NELSSA: A GPU-PNM Heterogeneous System for Mixed-Length LLM Serving via Length-based Request Placement

Sookyung Choi
SK hynix
Icheon, Republic of Korea
sookyung.choi@sk.com

Seungyong Lee
SK hynix
Icheon, Republic of Korea
seungyong.lee@sk.com

Kangkyu Park
SK hynix
Icheon, Republic of Korea
kangkyu.park@sk.com

Yunseo Chun
SK hynix
Icheon, Republic of Korea
yunseo.chun@sk.com

Junseok Lee
SK hynix
Icheon, Republic of Korea
junseok3.lee@sk.com

Hyeongseok Gwak
SK hynix
Icheon, Republic of Korea
hyeongseok1.gwak@sk.com

Myunghyun Rhee
SK hynix
Icheon, Republic of Korea
myunghyun.rhee@sk.com

Euiseok Kim
SK hynix
Icheon, Republic of Korea
euiseok.kim@sk.com

Donguk Moon
SK hynix
Icheon, Republic of Korea
donguk.moon@sk.com

Kwangsik Shin
SK hynix
Icheon, Republic of Korea
kwangsik.shin@sk.com

Guseul Heo
KAIST
Daejeon, Republic of Korea
gsheo@casys.kaist.ac.kr

Youngpyo Joo
SK hynix
Icheon, Republic of Korea
youngpyo.joo@sk.com

Hoshik Kim
SK hynix
Icheon, Republic of Korea
hoshik.kim@sk.com

Jongse Park*
KAIST
Daejeon, Republic of Korea
jspark@casys.kaist.ac.kr

***Abstract*—Modern LLMs and their agentic applications are broadening the range of serving workloads, spanning context lengths from a few hundred tokens to hundreds of thousands. As these requests frequently interleave within the same serving window, LLM serving systems must handle highly heterogeneous *mixed-length* workloads. These workloads expose fundamental inefficiencies in GPU-centric serving architectures, whose throughput depends on large, memory-constrained batches.**

In this paper, we present NELSSA (Processing-NEar-Memory for Long Sequences with Sparse Attention), an LLM serving system that integrates GPUs with real-world Processing-near-Memory (PNM) accelerator devices to efficiently support mixed-length workloads. NELSSA employs length-based request placement to route short-context requests to GPUs and long-context requests to the PNM tier, incorporating runtime migration to accommodate dynamic context growth without recomputation. We prototype NELSSA as an end-to-end system, implementing device-level sparse attention on PNM, GPU decode kernels, and a host-side runtime that orchestrates scheduling and cross-tier memory movement over a CXL-enabled infrastructure with RPC and RDMA support. Across mixed-length LLM workloads, NELSSA improves decode throughput by up to 5.5× in tokens/sec and reduces P99 latency by up to 15× compared to GPU-only baselines. Our end-to-end prototype and experimental results demonstrate how workload-aware heterogeneous composition of GPU and PNM resources can reduce the resource inefficiencies of homogeneous GPU provisioning, suggesting a promising path toward scalable and resource-efficient LLM infrastructures.

*Corresponding author.

***Index Terms*—LLM serving, Processing-near-Memory (PNM), Heterogeneous system, Mixed-length, Request placement.**

I. Introduction

The recent surge of generative AI, initially driven by ChatGPT [1], has evolved toward agentic AI workloads. AI coding assistants such as Claude Code [2] and Codex [3] exemplify this shift, demonstrating how LLMs increasingly function as autonomous agents capable of planning and executing complex tasks [4]–[9]. As these agentic applications expand, LLM serving systems must support significantly more complex and longer-running workloads than in conventional chatbot deployments.

One prominent consequence of this shift is the emergence of *mixed-length* LLM serving workloads. In production traces, request lengths span from a few hundred tokens to well beyond 100K tokens, while short and long requests frequently interleave within the same serving window [10]–[15]. Such coexisting and unpredictable length variation forces modern LLM serving systems to manage heterogeneous workloads under shared resources.

However, GPU-centric LLM serving architectures struggle with mixed-length workloads. Long-context requests consume a substantially large KV cache, quickly exhausting limited GPU memory capacity, which collapses the effective batch

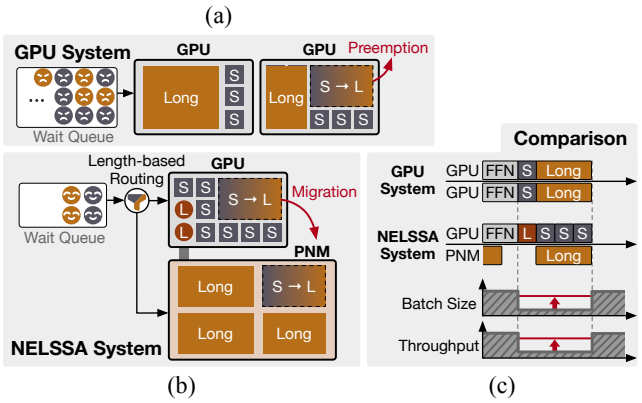


Fig. 1. (a) GPU-only systems share mixed-length requests in a single batch, where long-context executions reduce batch size and induce head-of-line blocking. (b) NELSSA separates requests via length-based placement, assigning short requests to GPUs and long requests to PNM, exploiting runtime migration to handle context growth. (c) This heterogeneity maintains larger batch sizes and improves throughput by mitigating interference from long-context requests.

size and reduces throughput [16]. When co-scheduled with short requests, long executions further induce head-of-line (HoL) blocking, inflating tail latency. Even a small fraction of long-context requests can therefore degrade overall system efficiency in GPU-only deployments.

To overcome these challenges, prior work has pursued two main directions. (1) **KV cache reduction techniques** shrink long-context memory footprints via compression, quantization, or pruning, but inevitably trade accuracy for capacity [17]–[23]. (2) **Memory disaggregation and offloading approaches** move KV caches beyond GPU's device memory to host memory or near-memory devices [24]–[28], alleviating capacity pressure but introducing interconnect bandwidth bottlenecks.

Recently, a line of work has attempted to combine these two directions by leveraging sparse attention on PNM accelerators [29], [30]. By executing sparse attention near large-capacity memory, these designs promise to alleviate GPU memory capacity pressure, while reducing interconnect bandwidth overhead. However, these approaches exhibit fundamental limitations in supporting modern mixed-length LLM serving. Consequently, sparse PNM-based designs face several key challenges:

- **Limitation 1: Long-only optimization.** Existing sparse PNM solutions optimize long-context requests in isolation. In mixed-length scenarios with short and long requests interleaved, naively routing all requests to PNMs can degrade performance for short requests that are more efficiently executed on GPUs.
- **Limitation 2: Lack of support for dynamic context growth.** Most prior approaches assume static request lengths and determine offloading decisions upfront. However, modern agentic LLM workloads often exhibit unpredictable growth in generated context during decoding, rendering static routing insufficient.
- **Limitation 3: Absence of full-stack system integration.** Existing works typically focus on architectural feasibility or algorithmic acceleration without demonstrating an end-to-end system integrating real PNM hardware, host-side coordination, and device-level support required for practical LLM serving.

To tackle these limitations, we build NELSSA, an LLM serving system that integrates GPUs with real-world PNM accelerator devices to efficiently support mixed-length workloads. NELSSA employs length-based request placement with runtime migration to handle dynamic context growth, routing short requests to GPUs and long-context requests to PNMs through hardware–runtime co-design. We prototype NELSSA as an end-to-end system, implementing device-level sparse attention on PNM, GPU decode kernels, and a host-side runtime that orchestrates scheduling and cross-tier memory movement over a CXL-enabled infrastructure with RPC and RDMA support. Figure 1 illustrates the differences between GPU-only systems and NELSSA in handling mixed-length workloads. Our contributions are as follows:

(1) Length-based request placement with hardware-aware routing. We design a routing mechanism that assigns short requests to GPUs and long-context requests to a sparse PNM tier based on a hardware-derived crossover threshold. Unlike heuristic or static partitioning, our approach analytically determines the placement boundary from device-level performance characteristics, enabling each request to execute on the most suitable compute tier. This split-batch hybrid routing maximizes effective GPU batch size, while offloading memory-capacity-bound long requests to PNM, improving throughput and mitigating head-of-line blocking under mixed-length workloads.

(2) Runtime migration for dynamic context growth. We introduce a lightweight migration mechanism that enables requests to transition from GPU to PNM execution as their context length grows during decoding. Rather than relying on recomputation or swap-based eviction when memory pressure arises, NELSSA performs a one-way handoff that preserves KV cache and avoids service interruption. This design ensures stable throughput and latency even under unpredictable, dynamically expanding workloads such as reasoning-heavy or agent-driven inference.

(3) End-to-end GPU–PNM heterogeneous system integration. We build an end-to-end GPU–PNM heterogeneous LLM serving system integrating CXL-attached PNM devices, device-level sparse attention engines, and a host-side control plane. Our implementation spans the full execution path, including RDMA communication, gRPC/TCP control, KV cache management across GPU and multi-PNM nodes, and distributed attention aggregation over a CXL interconnect. Deployed on real hardware, NELSSA demonstrates tightly coupled GPU–PNM co-execution, establishing heterogeneous memory–compute integration as a viable architecture for large-scale LLM serving, while enabling flexible and scalable disaggregated deployment through CXL.

To evaluate NELSSA, we conduct end-to-end experiments using realistic mixed-length serving traces that capture highly skewed and interleaved short and long-context requests. Compared to a state-of-the-art GPU-only serving system, NELSSA improves decode throughput by up to 5.5× in tokens/sec and reduces P99 latency by 15×, while maintaining the same level of accuracy as the GPU-only baseline. These gains stem from restoring effective running batch size on GPUs through length-based heterogeneous placement and mitigating head-of-line blocking caused by long-context requests. Even under dynamically growing contexts, NELSSA sustains stable throughput without recomputation overhead.

As emerging LLM workloads increasingly exhibit diverse and unpredictable context lengths, homogeneous GPU infrastructures face a growing mismatch with their heterogeneous resource demands. Scaling GPUs to accommodate memory-intensive requests scales compute and memory resources together, potentially leaving additional compute capacity underutilized. By combining hardware heterogeneity, runtime scheduling, and CXL-based disaggregation, NELSSA matches complementary compute and memory resources to the workloads they serve best, enabling resource-efficient composition for mixed-length LLM serving. Our end-to-end prototype and evaluation results suggest this design as a practical direction for serving evolving LLM workloads.

II. Motivation

A. The Mixed-Length Problem in LLM Serving

As LLMs with support for increasingly long contexts have become widely deployed, real-world serving workloads have come to exhibit a mix of short and long requests. Such mixed-length distributions have been observed in prior workload analyses [14], [15]; for instance, the Mooncake conversation trace [13] shows that approximately 34% of requests fall under 4K tokens while requests exceeding 100K tokens are served concurrently. As LLM usage expands into more complex workloads—such as multi-agent systems, chain-of-thought reasoning [31], and agentic coding–the proportion and length of long-context requests are expected to grow further.

When such heterogeneous requests share the same serving environment, inefficiency arises. A long-context request occupies a substantial portion of GPU HBM with its KV cache, preventing the scheduler from admitting new requests into the batch and, in memory-saturated conditions, forcing the running batch size to 1. This triggers Head-of-Line (HoL) blocking: subsequent short requests stall in the queue until the long request completes, leading to tail latency spikes and pending queue buildup. Figure 2 illustrates this effect on a single H100 GPU (94 GB), where scheduling a long-context request under HBM capacity pressure causes memory saturation, batch size collapse, and periodic throughput stalls.

The root cause lies in a fundamental resource mismatch between long-context decode attention and GPU HBM-based infrastructure. The decoding phase of Transformer-based LLM inference is auto-regressive, requiring the model to access the full accumulated KV cache to generate each token—rendering

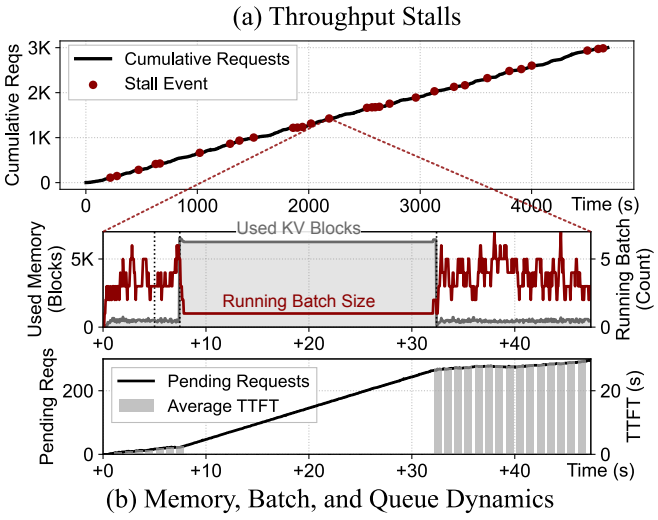


Fig. 2. GPU memory saturation, batch size collapse, and throughput stalls under mixed-length workload on a single H100 GPU (94 GB).

it a heavily memory-bound workload with a high memory-to-compute ratio [32]–[36]. As context length grows, this effect intensifies: long-context decode attention enters a capacity-bound regime in which the KV cache simultaneously saturates both HBM capacity and bandwidth, while GPU compute utilization decreases [37]. Scaling out GPUs does not resolve this efficiently [38]–[40], as adding GPUs increases total HBM capacity but scales compute and memory together, resulting in proportionally higher cost and power draw while leaving an increasing fraction of compute underutilized.

These observations point to a need for a dedicated memory tier that can provide sufficient capacity and bandwidth for long-context decode attention in a cost-effective manner, and that integrates seamlessly with existing GPU infrastructure. However, deploying such a tier in practice introduces two additional challenges that arise in real serving environments: the coexistence of heterogeneous short and long requests, and the unpredictable growth of context length during decoding. We analyze these in the following sections.

B. Limitations of Existing Approaches

As analyzed in Section II-A, GPU-only infrastructure introduces structural inefficiencies in long-context decode attention. Prior work has pursued two main directions to address this: reducing the KV cache memory footprint through algorithmic techniques, and offloading KV caches to memory outside the GPU. However, neither direction sufficiently addresses the challenges that arise in mixed-length serving environments.

KV cache reduction. KV compaction and quantization techniques [17], [19]–[23], [41] aim to reduce KV cache size to fit within GPU HBM capacity. However, since the tokens relevant to a given decode query cannot be known in advance, these methods discard information in a query-agnostic manner, leading to accuracy degradation. Aggressive compression am-

plifies this risk, creating a difficult tradeoff between capacity headroom and accuracy loss.

Sparse attention on GPU. Applying sparse attention directly on GPUs can reduce attention computation and memory bandwidth. However, accuracy-preserving sparse attention still requires the full KV cache to remain resident in HBM because token selection is query-dependent. Therefore, GPU-side sparsity alone does not remove the capacity pressure that collapses batch size and induces head-of-line blocking under mixed-length workloads.

CPU offloading with sparse attention. Approaches such as InfiniGen [25] and RetroInfer [28] retain the full KV cache in CPU DRAM and transfer a sparsely selected subset to the GPU via PCIe for each decode step. By preserving the full KV cache, these methods avoid the accuracy loss inherent in compaction. However, PCIe bandwidth poses a fundamental bottleneck regardless of the selection ratio chosen. Increasing the selection ratio improves accuracy but raises bandwidth pressure, while decreasing it reduces data movement but degrades accuracy. Furthermore, since attention computation ultimately executes on the GPU, the per-step transfer of selected KV vectors across PCIe cannot be avoided. Even under aggressive sparsity, this KV-vector-scale data movement sustains non-trivial bandwidth pressure on every decode step.

Sparse attention on PNM. PNM architectures have been explored as a promising direction to overcome these limitations. Early efforts such as CXL-PNM [42] and Hermes [43] demonstrated strong energy efficiency by executing dense attention directly on PNM. However, simply relocating dense attention execution does not escape the capacity-bandwidth dilemma. The large-capacity memory required for long-context serving inherently offers lower bandwidth than GPU HBM, imposing a fundamental constraint on decode throughput. To bridge this gap, subsequent work proposed combining sparse attention with PNM execution [29], [30], [44], demonstrating that sparsity can effectively offset the bandwidth disadvantage and enabling large-scale long-context serving.

However, existing PNM-based approaches uniformly route all requests to PNM regardless of length, which is not always the most efficient strategy. As shown in Figure 3, a crossover point exists between GPU dense attention and PNM sparse attention as a function of sequence length. Below this threshold, the fixed overhead of PNM offloading, including network latency and host-side dispatch, outweighs the bandwidth advantage PNM provides, making the GPU the faster execution path. Naively routing all requests to PNM can therefore introduce performance regression for short requests, which constitute the majority of real-world workloads.

In short, each approach offers advantages under specific conditions but falls short as a complete solution for mixed-length serving. GPU-only execution is prone to HoL blocking from long requests; algorithmic approaches incur either accuracy loss or interconnect bottlenecks; and PNM-only routing can degrade performance for short requests. Taken together, these observations point to the need for a length-based placement strategy that adaptively assigns each request to its most suitable execution tier, routing short requests to the GPU and long requests to PNM.

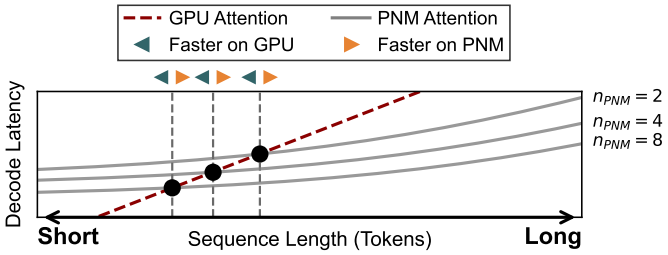


Fig. 3. Decode attention latency across sequence lengths. A crossover point exists beyond which PNM execution can outperform GPU execution, with its location varying across hardware configuration and system conditions.

C. The Challenge of Dynamic Context Growth

The approaches in Section II-B implicitly assume that the context length of a request is known before execution begins. While this holds in many scenarios, it does not always reflect the reality of modern serving workloads. Workloads such as chain-of-thought reasoning, multi-step agentic tasks, and complex inference often begin with relatively short inputs whose context grows substantially during decoding [45]–[49], in ways that are difficult to anticipate in advance [50]–[57]. When this growth causes GPU memory to be exhausted mid-execution, the scheduler must preempt the request.

Upon preemption, two recovery strategies are available. Recomputation discards the evicted KV cache and regenerates it from scratch when the request is rescheduled, consuming GPU computing resources on recovery rather than on new token generation. Swap retains the evicted KV cache in CPU DRAM but requires transferring it back over PCIe before inference can resume, reintroducing the very interconnect pressure that offloading sought to avoid [58]. Neither strategy resolves the underlying problem without significant overhead. Notably, vLLM [59] defaults to recomputation over swapping, as the system-level overhead of PCIe-based KV transfer outweighs its recovery latency benefit.

The fundamental limitation shared by both strategies is that execution remains anchored on the GPU. Regardless of where the KV cache is stored, it must be returned to the GPU before inference can resume. This points to the need for an approach that goes beyond storage relocation, one in which execution itself transitions away from the GPU so that attention can be performed directly where the KV cache resides, without a return trip. We develop this direction in Section III.

III. Design Principles and Architecture Overview

The analyses in Section II converge on a single conclusion. Long-context decode attention is inherently memory-bound, making it structurally ill-suited to GPU HBM-centric infrastructure. It saturates HBM capacity while leaving tensor core utilization low, exposing an efficiency mismatch that compounds as context length grows. Critically, this is not a

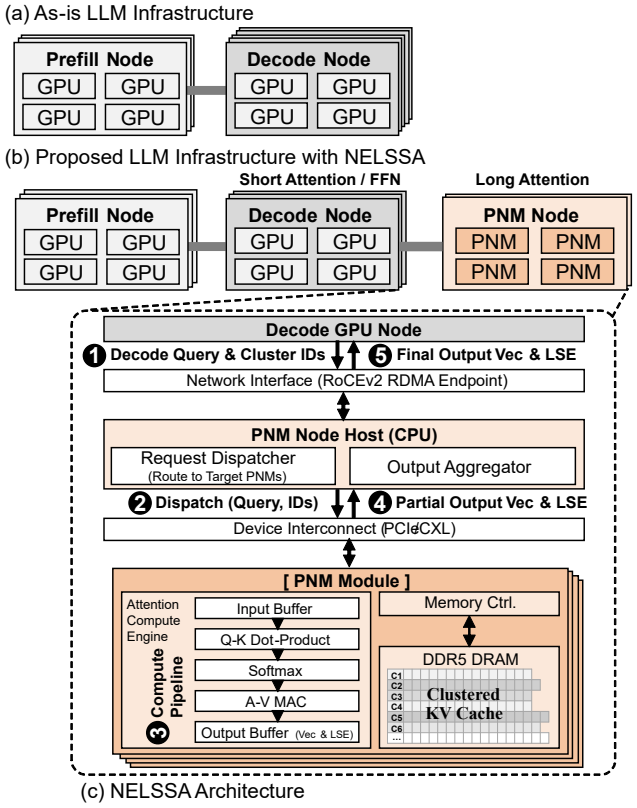


Fig. 4. Overall architecture of NELSSA.

problem of *where to store* the KV cache. It is a problem of *where to execute* attention, and fundamentally, an execution placement problem. NELSSA begins from this perspective, treating PNM as a dedicated execution tier for long-context attention rather than an auxiliary offload target. Global sparse attention computation completes entirely within PNM, without returning the historical KV cache to the GPU.

This assignment is effective for two reinforcing reasons. First, long-context decode attention is inherently memory-bound, which aligns well with PNM architecture. Each decode step requires reading the full accumulated KV cache with low arithmetic intensity, matching the characteristics of large-capacity memory with data-proximate access. Second, sparse attention structurally complements PNM's lower memory bandwidth relative to GPU HBM, offsetting the bandwidth gap and making the combination more capable than either alone. These characteristics motivate a fundamental design principle in NELSSA. No single hardware tier is universally superior, and heterogeneous resources should instead be matched to the workloads they serve best.

NELSSA applies this principle by assigning each tier according to its strengths. The GPU executes dense attention for short requests and FFN computations for all requests, while PNM exclusively handles sparse attention for long-context requests. This specialization introduces two key requirements. First, execution placement must be determined by request length. GPU-only execution is susceptible to HoL blocking from long requests, while routing all requests to PNM degrades performance for short ones, and no single strategy handles the full spectrum efficiently. Second, once the historical KV cache transfers to the PNM tier, it must not return to the GPU. Prior approaches differ in how much data they move, but share the structural constraint that execution over the offloaded KV cache ultimately resumes on the GPU. NELSSA eliminates this constraint entirely.

NELSSA satisfies these requirements through three mechanisms:

- **Length-aware hybrid routing (Section IV-A-IV-B)**: routes short requests to GPU FlashAttention and long requests to PNM sparse attention, with the placement boundary derived from hardware performance characteristics.
- **Seamless background migration (Section IV-C)**: transfers the KV cache of dynamically growing requests to PNM via a background stream, without interrupting ongoing token generation.
- **GPU–PNM heterogeneous system architecture (Section V)**: realizes both mechanisms on real PNM hardware within an end-to-end integrated serving stack.

The overall system architecture is illustrated in Figure 4.

IV. Length-Aware Execution Placement

A. Routing Threshold Derivation

The first requirement established in Section III calls for placing decode requests on different execution tiers based on sequence length, which requires a concrete placement boundary. This boundary, the routing threshold T_{input}, determines which requests are handled on the GPU and which are offloaded to the PNM tier. Rather than fixing this boundary heuristically, NELSSA derives T_{input} from the hardware performance characteristics of each execution path, providing a principled baseline that captures when PNM execution offers a more suitable operating point than GPU execution across latency, throughput, and efficiency considerations. To this end, NELSSA constructs a decode attention latency model for both paths and identifies their crossover point as T_{input}.

GPU dense attention. During the decode phase, each generated token requires reading the full accumulated KV cache, making attention strictly memory-bound [33]. The latency of GPU attention is dominated by HBM bandwidth and can be approximated as:

$$T_{\text{GPU}}(L) = \frac{L \cdot M_{kv}}{BW_{\text{GPU}}} \tag{1}$$

where L is the sequence length, M_{kv} the KV cache size per token, and BW_{GPU} the HBM bandwidth of the GPU.

PNM sparse attention. For NELSSA with N PNM devices, the decode latency is approximated as the sum of three dominant components: GPU-side centroid similarity search, distributed sparse attention across N PNM devices, and communication overhead $T_{comm}(N)$:

$$T_{\text{PNM}}(L, N) = \frac{L \cdot M_{kv}}{C \cdot BW_{\text{GPU}}} + \frac{L \cdot S \cdot M_{kv}}{N \cdot BW^{\text{eff}}_{\text{PNM}}} + T_{\text{comm}}(N) \tag{2}$$

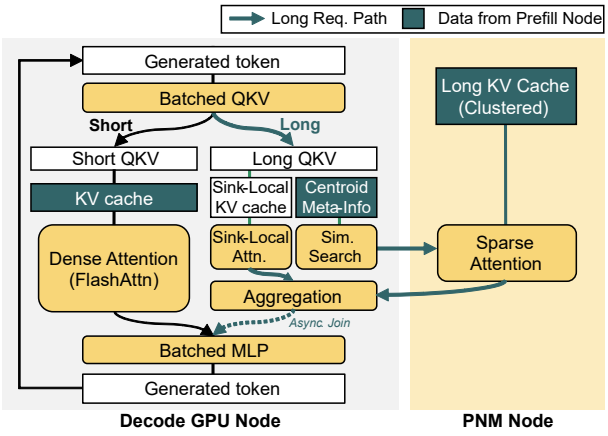


Fig. 5. Split-batch hybrid routing branches only the attention phase by sequence length, while QKV and MLP computations are globally batched on the GPU.

where C is the compression factor of the centroid search, S the sparse selection ratio, and $BW_{\text{PNM}}^{\text{eff}}$ the effective per-device bandwidth measured on our prototype under sparse random access. The communication term $T_{\text{comm}}(N)$ captures three additive costs: a fixed base network round-trip latency that applies to all configurations, a per-device incremental transfer cost that scales modestly with N, and a constant host-side dispatch and aggregation overhead that arises only in multi-PNM configurations.

Crossover point. Equating $T_{GPU}(L)$ and $T_{PNM}(L, N)$ yields a hardware-derived crossover point as a function of device count:

$$L_{\text{crossover}}(N) = \frac{T_{\text{comm}}(N)}{M_{kv}\left(\frac{C-1}{C \cdot BW_{\text{GPU}}} - \frac{S}{N \cdot BW_{\text{PNM}}^{\text{eff}}}\right)} \quad (3)$$

This formulation shows that the bandwidth gap between HBM and DDR5 can be offset by sparsity and multi-PNM parallelism as long as $\frac{S}{N \cdot BW_{\text{PNM}}^{\text{eff}}} < \frac{C-1}{C \cdot BW_{\text{GPU}}}$, a condition that becomes progressively easier to satisfy as N increases. Figure 3 illustrates how $L_{\text{crossover}}(N)$ varies with device count. With a single PNM device, the crossover lies at a sequence length that exceeds most practical workloads, motivating the multi-PNM design. As N increases, the crossover shifts toward shorter sequences, enabling a broader range of requests to be efficiently served on the PNM tier.

The purpose of this model is not to predict exact latency, but to expose the hardware-dependent trade-offs that govern when execution should transition across tiers. Applying the model to the prototype determines a hardware-specific crossover point, which NELSSA adopts as the routing threshold T_{input}. The exact value varies with hardware characteristics and deployment configuration.

B. Split-Batch Hybrid Routing

Based on the threshold T_{input} derived in Section IV-A, NELSSA partitions incoming requests into two execution paths. Short requests—those with sequence length below T_{input}—are executed on the GPU via FlashAttention, while long requests exceeding T_{input} are routed to the PNM tier for sparse attention. This separation is enforced as an architectural invariant, not merely a routing policy: once a long request's attention is assigned to the PNM tier, it executes exclusively there without GPU recall. This structurally eliminates interconnect round-trips that would otherwise reintroduce interference to concurrently running short requests.

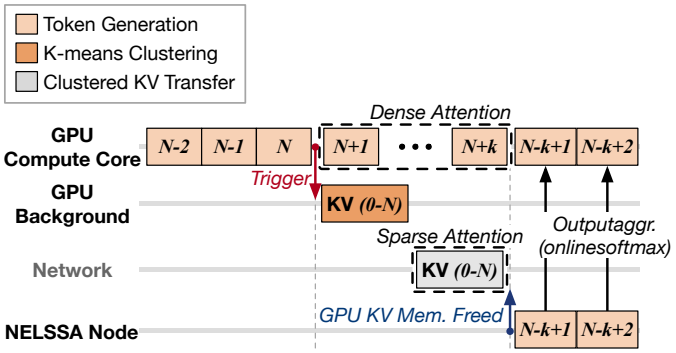


Fig. 6. Seamless background migration. Historical KV caches are asynchronously transferred to the PNM Node while GPU token generation continues uninterrupted.

Batched execution. Separating the two execution paths must not come at the cost of GPU compute efficiency. To preserve batch utilization, NELSSA globally batches the QKV and MLP computations of both short and long requests on the GPU; the two paths diverge only at the attention stage. Short requests execute FlashAttention entirely within the GPU. For long requests, computation bifurcates: the GPU handles a local attention window covering sink and recent tokens [34], while cluster-based sparse attention over the global context is asynchronously offloaded to the PNM. The sparse attention leverages query-driven cluster selection [28], and the partial results from GPU local attention and PNM global attention are merged via online softmax-based distributed aggregation—a combination that is mathematically equivalent to attention over the union of the local and selected sparse contexts via LSE-based merging. While PNM offloading proceeds, the GPU immediately continues with MLP computations for short requests; once the PNM result returns, the long request rejoins the next available MLP batch. This ensures that the execution of short requests is never stalled by PNM offloading of long requests.

The overall structure is illustrated in Figure 5. This design maximizes tensor core utilization while providing the memory capacity required for long-context execution.

C. Seamless Background Migration

The split-batch routing described in Section IV-B determines the execution tier based on the input length of each request. However, as analyzed in Section II-C, context length is often not fixed at the time a request begins execution. A request that starts below T_{input} and is initially assigned to the GPU can accumulate a growing KV cache during decoding,

dynamically giving rise to the same HBM capacity pressure identified in Section IV-A. To handle this, NELSSA extends the one-way execution handoff principle to dynamically growing requests, transferring the accumulated KV cache to the PNM tier in the background and executing sparse attention there without returning to the GPU.

Migration logic. NELSSA employs a threshold-based migration policy triggered by GPU memory pressure. A request becomes a migration candidate once its accumulated context length exceeds T_{input}. Among all candidates, migration is initiated for the request with the largest accumulated context when preemption is imminent, ensuring that memory is reclaimed without disrupting requests that can continue executing on the GPU. This design avoids premature migration under sufficient memory headroom while preventing preemption-induced recomputation overhead.

Background handoff. Upon triggering, the GPU asynchronously transfers the historical KV cache to the PNM tier via a background stream, without interrupting token generation. Once the transfer completes, the request transitions to NELSSA's distributed execution mode. At handoff, the GPU retains only the lightweight Centroid Meta-Info required for subsequent routing and immediately reclaims the historical KV memory to accommodate new requests. KV cache generated by tokens produced after the handoff is periodically clustered in predefined chunks and appended to the PNM tier incrementally, amortizing clustering overhead and minimizing memory fragmentation. By consistently applying the one-way handoff principle to both statically classified long requests and dynamically growing ones, NELSSA separates long-context attention from the GPU in both cases, preserving overall system memory efficiency and throughput. The flow is shown in Figure 6.

V. GPU–PNM Heterogeneous System

A. NELSSA System Architecture

The length-based routing and background migration mechanisms in Section IV presuppose tight GPU–PNM interaction, and this section presents the architecture that makes them practically realizable. Building on prefill-decode disaggregation [60], [61]—a direction widely adopted in recent LLM serving systems—NELSSA introduces a disaggregated hybrid architecture that further separates long-context workloads onto a dedicated high-capacity memory tier, addressing the decode attention capacity bottleneck that prefill-decode separation alone cannot resolve (Figure 4(b)).

The system comprises three components. The Prefill Node handles prefill computation for all requests. For long-context requests, it performs segmented K-means clustering in parallel with prefill computation, preparing the KV cache in a clustered form suitable for PNM offloading. This clustering scheme is adopted from RetroInfer [28], which reports that the associated overhead accounts for approximately 2% of total prefill time. Short-context requests are forwarded directly to the Decode GPU Node upon prefill completion. The Decode GPU Node executes FlashAttention for short-context requests and all FFN computations. For long-context requests, it retains only the lightweight Centroid Meta-Info and performs similarity search to derive the target cluster IDs for routing to the NELSSA Node. The NELSSA Node is a high-capacity memory node connected via RoCEv2, housing multiple PNM modules. Each module stores its assigned partition of the clustered KV cache in DDR5 DRAM following a head-wise partitioning scheme (Section V-C), and executes sparse attention in parallel.

Architectural isolation. In this disaggregated infrastructure, the PNM Node must concurrently ingest massive prefill datasets and execute decode attention streams. Rather than relying on a complex runtime scheduler to dynamically balance these workloads, NELSSA enforces architectural isolation. Prefill data traffic is managed through an independent host-side memory path, bypassing the PNM's internal compute pipelines. This structural design prevents data-ingestion traffic from introducing latency jitter into decode attention execution.

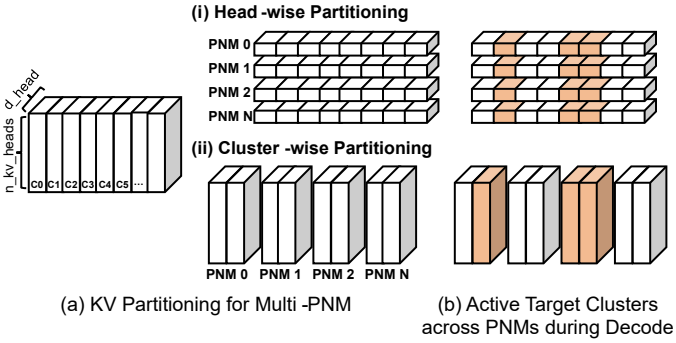
Independent scaling. Beyond functional separation, the disaggregated structure enables independent scaling of each tier. Since the Decode GPU Node and the NELSSA Node connect over commodity Ethernet (RoCEv2), each tier scales independently. As the proportion of long-context workloads grows, additional NELSSA Nodes expand both memory capacity and attention throughput; when GPU-side compute becomes the bottleneck, Decode GPU Nodes scale out instead. In this way, disaggregation serves as a structural enabler that allows each tier to be provisioned according to its own workload characteristics.

B. Execution Pipeline

NELSSA's execution pipeline consists of an initial Prefill Phase and an iterative Decode Phase. The routing decision that determines whether a request is handled on the GPU or offloaded to the NELSSA Node is derived in Section IV-A–IV-B; here we describe the execution flow for each request type once that assignment has been made.

Prefill phase: For long-context requests, segmented K-means clustering is performed layer-by-layer, overlapping with prefill computation to amortize the clustering overhead. The resulting Clustered KV Cache is offloaded to the NELSSA Node, while the GPU retains only the lightweight Centroid Meta-Info for subsequent routing.

Decode phase: During token generation, the GPU performs a similarity search using the Centroid Meta-Info to derive the Top-K target cluster IDs, transmitting them alongside the query to the NELSSA Node (❶). NELSSA's Host CPU Dispatcher distributes computation commands to the specific PNM modules holding the target data (❷). Each PNM module retrieves the target clusters from DDR5 DRAM into its input buffer and executes sparse attention locally, producing partial output vectors and Log-Sum-Exp (LSE) values (❸), ❹). The Host's Output Aggregator collects these results and returns them to the GPU (❺), which then merges them with its own local attention results via online softmax-based LSE merging to generate the final token.



(a) KV Partitioning for Multi -PNM (b) Active Target Clusters across PNMs during Decode

Fig. 7. KV cache partitioning strategies for multi-PNM development: (i) head-wise and (ii) cluster-wise.

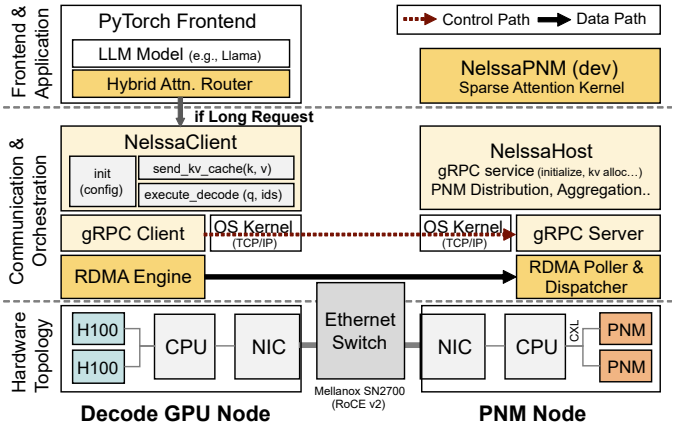


Fig. 8. Overall NELSSA SW stack.

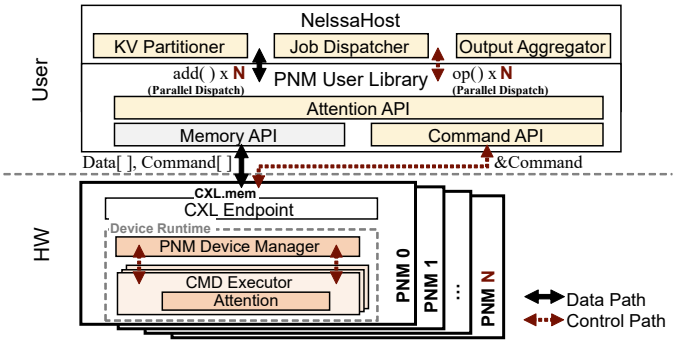


Fig. 9. NELSSA PNM software stack.

Dynamic context update: Whenever newly generated tokens accumulate to a predefined chunk size in GPU memory, the GPU independently clusters this chunk, appends it to the NELSSA Node, and updates the Centroid Meta-Info. This incremental approach amortizes clustering overhead and accommodates dynamic context growth while minimizing memory fragmentation.

C. KV Partitioning for Multi-PNM Scalability

NELSSA distributes the KV cache across multiple PNM modules to scale both memory capacity and attention throughput in proportion to deployment needs. The partitioning strategy is a key design choice that determines the efficiency of the multi-PNM system (see Figure 7). Cluster-wise partitioning (Figure 7(ii)) stores complete clusters on individual modules, preserving large access granularity. However, depending on workload characteristics and sparse selection patterns, target clusters may concentrate on specific modules, potentially leading to load imbalance. While this effect may diminish as context length grows, NELSSA adopts head-wise partitioning (Figure 7(i)), where all PNM modules process the same list of clusters but compute only their assigned attention heads. This provides robust load balancing regardless of context length or dynamic sparsity patterns. Although partitioning across multiple modules reduces the access granularity per cluster, this size still exceeds the typical row buffer capacity (1–2 KB) of DDR5 DRAM, sufficiently amortizing the row activation penalty (t_RCD). Combined with bank-level parallelism, head-wise partitioning maintains high effective bandwidth even under random access patterns.

VI. Implementation

The architecture in Section V is a reference design targeting a dedicated compute pipeline. Our prototype realizes it on real PNM hardware with DDR5 DRAM and ARM Neoverse V2 cores over CXL. Using these general-purpose cores in place of the dedicated compute units, the prototype provides a conservative characterization of end-to-end behavior and enables direct measurement of the system-level properties central to this work.

A. End-to-End System Integration

We implemented an end-to-end prototype system connecting the Decode GPU Node and the PNM Node using real hardware and open-source frameworks (see Figure 8). The frontend on the GPU Node is implemented in Python for integration with PyTorch, while the performance-critical communication backend is implemented in C++. To avoid the latency and CPU overhead of routing all traffic through a kernel-managed network stack, NELSSA decouples the control path from the data path. Latency-insensitive operations such as session initialization and KV cache allocation are handled via gRPC over TCP/IP, while performance-critical transfers, specifically KV cache offloading and query/output vector exchanges, are executed over RoCEv2 RDMA, bypassing the OS kernel and enabling direct memory-to-memory transfers. A dedicated RDMA Poller thread eliminates context-switching delays, achieving a one-way latency of a few microseconds. On the PNM Node, the NelssaHost dispatches commands to the target modules and returns the aggregated results to the GPU Node over RDMA.

B. PNM Software Stack

The PNM software stack is illustrated in Figure 9, organized into three layers: the NelssaHost, the PNM User Library, and the OS/HW layer. During prefill, the NelssaHost's KV

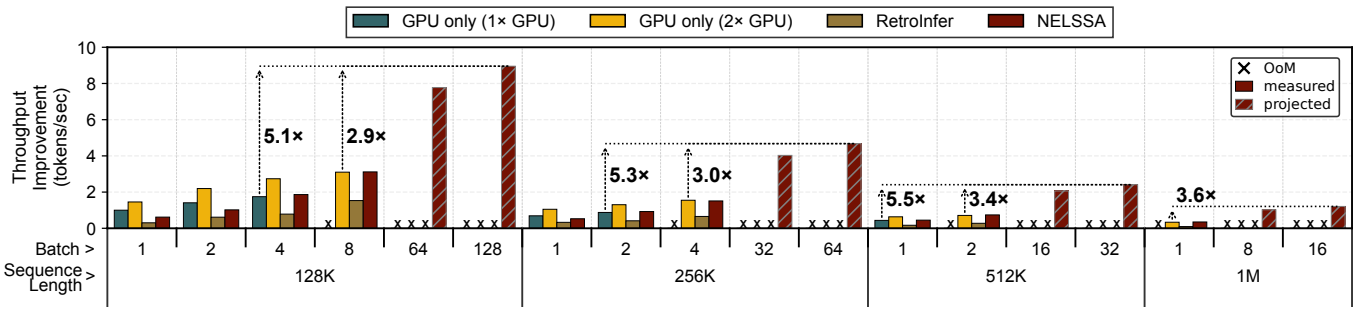


Fig. 10. Decode throughput comparison across varying sequence lengths and batch sizes (measured/projected). NELSSA achieves up to 5.5× higher throughput, including configurations where GPU-only baselines fail due to out-of-memory (OoM).

Partitioner distributes the clustered KV cache across N PNM devices via RDMA. During decode, the Job Dispatcher concurrently issues compute commands to all PNM devices, and the Output Aggregator collects the resulting partial output vectors and LSE values, merges them via online softmax, and returns the final output to the Decode GPU Node. Since only output vectors and LSE values are aggregated, merging overhead remains small and independent of KV cache size.

The PNM User Library exposes a Memory API for KV cache allocation and a Command API for compute operations. To balance prefill data ingestion and decode computing, NELSSA relies on API-level structural isolation rather than complex dynamic scheduling. The Memory API handles prefill data distribution by bypassing the device's compute pipeline entirely. Meanwhile, the Command API independently manages decode execution, ensuring that the dedicated CMD Executor threads remain uninterrupted by background memory tasks. On the device side, the Device Runtime interprets command descriptors and assigns a CMD Executor thread per descriptor for parallel execution, with each executor running the target kernel and returning the result upon completion.

VII. Evaluation Methodology

Hardware configuration. The Decode GPU Node consists of an Intel Xeon Platinum 8452Y CPU, 1 TB DDR5-4800 memory, and one or two NVIDIA H100 NVL GPUs (94 GB each). In the 2× GPU configuration, the two GPUs are connected via NVLink. The PNM Node, connected to the Decode GPU Node via a commodity Ethernet fabric, pairs an Intel Xeon 6952P CPU with four CXL-attached PNM devices, each equipped with four channels of on-board DDR5-6400 DRAM. In the current prototype, each PNM device is limited to 32 GB of device memory due to the on-board DRAM capacity. We accordingly report projected throughput at a 512 GB per-device configuration, consistent with commodity DDR5 platforms and the full-capacity PNM board we are currently developing. Table I details both hardware configurations. The projected model assumes only a memory expansion to 512 GB, keeping all other specifications identical to the prototype. The projection methodology and its validation are described in Section VIII-A.

Component	Prototype (Measured)	Projected
Compute Core	16x ARM Neoverse V2	16x ARM Neoverse V2
Interconnect	CXL 2.0 x16	CXL 2.0 x16
Memory Interface	4-CH DDR5-6400	4-CH DDR5-6400
Memory Bandwidth	200 GB/s	200 GB/s
Memory Capacity	32 GB	512 GB

TABLE I
HARDWARE SPECIFICATIONS FOR THE MEASURED PROTOTYPE AND PROJECTED CONFIGURATION.

Workloads. We evaluate on Llama3-8B-1048K [62] across sequence lengths from 128K to 1024K tokens. Accuracy is measured using the RULER benchmark [63]. For mixed-length serving evaluation, we construct a synthetic workload by sampling requests from the Mooncake conversation trace [13] under a Poisson arrival process, configured to reflect target mixed-length scenarios.

Baselines. We compare against three baselines. GPU-only (1× H100 NVL) and GPU-only (2× H100 NVL, NVLink) both execute dense decode attention via FlashAttention without any offloading. The 2×GPU baseline is included as a cost-comparable reference to account for the additional hardware resources in the NELSSA configuration. RetroInfer [28] is a state-of-the-art CPU offloading system with sparse attention. NELSSA is evaluated as an end-to-end system comprising the Decode GPU Node and the PNM Node.

Metrics. The primary metric is decode throughput in tokens per second. We additionally report tail latency (P95 and P99 TPOT) and GPU memory utilization as secondary metrics.

VIII. Experimental Results

A. End-to-End Throughput Evaluation

Decode throughput. Figure 10 compares decode throughput across baselines and NELSSA over varying sequence lengths and batch sizes. All experiments are conducted at a 4% selection ratio, which ensures stable accuracy as established in Section VIII-B. Within the directly measured range (up to batch 8 at 128K, batch 4 at 256K, batch 2 at 512K, and batch 1 at 1M tokens), NELSSA serves all configurations where GPU-only baselines fail due to memory exhaustion, while sustaining throughput comparable to the GPU-only baselines even on our capacity-constrained prototype. Beyond this range, we project throughput to the full-capacity 512 GB per-device configuration described in Section VII. The throughput advantage

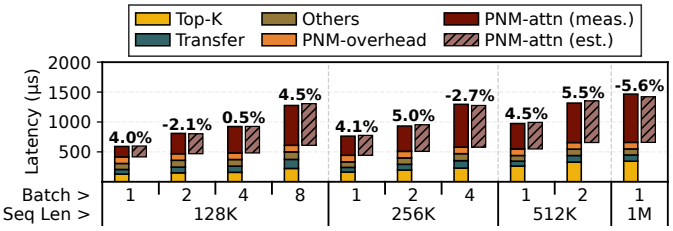


Fig. 11. Component-level breakdown of decode attention latency per step across sequence lengths and batch sizes. Discrepancy between measured (meas.) and estimated (est.) values for PNM-attn is within ±5.6%.

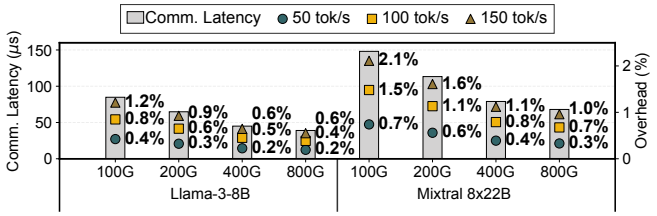


Fig. 12. GPU-PNM cross-node communication overhead as a fraction of total decode latency across different target throughputs and network configuration (analytical model).

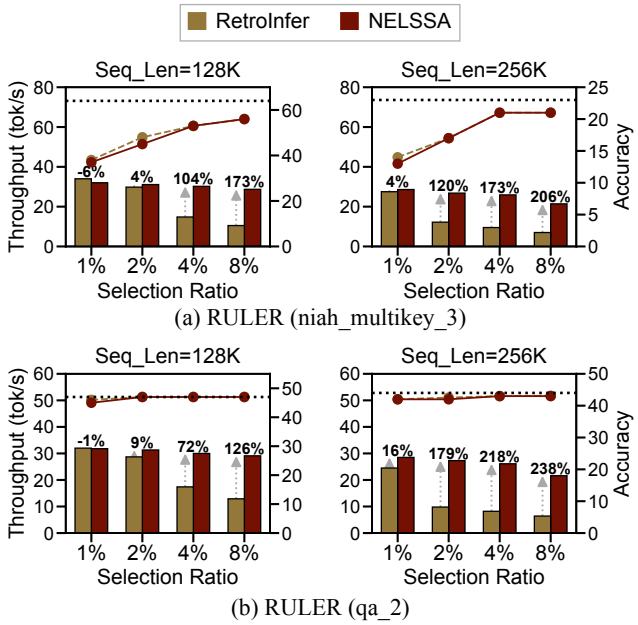


Fig. 13. Decode throughput and accuracy under varying selection ratios (Llama3-8B-1048K). The dashed horizontal line denotes the accuracy of full dense attention.

grows with sequence length and batch size, as larger workloads improve PNM compute efficiency and reduce the relative impact of fixed per-step costs. Across the full evaluation range, including the projected configurations, NELSSA achieves up to 5.5× higher throughput than the 1×GPU baseline and up to 3.6× higher throughput than the 2×GPU baseline. These peak ratios compare the highest throughput each system attains at a given sequence length, since the GPU-only baselines cannot execute the larger batch sizes.

NELSSA achieves these gains using a PNM tier built on commodity DDR5 with substantially lower memory bandwidth than GPU HBM, demonstrating how heterogeneous systems can exploit complementary hardware strengths through workload-aware placement rather than relying on uniformly higher-performance hardware.

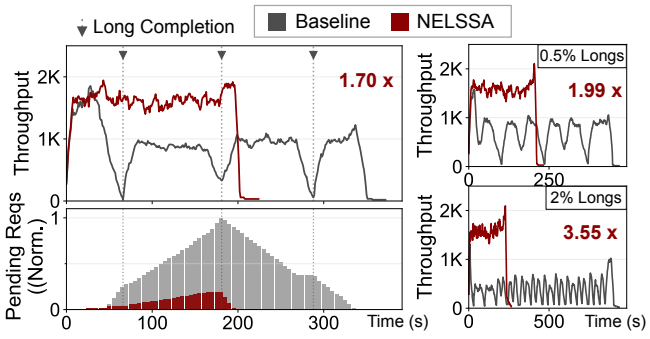
Latency breakdown and projection. Figure 11 presents a component-level breakdown of the measured decode attention latency per step. The five components are Top-K similarity search, Transfer, PNM-overhead, PNM-attn, and Others, all of which are directly measured on the hardware prototype. For PNM-attn, estimated values derived from effective bandwidth and data volume are shown alongside measured values, with a discrepancy within ±5.6%. The projection is constructed from this breakdown. Top-K values are taken from direct measurements across the full projected range. Transfer and Others are treated as constants using their measured averages within the evaluated range. PNM-overhead may diminish as data volume grows, but its measured average is held fixed as a conservative estimate. For PNM-attn, measurements show a tendency for effective bandwidth to increase with data volume, but the effective bandwidth observed within the measured range is held constant across the full projected range. As only PNM-attn relies on estimation, the projection error of the overall breakdown is dominated solely by this term.

Network sensitivity. We analyze the GPU-PNM cross-node communication overhead under 100G–800G network configurations (Figure 12). Even under a stringent 150 tok/s target, which is a challenging condition for long-context workloads, the communication accounts for at most 2.1% of the total decode latency. Due to the minimal per-step data exchange, NELSSA structurally avoids network bottlenecks despite requiring per-step coordination.

B. Accuracy and Throughput under Varying Sparsity

Accuracy. Figure 13 presents decode throughput and accuracy across selection ratios on RULER benchmark tasks. At a selection ratio of 4% and above, NELSSA maintains stable accuracy across both tasks while sustaining competitive throughput, confirming that this range provides a stable accuracy-throughput tradeoff. The decode throughput evaluation in Section VIII-A is accordingly conducted at a 4% selection ratio.

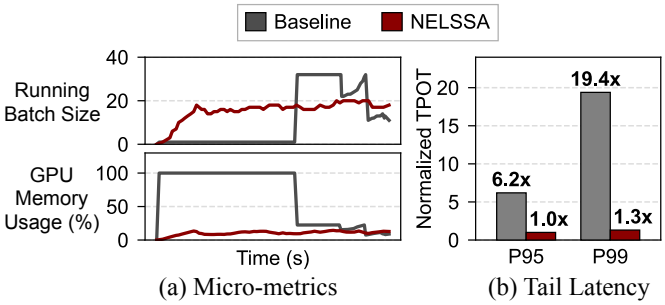
Throughput under varying sparsity. NELSSA exhibits a gradual and stable throughput trend as the selection ratio increases. Because attention computation completes entirely within PNM and only the result is returned, changes in the selection ratio do not directly affect interconnect traffic. In contrast, CPU offloading approaches transfer a proportionally larger volume of KV vectors across PCIe as the selection ratio increases, exposing them to interconnect pressure as a structural constraint. Throughput in such approaches can also vary depending on workload characteristics. As a result, CPU offloading approaches tend to achieve stable throughput only at



(a) Throughput & Queue Dynamics (3 Longs)

(b) Impact of Long Request Ratio

Fig. 14. System-wide decode throughput and pending queue dynamics (LLaMA-3-8B). (a) NELSSA sustains throughput under long request injection while the baseline stalls. (b) NELSSA's throughput advantage grows as the fraction of long requests increases.



(a) Micro-metrics

(b) Tail Latency

Fig. 15. Directly measured GPU memory utilization, running batch size, and tail latency upon injecting a single 256K long request (normalized to NELSSA P95). NELSSA preserves memory headroom and batch size, achieving 6.2× and up to 15× reduction in TPOT at P95 and P99, respectively.

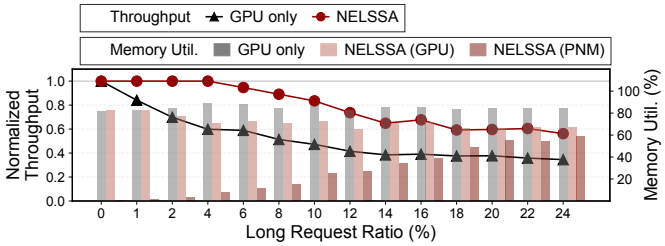


Fig. 16. Decode throughput under varying long request fractions from 0% to 24% for the baseline and NELSSA.

low selection ratios, whereas NELSSA sustains a broad region where both accuracy and throughput requirements are jointly satisfied across a wide range of selection ratios.

C. Head-of-Line Blocking Mitigation

Evaluation methodology. To evaluate the system-wide impact of NELSSA's hybrid routing, we employ a two-stage methodology combining direct hardware measurement with measurement-grounded system-level emulation. (i) GPU memory utilization, running batch size, and tail latency (P95/P99 TPOT) are measured directly on our hardware prototype, providing empirical evidence that NELSSA preserves GPU memory headroom and sustains higher running batch sizes under mixed-length workloads. (ii) For end-to-end decode throughput, we adopt an emulation methodology grounded in direct hardware measurements. The key insight is that the impact of NELSSA on GPU-side throughput is governed by its GPU memory and compute occupancy patterns, both of which are directly measurable on our prototype. We construct a workload injection sequence calibrated to these measured values and inject it into an unmodified vLLM engine, preserving its original scheduling, batching, and memory management behaviors so that request-level interactions are faithfully reproduced. Since the injected workload is constructed from requests at approximately 100K tokens, validation is performed at this length against direct hardware execution, confirming a measurement error of 4.12%. The directly measured micro-metrics independently corroborate the same architectural trends, making the two methods mutually reinforcing in characterizing the system-level benefits of our approach.

Micro-metrics and tail latency. The fundamental mechanism driving HoL blocking mitigation is evident in the directly measured micro-metrics (Figure 15). Upon the arrival of long requests, the baseline's GPU memory hits its capacity, forcing the scheduler to reduce the running batch size to as low as 1. In contrast, by decoupling the massive attention memory footprint, NELSSA maintains memory headroom on the GPU, seamlessly sustaining a higher running batch size. This architectural isolation directly translates into substantial improvements in the directly measured tail latency of short requests. Based on normalized Time-Per-Output-Token (TPOT) measurements, NELSSA achieves a 6× improvement at the 95th percentile (P95) and up to a 15× improvement at the 99th percentile (P99) compared to the baseline.

Throughput and queue dynamics. These micro-architectural differences directly manifest in the macroscopic throughput and pending queue dynamics, as illustrated in Figure 14(a). Under mixed traffic (RPS=10), the baseline throughput experiences a notable drop upon long request injection, where the extent and duration of the throughput degradation vary depending on the burst size (e.g., BS=2 vs. BS=1). Furthermore, even after the long requests complete, the baseline fails to immediately restore its peak decode throughput. This sluggish recovery is primarily attributed to the overhead of swapping in previously evicted KV caches from the CPU and the scheduler's gradual restoration of the running batch size. Consequently, pending short requests rapidly accumulate to saturation (1.0). Conversely, NELSSA sustains consistently high throughput and averts queue accumulation.

Figure 14(b) demonstrates NELSSA's robustness across diverse workload configurations with varying RPS and long request counts. While the baseline throughput is volatile under stress, NELSSA remains stable across the tested configurations. As the frequency of long requests increases (e.g., from 1 to 7 bursts, or reaching 2% of total requests), the stalling in the baseline worsens, and NELSSA's relative throughput

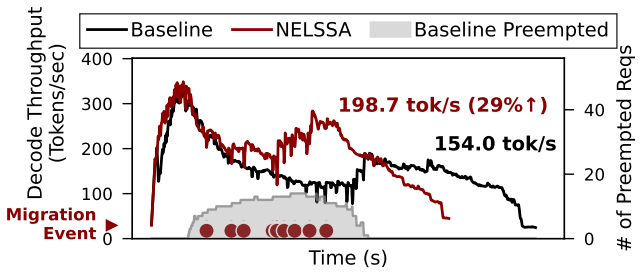


Fig. 17. System-wide decode throughput and preemption count under dynamic context growth. NELSSA sustains stable throughput via seamless background migration while the baseline suffers repeated preemptions.

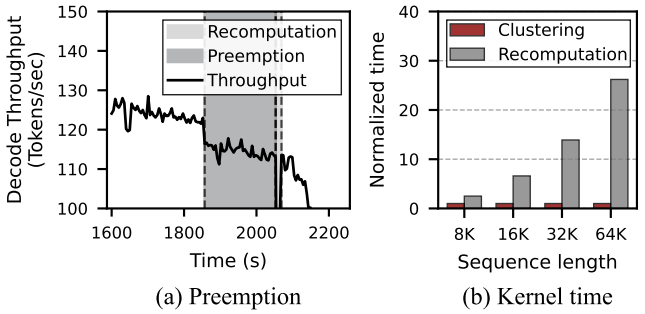


Fig. 18. (a) Throughput degradation at preemption events in the baseline. (b) Recomputation latency vs. incremental clustering overhead, directly measured on hardware.

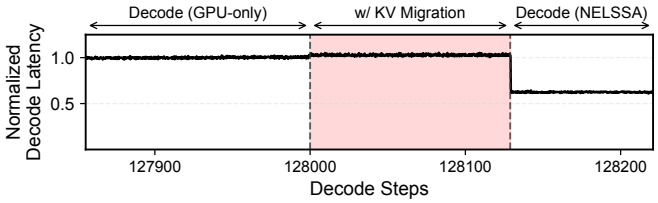


Fig. 19. Decode latency jitter during concurrent KV cache migration (microbenchmark at 128K sequence length).

speedup shows an increasing trend. Considering the trajectory of future workloads–where the proportion and length of long-context requests are projected to increase–NELSSA's ability to fundamentally mitigate HoL blocking will be an increasingly important property for scalable LLM serving.

Long request sensitivity. To stress NELSSA under skewed request mixes, we fix the arrival rate and sweep the fraction of long-context requests from 0% to 24%. Figure 16 shows that the GPU-only baseline quickly loses throughput as long requests consume HBM and collapse the running batch size. In contrast, NELSSA's throughput degradation occurs only at a much higher long-request fraction and remains well above the baseline throughout the evaluated range, demonstrating robustness to skewed workload mixes.

D. Seamless Migration under Dynamic Context Growth

Evaluation methodology. To evaluate NELSSA's effectiveness under dynamic context growth, we configure a synthetic workload with an input-to-output length ratio of 1:2, reflecting long-output generation patterns common in workloads such as reasoning and chain-of-thought inference. In this setting, the number of output tokens is difficult to predict in advance [50]–[57], causing frequent GPU memory saturation and out-of-memory events in the baseline. When GPU memory utilization reaches the migration threshold, NELSSA applies its migration policy to transfer the KV cache of requests exceeding the threshold to the PNM node via seamless background migration, allowing inference to continue without interruption. System-wide throughput trends are captured using the same emulation methodology as Section VIII-C, while migration overhead is directly measured on the hardware prototype.

Throughput and migration overhead. Figure 17 shows system-wide throughput under the dynamic growth scenario. In the baseline, GPU memory saturation causes the running batch size to shrink progressively, increasing the number of preempted requests and reducing overall throughput. NELSSA reclaims GPU memory headroom through seamless background migration and sustains a stable running batch size throughout the entire execution. In this scenario, token generation time accounts for the majority of end-to-end latency, which naturally limits the relative throughput gain from migration. Nevertheless, NELSSA achieves a 29% improvement over the baseline.

Furthermore, a 128K sequence microbenchmark (Figure 19) shows that background KV cache transfers cause no significant decode latency jitter. This confirms concurrent migration introduces minimal interference to active inference.

Figure 18 examines the underlying cause of this throughput difference. Figure 18(a) shows that preemption events in the baseline cause throughput degradation. As shown in Figure 18(b), this stems from recomputation latency, which grows steeply with context length. In contrast, NELSSA's incremental clustering overhead remains relatively insensitive to context length, a property confirmed through direct hardware measurement. Taken together, these results demonstrate that the one-way handoff principle in NELSSA extends to dynamically growing requests, sustaining stable execution even when context length exceeds GPU memory capacity at runtime.

IX. Related Work

A. Algorithmic KV Cache Management

Algorithmic approaches to managing long-context KV caches fall into two categories. KV compaction and quantization methods [17]–[19], [22], [23], [41] reduce HBM footprint by pruning or summarizing less important tokens, but discard information in a query-agnostic manner, leading to accuracy degradation that worsens as the compression ratio increases. Sparse attention approaches with CPU offloading [24]–[28], [64] avoid this loss by retaining the full KV cache in host memory, but face an unavoidable tradeoff: increasing the

selection ratio raises PCIe bandwidth pressure, while reducing it degrades accuracy. In both categories, a single parameter (compression ratio or selection ratio) jointly governs accuracy and throughput, and this tradeoff becomes more sensitive under varying workload characteristics.

B. Sparse Attention on PNM

Early PNM-based efforts [42], [43] demonstrated the viability of near-memory attention execution but remained limited in throughput by lower memory bandwidth than GPU HBM. To address this, subsequent work proposed combining sparse attention with PNM execution [29], [30], [44], and the prior work [44] demonstrated through simulation that sparsity can effectively mitigate the capacity-bandwidth tradeoff.

LongSight [29] and ScalablePNM [30] propose more concrete architectures in this direction, but differ fundamentally from NELSSA in execution model. LongSight [29] performs filtering and Top-K selection within a compute-enabled memory device, but transfers the selected Top-K QKT and Value vectors to the GPU for final attention computation, leaving interconnect overhead unresolved and constraining the selection ratio through hardware limits. ScalablePNM [30] executes the full pipeline from token selection to attention computation entirely within PNM, but relies on a custom LPDDR5X-based PNM platform delivering 1.1 TB/s per module and is validated only through architectural simulation. On commodity DRAM, the bandwidth available for near-memory processing is substantially lower, and performing token selection within PNM under this constraint introduces non-trivial decode latency overhead in practical deployments. NELSSA accounts for this by assigning token selection to GPU-side centroid search and confining attention execution to PNM, arriving at a practical operating point that satisfies both latency and accuracy requirements on commodity DRAM. This placement boundary is derived from hardware performance characteristics rather than set heuristically.

Both LongSight [29] and ScalablePNM [30] assume static request lengths and do not address system-level challenges such as mixed-length workload heterogeneity or dynamic context growth in real serving environments. To the authors' knowledge, NELSSA is the first end-to-end system implemented on real PNM hardware that jointly addresses these challenges within a unified serving stack.

X. Discussion

Adaptive routing under workload skew. While NELSSA's length-aware routing provides a stable first-order placement policy, unpredictable workload skews in real-world multi-tenant environments may temporarily saturate a specific hardware tier. A cluster-level orchestrator can address this limitation by monitoring queueing delay and resource utilization across GPU and PNM nodes and dynamically adjusting the routing threshold T_{input} as system conditions change. Because NELSSA is designed as a modular extension to prefill–decode disaggregated infrastructures, such load-aware scheduling can be incorporated at the orchestration layer without changing the underlying execution mechanisms or data-center topology.

Broader workload and hardware heterogeneity. NELSSA focuses on one form of workload and hardware heterogeneity by matching mixed-length requests across GPU and PNM resources. Future LLM serving environments may exhibit substantially broader heterogeneity on both sides. Agentic workloads can introduce increasingly diverse and dynamic execution characteristics, while emerging hardware platforms span different tradeoffs in compute capability, memory bandwidth and capacity, and interconnect characteristics. Supporting such environments will require serving systems to move beyond placement across a fixed pair of hardware tiers and continuously match evolving workload demands to a diverse pool of heterogeneous resources. Extending workload-aware placement across broader dimensions of workload and hardware heterogeneity remains an important systems challenge.

Cost-aware heterogeneous provisioning. NELSSA primarily optimizes performance and resource utilization rather than directly optimizing deployment cost. Although our evaluation includes a cost-comparable 2×GPU baseline, the most effective heterogeneous configuration ultimately depends on hardware cost, power consumption, workload composition, and service-level objectives. Future systems can incorporate these factors into both provisioning and placement decisions, jointly determining how much capacity to allocate to each hardware tier and how requests should be mapped across them. Such cost-aware orchestration could further reduce over-provisioning by scaling heterogeneous resources according to the workload characteristics they serve best.

XI. Conclusion

This work builds NELSSA, a GPU–PNM heterogeneous LLM serving system for mixed-length workloads, introducing length-based request placement with a hardware-aware crossover threshold and a seamless one-way migration mechanism that enables execution handoff under dynamic context growth without recomputation. Our results show that heterogeneous execution can effectively address the resource mismatch of mixed-length LLM serving by assigning GPU and PNM resources to the workloads they serve best. These results motivate a broader direction for LLM infrastructure as agentic AI introduces increasingly diverse and dynamic workloads. Rather than uniformly scaling a single hardware tier, future serving systems can compose heterogeneous compute, memory, and interconnect resources according to their complementary strengths, enabling more scalable and resource-efficient LLM serving. NELSSA takes an initial step toward this vision, demonstrating how workload-aware heterogeneous execution can be realized in a practical GPU–PNM serving system.

XII. Acknowledgements

This work was conducted as part of a research collaboration between the SK hynix Memory Systems Research (MSR) and the KAIST CASYS Lab.

REFERENCES

[1] OpenAI, "ChatGPT," https://openai.com/blog/chatgpt, 2022, accessed: 2026-04-07.

[2] Anthropic, "Claude Code," https://www.anthropic.com/claude-code, 2025, accessed: 2026-04-07.

[3] OpenAI, "Codex," https://openai.com/index/introducing-gpt-5-3-codex/, 2026, accessed: 2026-04-07.

[4] F. F. Xu, Y. Song, B. Li, Y. Tang, K. Jain, M. Bao *et al.*, "Theagentcompany: Benchmarking LLM agents on consequential real world tasks," in *The Thirty-ninth Annual Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2026.

[5] X. Liu, H. Yu, H. Zhang, Y. Xu, X. Lei, H. Lai *et al.*, "Agentbench: Evaluating LLMs as agents," in *The Twelfth International Conference on Learning Representations*, 2024.

[6] S. Zhou, F. F. Xu, H. Zhu, X. Zhou, R. Lo, A. Sridhar *et al.*, "Webarena: A realistic web environment for building autonomous agents," in *The Twelfth International Conference on Learning Representations*, 2024.

[7] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. R. Narasimhan *et al.*, "React: Synergizing reasoning and acting in language models," in *The Eleventh International Conference on Learning Representations*, 2023.

[8] J. Yang, C. E. Jimenez, A. Wettig, K. Lieret, S. Yao, K. R. Narasimhan *et al.*, "SWE-agent: Agent-computer interfaces enable automated software engineering," in *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

[9] C. E. Jimenez, J. Yang, A. Wettig, S. Yao, K. Pei, O. Press *et al.*, "SWE-bench: Can language models resolve real-world github issues?" in *The Twelfth International Conference on Learning Representations*, 2024.

[10] V. Srivatsa, Z. He, R. Abhyankar, D. Li, and Y. Zhang, "Preble: Efficient distributed prompt scheduling for LLM serving," in *The Thirteenth International Conference on Learning Representations*, 2025.

[11] Y. Wang, Y. Chen, Z. Li, X. Kang, Y. Fang, Y. Zhou *et al.*, "Burstgpt: A real-world workload dataset to optimize llm serving systems," in *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2*, 2025, p. 5831–5841.

[12] Y. Xiang, X. Li, K. Qian, Y. Zhang, W. Yu, E. Zhai *et al.*, "ServeGen: Workload characterization and generation of large language model serving in production," in *23rd USENIX Symposium on Networked Systems Design and Implementation (NSDI 26)*, 2026, pp. 1845–1859.

[13] R. Qin, Z. Li, W. He, J. Cui, F. Ren, M. Zhang *et al.*, "Mooncake: Trading more storage for less computation — a KVCache-centric architecture for serving LLM chatbot," in *23rd USENIX Conference on File and Storage Technologies (FAST 25)*, 2025, pp. 155–170.

[14] A. Agrawal, H. Qiu, J. Chen, Í. Goiri, C. Zhang, R. Shahid *et al.*, "No request left behind: Tackling heterogeneity in long-context llm inference with medha," 2025.

[15] A. Agrawal, N. Kedia, A. Panwar, J. Mohan, N. Kwatra, B. S. Gulavani *et al.*, "Taming throughput-latency tradeoff in llm inference with sarathi-serve," in *Proceedings of the 18th USENIX Conference on Operating Systems Design and Implementation*, 2024.

[16] C. Hooper, S. Kim, H. Mohammadzadeh, M. W. Mahoney, Y. S. Shao, K. Keutzer *et al.*, "Kvquant: towards 10 million context length llm inference with kv cache quantization," in *Proceedings of the 38th International Conference on Neural Information Processing Systems*, 2024.

[17] G. Xiao, Y. Tian, B. Chen, S. Han, and M. Lewis, "Efficient streaming language models with attention sinks," in *The Twelfth International Conference on Learning Representations*, 2024.

[18] Z. Zhang, Y. Sheng, T. Zhou, T. Chen, L. Zheng, R. Cai *et al.*, "H2o: Heavy-hitter oracle for efficient generative inference of large language models," in *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

[19] J.-H. Kim, J. Kim, S. Kwon, J. W. Lee, S. Yun, and H. O. Song, "KVzip: Query-agnostic KV cache compression with context reconstruction," in *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

[20] Y. Li, Y. Huang, B. Yang, B. Venkitesh, A. Locatelli, H. Ye *et al.*, "Snapkv: Llm knows what you are looking for before generation," in *Proceedings of the 38th International Conference on Neural Information Processing Systems*, 2024.

[21] Z. Cai, Y. Zhang, B. Gao, Y. Liu, Y. Li, T. Liu *et al.*, "PyramidKV: Dynamic KV cache compression based on pyramidal information funneling," in *Second Conference on Language Modeling*, 2025.

[22] M. Kim, K. Shim, J. Choi, and S. Chang, "Infinipot: Infinite context processing on memory-constrained llms," in *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, 2024, pp. 16 046–16 060.

[23] A. Zweiger, X. Fu, H. Guo, and Y. Kim, "Fast KV compaction via attention matching," in *Forty-third International Conference on Machine Learning*, 2026.

[24] Y. Sheng, L. Zheng, B. Yuan, Z. Li, M. Ryabinin, B. Chen *et al.*, "Flexgen: high-throughput generative inference of large language models with a single gpu," in *Proceedings of the 40th International Conference on Machine Learning*, 2023.

[25] W. Lee, J. Lee, J. Seo, and J. Sim, "Infinigen: efficient generative inference of large language models with dynamic kv cache management," in *Proceedings of the 18th USENIX Conference on Operating Systems Design and Implementation*, 2024.

[26] D. Liu, M. Chen, B. Lu, H. Jiang, Z. Han, Q. Zhang *et al.*, "Retrievalattention: Accelerating long-context LLM inference via vector retrieval," in *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

[27] R. Chen, Z. Wang, B. Cao, T. Wu, S. Zheng, X. Li *et al.*, "Arkvale: efficient generative llm inference with recallable key-value eviction," in *Proceedings of the 38th International Conference on Neural Information Processing Systems*, 2024.

[28] Y. Chen, J. Zhang, B. Lu, Q. Zhang, C. Zhang, J. Liu *et al.*, "Retroinfer: A vector storage engine for scalable long-context llm inference," *Proc. VLDB Endow.*, p. 1016–1031, 2026.

[29] D. Quinn, E. E. Yücel, J. Kim, J. F. Martínez, and M. Alian, "Longsight: Compute-enabled memory to accelerate large-context llms via sparse attention," in *Proceedings of the 58th IEEE/ACM International Symposium on Microarchitecture*, 2025, p. 34–48.

[30] D. Kim, M. Lee, J. Kim, H. Kwon, H. Jeong, S.-S. Park *et al.*, "Scalable processing-near-memory for 1m-token llm inference: Cxl-enabled kv-cache management beyond gpu limits," in *2025 34th International Conference on Parallel Architectures and Compilation Techniques (PACT)*. IEEE Press, 2025, p. 1–13.

[31] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia *et al.*, "Chain-of-thought prompting elicits reasoning in large language models," in *Proceedings of the 36th International Conference on Neural Information Processing Systems*, 2022.

[32] X. Ma and D. Patterson, "Challenges and research directions for large language model inference hardware," *Computer*, vol. 59, no. 5, pp. 55–64, 2026.

[33] Z. Yuan, Y. Shang, Y. Zhou, Z. Dong, Z. Zhou, C. Xue *et al.*, "Llm inference unveiled: Survey and roofline model insights," 2024.

[34] T. Dao, D. Y. Fu, S. Ermon, A. Rudra, and C. Ré, "Flashattention: fast and memory-efficient exact attention with io-awareness," in *Proceedings of the 36th International Conference on Neural Information Processing Systems*, 2022.

[35] M. Rhee, J. Sim, T. Ahn, S. Lee, D. Yoon, E. Kim *et al.*, "HPU: High-bandwidth processing unit for scalable, cost-effective llm inference via gpu co-processing," 2025.

[36] R. Pope, S. Douglas, A. Chowdhery, J. Devlin, J. Bradbury, J. Heek *et al.*, "Efficiently scaling transformer inference," in *MLSys*. mlsys.org, 2023.

[37] A. Zhao and J. Liu, "Heterogeneous computing: The key to powering the future of ai agent inference," *Computer*, vol. 59, no. 4, pp. 165–171, 2026.

[38] M. Kim, S. Hong, R. Ko, S. Choi, H. Lee, J. Kim *et al.*, "Oaken: Fast and efficient llm serving with online-offline hybrid kv cache quantization," in *Proceedings of the 52nd Annual International Symposium on Computer Architecture*, 2025, p. 482–497.

[39] P. G. Recasens, F. Agullo, Y. Zhu, C. Wang, E. K. Lee, O. Tardieu *et al.*, "Mind the memory gap: Unveiling GPU bottlenecks in large-batch LLM inference," in *CLOUD*. IEEE, 2025, pp. 277–287.

[40] H. Zhang, P. Patel, A. Ning, and D. Wentzlaff, "Spad: Specialized prefill and decode hardware for disaggregated llm inference," 2025. [Online]. Available: https://arxiv.org/abs/2510.08544

[41] G. Xiao, J. Tang, J. Zuo, junxian guo, S. Yang, H. Tang *et al.*, "Duoattention: Efficient long-context LLM inference with retrieval and streaming heads," in *The Thirteenth International Conference on Learning Representations*, 2025.

[42] S.-S. Park, K. Kim, J. So, J. Jung, J. Lee, K. Woo *et al.*, "An lpddr-based cxl-pnm platform for tco-efficient inference of transformer-based large

language models," in *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2024, pp. 970–982.

[43] L. Liu, S. Zhao, B. Li, H. Ren, Z. Xu, M. Wang *et al.*, "Make llm inference affordable to everyone: Augmenting gpu memory with ndp-dimm," in *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2025, pp. 1751–1765.

[44] S. Choi, M. Rhee, E. Kim, K. Shin, Y. Joo, and H. Kim, "Pnm meets sparse attention: Enabling multi-million tokens inference at scale," *IEEE Computer Architecture Letters*, vol. 24, no. 2, pp. 353–356, 2025.

[45] R. Gong, S. Bai, S. Wu, Y. Fan, Z. Wang, X. Li *et al.*, "Past-future scheduler for llm serving under sla guarantees," in *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, 2025, p. 798–813.

[46] A. Chung, Y. Zhang, K. Lin, A. Rawal, Q. Gao, and J. Chai, "Evaluating long-context reasoning in llm-based webagents," in *NeurIPS 2025 Workshop on Bridging Language, Agent, and World Models for Reasoning and Planning*, 2025.

[47] S. Nayab, G. Rossolini, M. Simoni, A. Saracino, G. Buttazzo, N. Manes *et al.*, "Concise thoughts: Impact of output length on llm reasoning and cost," *Information Sciences*, vol. 757, p. 123939, 2026.

[48] T. Liu, Z. Wang, J. Miao, I.-H. Hsu, J. Yan, J. Chen *et al.*, "Budget-aware tool-use enables effective agent scaling," in *Third Conference on Language Modeling*, 2026.

[49] K. Huang, S. Liu, X. Hu, T. Xu, L. Bao, and X. Xia, "Reasoning efficiently through adaptive chain-of-thought compression: A self-optimizing framework," 2026. [Online]. Available: https://arxiv.org/abs/2509.14093

[50] Z. Zheng, X. Ren, F. Xue, Y. Luo, X. Jiang, and Y. You, "Response length perception and sequence scheduling: An LLM-empowered LLM inference pipeline," in *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

[51] Y. Fu, S. Zhu, R. Su, A. Qiao, I. Stoica, and H. Zhang, "Efficient LLM scheduling by learning to rank," in *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

[52] R. Shahout, eran malach, C. Liu, W. Jiang, M. Yu, and M. Mitzenmacher, "DON't STOP ME NOW: EMBEDDING BASED SCHEDULING FOR LLMS," in *The Thirteenth International Conference on Learning Representations*, 2025.

[53] H. Xie, Y. Chen, L. Wang, L. Hu, and D. Wang, "Predicting LLM output length via entropy-guided representations," in *The Fourteenth International Conference on Learning Representations*, 2026.

[54] S. Choi, J. Goo, E. Jeon, M. Yang, and M. Jang, "Elis: Efficient llm iterative scheduling system with response length predictor," 2025.

[55] H. Qiu, W. Mao, A. Patke, S. Cui, S. Jha, C. Wang *et al.*, "Efficient interactive llm serving with proxy model-based sequence length prediction," 2024.

[56] Y. Jin, C.-F. Wu, D. Brooks, and G.-Y. Wei, "S3: increasing gpu utilization during generative inference for higher throughput," in *Proceedings of the 37th International Conference on Neural Information Processing Systems*, 2023.

[57] M. Xiao, A. Wang, Q. Hu, Z. Miao, H. Shen, L. Wang *et al.*, "Can llms track their output length? a dynamic feedback mechanism for precise length regulation," 2026.

[58] X. Pan, E. Li, Q. Li, S. Liang, Y. Shan, K. Zhou *et al.*, "Instinfer: In-storage attention offloading for cost-effective long-context llm inference," 2024.

[59] W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu *et al.*, "Efficient memory management for large language model serving with pagedattention," in *Proceedings of the 29th Symposium on Operating Systems Principles*, 2023, p. 611–626.

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

[61] Y. Zhong, S. Liu, J. Chen, J. Hu, Y. Zhu, X. Liu *et al.*, "Distserve: disaggregating prefill and decoding for goodput-optimized large language model serving," in *Proceedings of the 18th USENIX Conference on Operating Systems Design and Implementation*, 2024.

[62] L. Pekelis, M. Feil, F. Moret, M. Huang, and T. Peng, "Llama 3 gradient: A series of long context models," 2024. [Online]. Available: https://gradient.ai/blog/scaling-rotational-embeddings-for-long-context-language-models

[63] C.-P. Hsieh, S. Sun, S. Kriman, S. Acharya, D. Rekesh, F. Jia *et al.*, "RULER: What's the real context size of your long-context language models?" in *First Conference on Language Modeling*, 2024.

[64] Y. Liu, Y. Cheng, J. Yao, Y. An, X. Chen, S. Feng *et al.*, "Lmcache: An efficient kv cache layer for enterprise-scale llm inference," 2025.