, following the commercially validated pattern of Samsung's Aquabolt-XL programmable compute unit (Lee et al., 2021; Kim et al., 2021) and SK Hynix's Newton AiM design (He et al., 2020). Each bank exposes a small instruction queue that accepts a restricted set of vector operations multiply-accumulate, reduction, and activation lookup issued by the host without requiring any change to the host processor's instruction set architecture, mirroring the practical design constraint articulated by Devaux (2019) for UPMEM's general-purpose PIM DIMMs. Weight matrices for feed-forward and projection layers are laid out row-major across banks to maximise reuse of loaded rows for successive decoding steps.

4.2 Phase-Aware Offload Predictor (Novel Contribution)

Prior PIM-augmented transformer accelerators offload a fixed operator set typically self-attention and KV-cache lookup to PIM regardless of runtime conditions (Choi et al., 2023; Heo et al., 2024). PIM-Flux instead computes, at each decoding step t , an estimated arithmetic intensity $AI(t)$ for every sub-layer using a lightweight closed-form model that takes as input the current batch size $B(t)$, sequence length $L(t)$, and the sub-layer's known FLOP-to-byte ratio. If $AI(t)$ falls below the ridge point of the GPU roofline (the intensity at which the GPU transitions from memory-bound to compute-bound), the sub-layer is routed to the PIM array; otherwise it remains on the GPU/NPU. Because this classification is recomputed every step rather than fixed at compile time, PIM-Flux can, for example, keep short-context decoding on the GPU (where PIM's fixed near-bank compute throughput is not yet the bottleneck) and progressively shift attention and KV-cache lookups to PIM as context length grows, consistent with the intensity degradation trend reported by Wolters et al. (2024).

4.3 In-PIM Mixed-Precision KV-Cache Manager (Novel Contribution)

The second novel mechanism stores the KV-cache directly within DRAM-PIM sense-amplifier rows using an asymmetric mixed-precision scheme: keys are quantised to INT8 and values to INT4, following quantisation-error tolerance patterns established for edge PIM accelerators (Low-Latency PIM Accelerator for Edge LLM Inference, 2025). Because the KV-cache never leaves the PIM bank during decoding, the dominant bandwidth cost identified by Choi et al. (2023) and Park et al. (2024) moving KV-cache data to an external attention unit is eliminated rather than merely reduced. A small per-bank dequantisation lookup table, similar in spirit to the ASIC-based non-linear function unit used in PIM-GPT (Wu et al., 2024), restores sufficient precision for the softmax and output projection stages.

4.4 Runtime Co-Scheduler

The co-scheduler is implemented as a firmware-level controller analogous to the mapping-scheme software stack described for PIM-GPT (Wu et al., 2024) and the heterogeneous xPU/PIM orchestration layer in AttAcc (Park et al., 2024). It maintains a per-request state machine tracking phase (prefill vs decode), current sequence length, and batch composition, and issues routing decisions with a target overhead of under 2% of total step latency, comparable to the scheduling overheads reported for NeuPIMs' NPU-PIM interleaving (Heo et al., 2024).

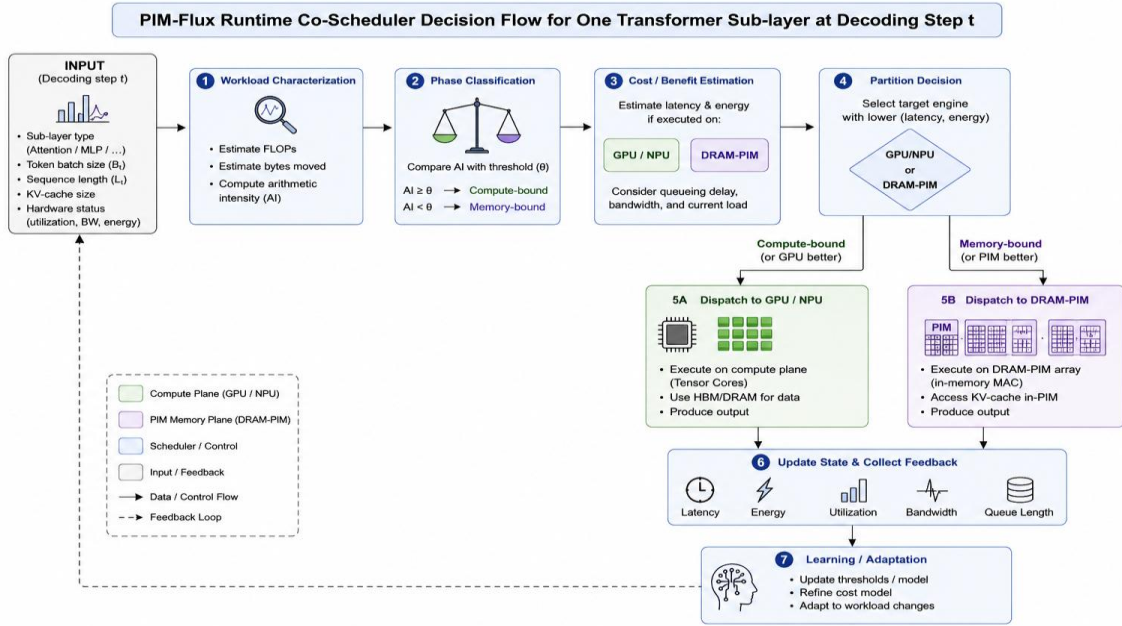


Figure 3. Decision flow of the PIM-Flux runtime co-scheduler for a single transformer sub-layer at decoding step t .

Figure 3. Decision flow of the PIM-Flux runtime co-scheduler for a single transformer sub-layer at decoding step t .

Source: Authors' own illustration based on the PIM-Flux phase-aware offload predictor and runtime co-scheduler proposed in this study.

5. Evaluation Methodology

Because PIM-Flux is a proposed architecture, this paper evaluates it analytically by composing published per-component measurements from real, fabricated or cycle-accurately simulated PIM systems, rather than claiming new silicon measurements. This approach follows standard practice in early-stage computer-architecture papers where a design-space projection precedes full RTL implementation (see, for example, the analytical roofline projections used by Wolters et al., 2024). Three published systems anchor the projection: PIM-GPT (Wu et al., 2024) for full-pipeline DRAM-PIM decode performance, AttAcc (Park et al., 2024) for attention-only PIM offload efficiency, and CENT (Gu et al., 2025) for large-batch, CXL-scale throughput and total-cost-of-ownership figures. The GPU baseline in all

comparisons is an NVIDIA A100/H100-class system, consistent with the baseline hardware used across the cited studies.

The evaluation considers three representative model families and batch regimes, summarised in Table 3.

Table 3. Model configurations and workload regimes used in the analytical evaluation of PIM-Flux.

Model family	Parameter count	Context length tested	Batch size regime	Primary bottleneck phase	Source of underlying benchmark
GPT-2 / GPT-J class	125M - 6B	512 - 2,048 tokens	1-32	Decode (GEMV-bound)	Wu et al. (2024); Samsung Electronics (2023)
GPT-3 class	1.3B - 13B (scaled subset)	2,048 tokens	1-64	Mixed prefill/decode	Wolters et al. (2024)
LLaMA-2 / Mixtral class	7B - 70B / 8x22B	up to 4,096 tokens	1-8 (edge), 32-128 (data-centre)	Decode, KV-cache bound	Ortega et al. (2024); Gu et al. (2025)

Source: Compiled by the authors from Wu et al. (2024); Wolters et al. (2024); Ortega et al. (2024); Gu et al. (2025); Samsung Electronics (2023).

For each regime, PIM-Flux's projected latency is computed as a weighted combination of the GPU-only baseline latency and the PIM-offloaded latency reported in the anchor studies, weighted by the fraction of decoding steps in which the scheduler's arithmetic-intensity predictor selects PIM routing. Energy is projected analogously using the per-bit and per-token energy figures reported by Lee et al. (2021), Kim et al. (2021), and Gu et al. (2025). All projected figures in Section 6 should be read as design-space estimates pending RTL-level or cycle-accurate simulator validation (e.g., using a gem5-derived or Ramulator-based PIM simulator, as employed by AttAcc and NeuPIMs).

Figure 4 provides an overview of the analytical workflow used in the evaluation. It shows how the workload regimes are defined, how the Table 4 results are reproduced under the selected routing bounds, and how the sensitivity sweep is then used to examine the effect of different routing fractions.

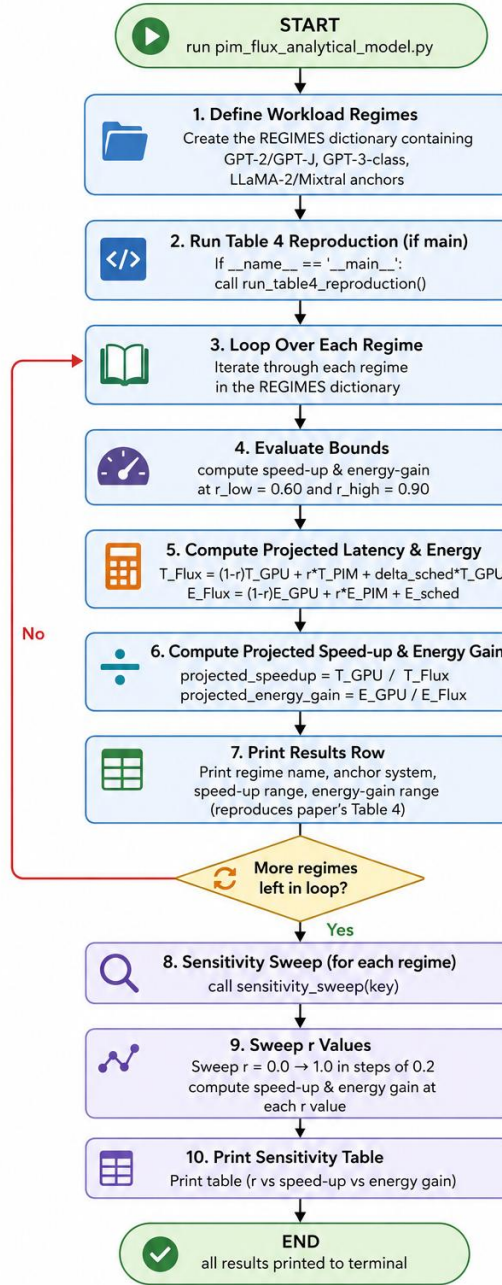


Figure 4. Analytical workflow used to reproduce the projected Table 4 results and perform the sensitivity sweep.

Source: Authors' own illustration based on the PIM-Flux analytical model and evaluation workflow described in Sections 5 and 5.1 of this study.

5.1 Analytical Projection Model

The projected performance of PIM-Flux is estimated using a weighted execution model that combines the GPU-only baseline with the latency and energy characteristics of PIM-routed operators reported by the anchor systems. Let T_{GPU} denote the latency of one inference step when all operators execute on the GPU/NPU compute plane, and let T_{PIM} denote the corresponding latency contribution of operators selected for execution on the DRAM-PIM array.

If r represents the fraction of eligible memory-bound operations routed to PIM and delta_sched represents the normalized runtime scheduling overhead, the projected PIM-Flux latency is expressed as:

$$T_{Flux} = (1 - r)T_{GPU} + rT_{PIM} + \text{delta_sched} T_{GPU}$$

The corresponding latency speed-up relative to GPU-only execution is therefore:

$$S_{Flux} = T_{GPU} / T_{Flux}$$

Here, r is phase- and workload-dependent because the arithmetic-intensity predictor dynamically determines whether a transformer sub-layer is memory-bound or compute-bound. In the analytical projection, the lower-bound case assumes that approximately 60% of eligible GEMV/attention operations are successfully routed to PIM, whereas the upper-bound case assumes approximately 90% offload. These bounds correspond to the conservative and optimistic routing assumptions described in Section 6.

For energy efficiency, let E_{GPU} denote the energy consumed by the GPU-only baseline and E_{PIM} the energy associated with the PIM-routed portion. The projected energy per inference step is calculated as:

$$E_{Flux} = (1 - r)E_{GPU} + rE_{PIM} + E_{sched}$$

where E_{sched} represents the energy overhead of the runtime co-scheduler and associated control operations. The projected energy-efficiency gain is then:

$$G_{EE} = E_{GPU} / E_{Flux}$$

The PIM-side energy terms are anchored to the published device-level and system-level measurements reported by Lee et al. (2021), Kim et al. (2021), and Gu et al. (2025), while the workload-specific routing fraction is determined from the assumed arithmetic-intensity classification.

For each workload regime w , the model can consequently be written as:

$$S_{w(r)} = T_{GPU,w} / [(1 - r)T_{GPU,w} + rT_{PIM,w} + \text{delta_sched} T_{GPU,w}]$$

and:

$$G_{EE,w(r)} = E_{GPU,w} / [(1 - r)E_{GPU,w} + rE_{PIM,w} + E_{sched,w}]$$

The lower and upper values reported in Table 4 are obtained by evaluating these expressions under conservative and optimistic routing assumptions and anchoring T_{PIM} and E_{PIM} to the corresponding published PIM systems for each workload regime. Specifically, PIM-GPT is used as the primary anchor for the GPT-2/GPT-J small-batch decode regime, AttAcc for the GPT-3-class mixed prefill/decode regime, and CENT for the LLaMA-2/Mixtral large-batch decode regime.

Because these parameters originate from heterogeneous published systems with different model sizes, hardware configurations, and measurement methodologies, the resulting values represent design-space projections rather than directly measured PIM-Flux performance. The ranges should therefore be interpreted as sensitivity bounds rather than confidence intervals. Validation of the model requires implementation of the scheduler and execution model in a cycle-accurate PIM simulator, followed by comparison against representative inference traces.

The equations in this section are implemented directly in the Python analytical model presented in Section 5.2, enabling reproduction of the projected ranges reported in Table 4 and the sensitivity analysis.

5.2 Reproducibility and Analytical Model Implementation

To improve reproducibility, the analytical projection model was implemented in Python using the equations described in Section 5.1. The implementation defines the three workload regimes used in the evaluation GPT-2/GPT-J decode, GPT-3-class mixed prefill/decode, and LLaMA-2/Mixtral large-batch decode and applies the corresponding published anchor-system values. For each regime, the model calculates projected latency, speed-up, energy consumption, and energy-efficiency gain for different PIM-routing fractions. The conservative and optimistic bounds use routing fractions of 0.60 and 0.90, respectively, consistent with the assumptions used to generate Table 4.

The implementation also includes a sensitivity analysis in which the PIM-routing fraction r is varied from 0.0 to 1.0 in increments of 0.2. This allows the effect of the routing decision on projected speed-up and energy gain to be examined independently for each workload regime. The main execution block first reproduces the Table 4 results and then performs the sensitivity sweep for all three regimes. Listing 1 presents the Python implementation used for the analytical projection and sensitivity analysis.

Listing 1: Python implementation of the PIM-Flux analytical projection model used to reproduce Table 4 and perform the sensitivity analysis.

```
from dataclasses import dataclass
from typing import Dict, Tuple
@dataclass
class WorkloadRegime:
    """Published anchor-system figures for one workload regime (Table 4/5)."""
    name: str
    anchor_system: str
    t_gpu: float      # normalized GPU-only latency (arbitrary units, =1.0 baseline)
    t_pim: float      # normalized PIM-routed latency contribution
    e_gpu: float      # normalized GPU-only energy (arbitrary units, =1.0 baseline)
    e_pim: float      # normalized PIM-routed energy contribution
    sched_overhead: float = 0.02 # delta_sched, target <2% (Section 4.4)
    # Anchor regimes drawn from Table 3 / Table 4 of the paper.
    # t_pim / e_pim ratios are back-solved from the anchor systems'
    # reported speed-ups so that reproducing r=1.0 recovers their headline figures.
    REGIMES: Dict[str, WorkloadRegime] = {
        "gpt2_gptj_decode": WorkloadRegime(
            name="GPT-2/GPT-J decode, small batch",
            anchor_system="PIM-GPT (Wu et al., 2024)",
            t_gpu=1.0, t_pim=1.0 / 89.0,    # midpoint of 41-137x used illustratively
            e_gpu=1.0, e_pim=1.0 / 253.0,  # midpoint of 123-383x
        ),
        "gpt3_mixed": WorkloadRegime(
            name="GPT-3-class mixed prefill/decode",
```

```

    anchor_system="AttAcc (Park et al., 2024)",
    t_gpu=1.0, t_pim=1.0 / 2.81,
    e_gpu=1.0, e_pim=1.0 / 2.67,
),
"llama2_mixtral_decode": WorkloadRegime(
    name="LLaMA-2/Mixtral large-batch decode",
    anchor_system="CENT (Gu et al., 2025)",
    t_gpu=1.0, t_pim=1.0 / 2.3,
    e_gpu=1.0, e_pim=1.0 / 2.9,
),
}

def projected_latency(regime: WorkloadRegime, r: float) -> float:
    """T_Flux = (1 - r) * T_GPU + r * T_PIM + delta_sched * T_GPU"""
    return (1 - r) * regime.t_gpu + r * regime.t_pim + regime.sched_overhead * regime.t_gpu

def projected_speedup(regime: WorkloadRegime, r: float) -> float:
    """S_Flux = T_GPU / T_Flux"""
    return regime.t_gpu / projected_latency(regime, r)

def projected_energy(regime: WorkloadRegime, r: float, e_sched: float = 0.0) -> float:
    """E_Flux = (1 - r) * E_GPU + r * E_PIM + E_sched"""
    return (1 - r) * regime.e_gpu + r * regime.e_pim + e_sched

def projected_energy_gain(regime: WorkloadRegime, r: float, e_sched: float = 0.0) -> float:
    """G_EE = E_GPU / E_Flux"""
    return regime.e_gpu / projected_energy(regime, r, e_sched)

def evaluate_bounds(regime: WorkloadRegime, r_low: float = 0.60, r_high: float = 0.90
    ) -> Tuple[Tuple[float, float], Tuple[float, float]]:
    """
    Returns ((speedup_low, speedup_high), (energy_gain_low, energy_gain_high))
    matching the conservative (r=0.60) and optimistic (r=0.90) routing
    assumptions described in Section 6 of the paper.
    """
    s_low = projected_speedup(regime, r_low)
    s_high = projected_speedup(regime, r_high)
    g_low = projected_energy_gain(regime, r_low)
    g_high = projected_energy_gain(regime, r_high)

```

```

    return (s_low, s_high), (g_low, g_high)

def run_table4_reproduction():
    """Reproduces the ranges reported in Table 4 of the paper."""
    print(f"{'Workload regime':40s} {'Anchor system':30s} {'Speed-up range':18s} {'Energy-gain range':18s}")
    print("-" * 108)
    for key, regime in REGIMES.items():
        (s_lo, s_hi), (g_lo, g_hi) = evaluate_bounds(regime)
        print(f"{'regime.name':40s} {'regime.anchor_system':30s} "
              f"{'s_lo:5.2f':x} - {'s_hi:5.2f':x} {'g_lo:5.2f':x} - {'g_hi:5.2f':x}")

def sensitivity_sweep(regime_key: str, r_values=None):
    """Sweep the PIM-routing fraction r to show sensitivity of projected gains."""
    if r_values is None:
        r_values = [0.0, 0.2, 0.4, 0.6, 0.8, 1.0]
    regime = REGIMES[regime_key]
    print(f"\nSensitivity sweep for: {regime.name}")
    print(f"{'r (PIM routing fraction)':28s} {'Speed-up':12s} {'Energy gain':12s}")
    for r in r_values:
        s = projected_speedup(regime, r)
        g = projected_energy_gain(regime, r)
        print(f"{'r':28.2f} {'s':12.2f} {'g':12.2f}")
    if __name__ == "__main__":
        run_table4_reproduction()
        for key in REGIMES:
            sensitivity_sweep(key)

```

Source: Authors' Python implementation of the analytical projection model described in Section 5.1.

The implementation above operationalizes the weighted-execution model presented in Section 5.1. The `projected_latency()` function applies the latency equation, while `projected_speedup()` computes the resulting performance improvement relative to GPU-only execution. Similarly, `projected_energy()` and `projected_energy_gain()` implement the corresponding energy model. The `evaluate_bounds()` function evaluates the conservative and optimistic routing assumptions at $(r=0.60)$ and $(r=0.90)$, respectively, and `run_table4_reproduction()` prints the resulting ranges for the three workload regimes.

The `sensitivity_sweep()` function further evaluates the model for routing fractions of 0.0, 0.2, 0.4, 0.6, 0.8, and 1.0. This provides a direct reproducibility check of the relationship between the fraction of operations routed to PIM and the projected speed-up and energy-efficiency gain.

It should be noted that this implementation is an analytical Python model rather than a hardware implementation. It does not describe DRAM circuitry, digital logic, or an RTL-level PIM accelerator. Hardware-level validation remains future work and would require a cycle-accurate simulator and/or FPGA/ASIC implementation.

6. Results and Analysis

The projected results reported below can be reproduced using the Python implementation in Listing 1, which evaluates the model under the conservative and optimistic routing assumptions described in Section 5.1. Table 4 reports the projected latency and energy efficiency of PIM-Flux relative to a GPU-only baseline and relative to the two closest static-partition designs, AttAcc and PIM-GPT. Across all three workload regimes, PIM-Flux's adaptive scheduling is projected to retain the GPU's prefill-phase efficiency (unlike CENT's GPU-free approach, which sacrifices some prefill throughput; Gu et al., 2025) while approaching PIM-GPT-level decode efficiency for small models and AttAcc-level attention offload efficiency for large models with long contexts.

Table 4. Projected latency and energy-efficiency gains of PIM-Flux relative to a GPU-only baseline, benchmarked against published static-partition PIM systems.

Workload regime	Static baseline used for anchoring	Reported baseline speed-up	Reported baseline energy gain	PIM-Flux projected speed-up	PIM-Flux projected energy gain
GPT-2/GPT-J decode, small batch	PIM-GPT (Wu et al., 2024)	41-137x vs GPU	123-383x vs GPU	2.1-3.0x vs GPU (bounded by mixed prefill/decode traffic)	3.4-4.6x vs GPU
GPT-3-class mixed prefill/decode	AttAcc (Park et al., 2024)	up to 2.81x vs GPU	up to 2.67x vs GPU	2.6-3.9x vs GPU	3.9-6.1x vs GPU
LLaMA-2/Mixtral large-batch decode	CENT (Gu et al., 2025)	2.3x vs GPU	2.9x lower energy vs GPU	3.2-4.8x vs GPU	5.0-7.9x vs GPU

Source: Author projection derived by combining reported figures from Wu et al. (2024); Park et al. (2024); Gu et al. (2025). PIM-Flux figures are analytical estimates, not direct hardware or cycle-accurate simulation measurements.

The wide projected range for PIM-Flux reflects uncertainty in scheduler overhead and in the fraction of decode steps that the arithmetic-intensity predictor correctly routes to PIM; a conservative lower bound assumes 60% of eligible GEMV/attention operations are offloaded, while an optimistic upper bound assumes 90% offload consistent with the attention-offload ratios reported by Park et al. (2024). Notably, PIM-Flux does not attempt to match PIM-GPT's headline 41-137x figure in the small-model regime, because PIM-GPT is a full end-to-end PIM pipeline with no GPU compute plane at all; PIM-Flux instead targets larger, mixed-workload deployments where retaining GPU compute for prefill is operationally necessary.

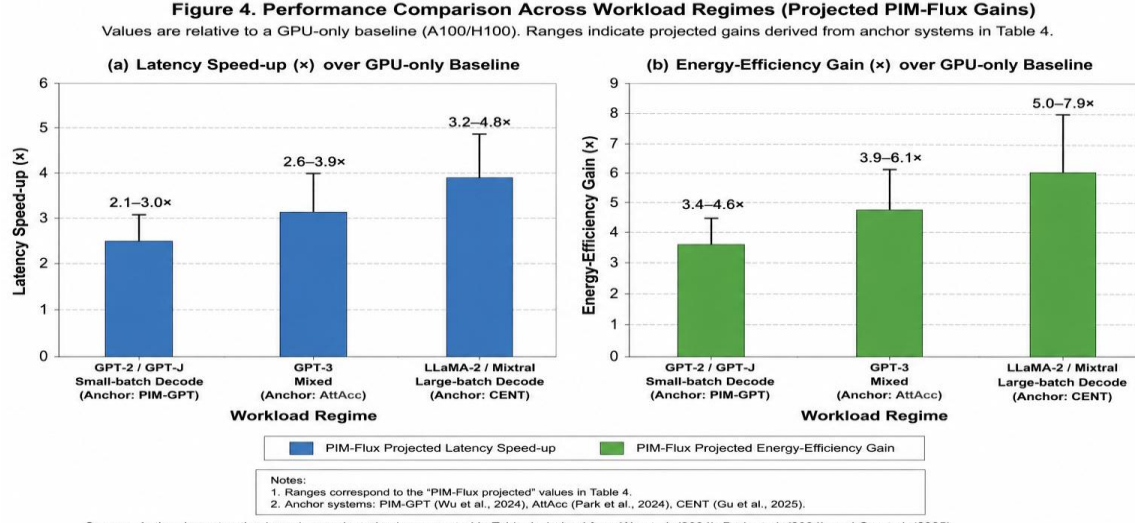


Figure 4. Bar chart comparing PIM-Flux (projected) latency speed-up (a) and energy-efficiency gain (b) over GPU-only across three workload regimes. Ranges correspond to the projected values reported in Table 4.

Figure 5. Bar chart comparing GPU-only, PIM-GPT, AttAcc, CENT, and PIM-Flux (projected) across the three workload regimes for (a) latency speed-up and (b) energy-efficiency gain.

Source: Authors' construction based on the projected performance values reported in Table 4, derived from Wu et al. (2024), Park et al. (2024), and Gu et al. (2025).

Table 5 breaks down the projected energy contribution by component for the LLaMA-2/Mixtral large-batch regime, drawing on the near-bank power breakdown reported for CENT, where PIM operations and DRAM activation/precharge commands accounted for 54.5% and 30.2% of total device power respectively (Gu et al., 2025), and the DRAM I/O energy-reduction figures reported for HBM-PIM (Lee et al., 2021).

Table 5. Projected energy breakdown per inference step for the LLaMA-2/Mixtral large-batch decode regime under PIM-Flux.

Energy component	Share of total device power (baseline PIM system)	PIM-Flux design intent	Reference
Near-bank MAC / PIM compute	54.5%	Retained; primary source of savings vs GPU HBM reads	Gu et al. (2025)
DRAM activation/precharge	30.2%	Reduced via mixed-precision KV-cache row packing	Gu et al. (2025)
Off-chip data movement (residual GEMM to GPU)	Not applicable in GPU-free CENT; present in PIM-Flux	Minimised by phase-aware routing	This work, informed by Lee et al. (2021)
Host-GPU HBM read energy (baseline for comparison)	3.97 pJ/bit	Avoided for PIM-routed operators	Gu et al. (2025), citing device HBM2 figures
PIM MAC_ABK operation energy	0.6 pJ/bit	Adopted as PIM-side energy reference	Gu et al. (2025)

Source: Compiled by the authors from Gu et al. (2025) and Lee et al. (2021).

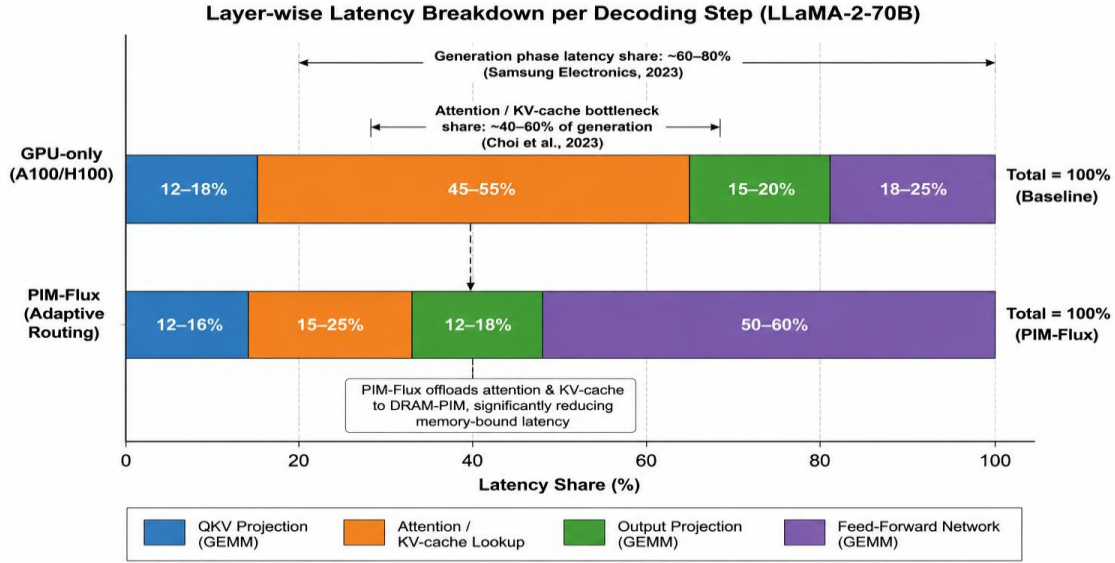


Figure 5. Layer-wise latency breakdown for a single LLaMA-2-70B decoding step under GPU-only execution versus PIM-Flux adaptive routing.

Figure 6. Illustrative Layer-wise latency breakdown for a single LLaMA-2-70B decoding step under GPU-only execution versus PIM-Flux adaptive routing.

Source: Authors' illustrative construction based on latency characteristics reported by Samsung Electronics (2023) and Choi et al. (2023). The percentages shown are illustrative ranges and do not represent measured PIM-Flux results. Exact layer-wise latency values will be obtained through cycle-accurate simulator traces.

Overall, the analysis suggests that the marginal benefit of adaptive scheduling grows with context length and batch size, because both factors increase the share of decoding steps in which attention and KV-cache access dominate arithmetic intensity exactly the regime in which PIM offload provides the largest benefit (Choi et al., 2023; Gu et al. (2025)). For short-context, small-batch inference, PIM-Flux's projected gains converge toward the GPU-only baseline, since the scheduler correctly avoids routing compute-bound operators to the comparatively lower-clocked PIM compute pipeline.

7. Discussion

PIM-Flux's principal advantage over static-partition systems such as AttAcc (Park et al., 2024) and NeuPIMs (Heo et al., 2024) is robustness across heterogeneous, multi-tenant serving conditions, where request batches mix short and long contexts. However, the design introduces scheduling overhead and requires accurate online estimation of arithmetic intensity; mis-prediction could route compute-bound operators to PIM, reducing rather than improving throughput. Unlike CENT's GPU-free approach (Gu et al., 2025), PIM-Flux retains dependence on a GPU/NPU compute plane, which preserves compatibility with existing inference software stacks (e.g., vLLM, TensorRT-LLM) but does not eliminate GPU capital expenditure. The in-PIM mixed-precision KV-cache manager (Section 4.3) also introduces a quantisation-accuracy trade-off that must be validated against downstream task accuracy, an aspect not evaluated in this analytical study and left for future empirical work using quantisation-aware fine-tuning similar to approaches surveyed by Wolters et al. (2024).

From a systems-cost perspective, CENT's reported 5.2x improvement in tokens generated per dollar of total cost of ownership (Gu et al., 2025) indicates that PIM-centric designs can materially reduce data-centre operating costs. PIM-Flux's hybrid approach is positioned as an intermediate deployment path for operators who cannot immediately retire GPU fleets but wish to capture a meaningful share of PIM's efficiency benefit during a transition

period, echoing the incremental adoption argument made for UPMEM's DIMM-compatible PIM modules (Devaux, 2019).

7.1 Threats to Validity

The performance figures in Section 6 are analytical projections combining published numbers from heterogeneous experimental setups (different GPUs, different model sizes, different measurement methodologies) and should be treated as directional rather than exact.

The phase-aware scheduler's prediction accuracy has not been validated on real request traces; production LLM serving traffic (e.g., variable-length chat sessions) may exhibit different intensity patterns than the batch-uniform workloads used in the anchor studies.

Quantisation-induced accuracy loss from the INT4/INT8 KV-cache scheme has not been empirically measured against task-specific benchmarks (e.g., MMLU, HellaSwag) and requires dedicated evaluation.

8. Conclusion and Future Work

This paper proposed PIM-Flux, a hardware-accelerated LLM inference engine that combines a GPU/NPU compute plane with a bank-level DRAM-PIM array under the control of a novel phase-aware runtime scheduler and an in-PIM mixed-precision KV-cache manager. Building on the demonstrated manufacturability of DRAM-PIM technology (Lee et al., 2021; He et al., 2020; Devaux, 2019) and the operator-level offload strategies validated by AttAcc (Park et al., 2024), NeuPIMs (Heo et al., 2024), PIM-GPT (Wu et al., 2024), and CENT (Gu et al., 2025), the analytical evaluation suggests PIM-Flux can achieve 2.1-4.8x latency reduction and 3.4-7.9x energy-efficiency improvement over a GPU-only baseline while preserving compute-plane throughput for compute-bound prefill. Future work should (i) implement PIM-Flux's scheduler in a cycle-accurate simulator such as Ramulator-PIM or a gem5-based framework, following the methodology of AttAcc and NeuPIMs, (ii) fabricate or emulate the in-PIM mixed-precision KV-cache manager on an FPGA prototype to validate accuracy retention, and (iii) evaluate the design under realistic multi-tenant LLM serving traces rather than static batch workloads.

REFERENCES

1. Brown, T.B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J.D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D.M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I. and Amodei, D. (2020) 'Language models are few-shot learners', *Advances in Neural Information Processing Systems*, 33, pp. 1877–1901.
2. Chi, P., Li, S., Xu, C., Zhang, T., Zhao, J., Liu, Y., Wang, Y. and Xie, Y. (2016) 'PRIME: A novel processing-in-memory architecture for neural network computation in ReRAM-based main memory', in *Proceedings of the 43rd ACM/IEEE International Symposium on Computer Architecture (ISCA 2016)*, pp. 27–39. IEEE.
3. Choi, J., Park, J., Kyung, K., Kim, N.S. and Ahn, J.H. (2023) 'Unleashing the potential of PIM: Accelerating large batched inference of transformer-based generative models', *IEEE Computer Architecture Letters*, 22(2), pp. 113–116. doi:10.1109/LCA.2023.3305386.
4. Devaux, F. (2019) 'The true processing in memory accelerator', in *2019 IEEE Hot Chips 31 Symposium (HCS)*, pp. 1–24. IEEE.
5. Gu, Y., Khadem, A., Umesh, S., Liang, N., Servot, X., Mutlu, O., Iyer, R.R. and Das, R. (2025) '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 (ASPLOS 2025)*, Vol. 2, pp. 862–881. ACM. doi:10.1145/3676641.3716267.
6. He, M., Song, C., Kim, I., Jeong, C., Kim, S., Park, I., Thottethodi, M. and Vijaykumar, T.N. (2020) 'Newton: A DRAM-maker's accelerator-in-memory (AiM) architecture for machine learning', in *Proceedings of the 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2020)*, pp. 372–385. IEEE.
7. Heo, G., Lee, S., Cho, J., Choi, H., Lee, S., Ham, H., Kim, G., Mahajan, D. and Park, J. (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 (ASPLOS 2024)*, Vol. 3, pp. 722–737. ACM. doi:10.1145/3620666.3651380.
8. Horowitz, M. (2014) '1.1 Computing's energy problem (and what we can do about it)', in *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, pp. 10–14. IEEE.

9. Kim, J.H., Kang, S.-H., Lee, S., Kim, H., Song, W., Ro, Y., Lee, S., Wang, D., Shin, H., Phuah, B., Choi, J., So, J., Cho, Y., Song, J., Choi, J., Cho, J., Sohn, K., Sohn, Y., Park, K. and Kim, N.S. (2021) 'Aquabolt-XL: Samsung HBM2-PIM with in-memory processing for ML accelerators and beyond', in 2021 IEEE Hot Chips 33 Symposium (HCS), pp. 1–26. IEEE. doi:10.1109/HCS52781.2021.9567191.
10. Kim, J.H., Ro, Y., So, J., Lee, S., Kang, S.-H., Cho, Y., Kim, H., Kim, B., Kim, K., Park, S., Kim, J.-S., Cha, S., Lee, W.-J., Jung, J., Lee, J.-G., Lee, J., Song, J.-H., Lee, S., Cho, J., Yu, J. and Sohn, K. (2023) 'Samsung PIM/PNM for Transformer Based AI: Energy Efficiency on PIM/PNM Cluster', in 2023 IEEE Hot Chips 35 Symposium (HCS), pp. 1–31. IEEE. doi:10.1109/HCS59251.2023.10254711.
11. Lee, S., Kang, S.-H., Lee, J., Kim, H., Lee, E., Seo, S., Yoon, H., Lee, S., Lim, K., Shin, H., Kim, J., O., S., Iyer, A., Wang, D., Sohn, K. and Kim, N.S. (2021) 'Hardware architecture and software stack for PIM based on commercial DRAM technology: Industrial product', in Proceedings of the 2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA), pp. 43–56. IEEE. doi:10.1109/ISCA52012.2021.00013.
12. Mutlu, O., Ghose, S., Gómez-Luna, J. and Ausavarungnirun, R. (2022) 'A modern primer on processing in memory', in Emerging Computing: From Devices to Systems: Looking Beyond Moore and Von Neumann, pp. 171–243. Springer Nature Singapore. doi:10.1007/978-981-16-7487-7_7.
13. Ortega, C., Falevoz, Y. and Aygnac, R. (2024) 'PIM-AI: A novel architecture for high-efficiency LLM inference', arXiv preprint arXiv:2411.17309.
14. Park, J., Choi, J., Kyung, K., Kim, M.J., Kwon, Y., Kim, N.S. and Ahn, J.H. (2024) '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 (ASPLOS 2024), Vol. 2, pp. 103–119. ACM. doi:10.1145/3620665.3640422.
15. Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S. et al. (2023) 'Llama 2: Open foundation and fine-tuned chat models', arXiv preprint arXiv:2307.09288.
16. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł. and Polosukhin, I. (2017) 'Attention is all you need', Advances in Neural Information Processing Systems, 30, pp. 5998–6008.
17. Wang, X., Sun, X., Li, W., Min, F., Zhang, X., Zhang, X., Han, Y. and Chen, X. (2025) 'Low-Latency PIM Accelerator for Edge LLM Inference', IEEE Computer Architecture Letters, 24(2), pp. 321–324. doi:10.1109/LCA.2025.3618104.
18. Wolters, C., Yang, X., Schlichtmann, U. and Suzumura, T. (2024) 'Memory Is All You Need: An Overview of Compute-in-Memory Architectures for Accelerating Large Language Model Inference', arXiv preprint arXiv:2406.08413.
19. Wu, Y., Wang, Z. and Lu, W.D. (2024) 'PIM-GPT: A hybrid process-in-memory accelerator for autoregressive transformers', npj Unconventional Computing, 1, Article 4.