SuperInfer: SLO-Aware Rotary Scheduling and Memory Management for LLM Inference on Superchips

Jiahuan Yu [1] **Mingtao Hu** [1] **Zichao Lin** [1] **Minjia Zhang** [1]

Abstract

Large Language Model (LLM) serving faces a fundamental tension between stringent latency Service Level Objectives (SLOs) and limited GPU memory capacity. When high request rates exhaust the KV cache budget, existing LLM inference systems often suffer severe head-of-line (HOL) blocking. While prior work explored PCIe-based offloading, these approaches cannot sustain responsiveness under high request rates, often failing to meet tight Time-To-First-Token (TTFT) and Time-Between-Tokens (TBT) SLOs. We present SuperInfer, a high-performance LLM inference system designed for emerging Superchips (e.g., NVIDIA GH200) with tightly coupled GPU-CPU architecture via NVLink-C2C. SuperInfer introduces RotaSched, the first proactive, SLO-aware rotary scheduler that rotates requests to maintain responsiveness on Superchips, and DuplexKV, an optimized rotation engine that enables full-duplex transfer over NVLink-C2C. Evaluations on GH200 using various models and datasets show that SuperInfer improves TTFT SLO attainment rates by up to 74.7% while maintaining comparable TBT and throughput compared to state-of-the-art systems, demonstrating that SLO-aware scheduling and memory co-design unlocks the full potential of Superchips for responsive LLM serving. Code is available in `https://github.com/Supercomputing-System-AI-Lab/SuperInfer`.

1 Introduction

Large Language Models (LLMs) (Achiam et al., 2023; Touvron et al., 2023; Liu et al., 2024a; Bai et al., 2023) increasingly power latency-critical applications. These services demand strict latency *Service Level Objectives* (SLOs), particularly *Time-To-First-Token* (TTFT) and *Time-Between-Tokens* (TBT), to ensure responsive user interactions. Consequently, building high-performance LLM serving systems capable of meeting SLOs while maintaining high throughput has become a critical research challenge.

The key difficulty lies in memory-intensive inference. Each request maintains an ever-growing key-value (KV) cache during autoregressive generation (Radford et al., 2019). Under high request rates, this cumulative KV cache quickly exhausts GPU memory, leading to severe *head-of-line* (HOL) blocking and SLO violations. While prior SLO-aware schedulers (Agrawal et al., 2024; Fu et al., 2024; Hong et al., 2025) reduce queuing delays via priority reordering, they are ultimately constrained by limited on-device memory, failing to serve requests whose KV cache no longer fits.

To overcome this bottleneck, a line of work explores *offloading*, which swaps KV cache or model parameters to CPU memory (Sheng et al., 2023; Jiang et al., 2024c; Hu et al., 2024). While expanding effective memory capacity, existing techniques suffer two critical limitations. First, most of them are SLO-unaware, reacting to memory pressure rather than latency urgency, often causing severe TBT SLO violations for requests stuck in swapped queues (§ 3.1). Second, they are typically designed for PCIe architectures, whose limited bandwidth (∼32-64GB/s) makes swapping too slow to alleviate HOL blocking under high load (§ 3.2).

The emergence of GPU-CPU tightly-coupled Superchips, such as the NVIDIA GH200 (NVIDIA, 2024), fundamentally shifts this landscape. GH200 integrates a Hopper GPU and a Grace CPU via NVLink-C2C, a cache-coherence interconnect with up to 900GB/s bandwidth, an order of magnitude higher than PCIe. While initial studies explore GH200 for LLM serving (Xu et al., 2024) and training (Lian et al., 2025), its full potential remains untapped. Specifically, directly porting existing PCIe-based offloading mechanisms to GH200 yields unexpectedly poor results due to <5% utilization of C2C bandwidth (§ 3.3). This counterintuitive result suggests the bottleneck lies in the software stack instead of hardware. What prevents today's serving stacks from exploiting the full potential of Superchips, and what design principles are needed to close this gap?

[1]Siebel School of Computing and Data Science, University of Illinois Urbana-Champaign, Champaign, USA. Correspondence to: Jiahuan Yu <jiahuan2@illinois.edu>, Mingtao Hu <mingtao4@illinois.edu>, Zichao Lin <zichaol3@illinois.edu>, Minjia Zhang <minjiaz@illinois.edu>.

Proceedings of the 9th MLSys Conference, Bellevue, WA, USA, 2026. Copyright 2026 by the author(s).

Our in-depth study of GH200's memory hierarchy and offloading behavior reveals fundamental software-hardware mismatches preventing current serving stacks from leveraging its full potential (§ 4.3.1). Guided by our observations, we introduce SuperInfer, a high-performance, SLO-aware LLM serving system optimized for Superchips. SuperInfer employs two co-designed techniques for responsive, full-duplex offloading: (1) **RotaSched** (§ 4.2), an OS-inspired rotary scheduler that, instead of using passive preemption for out-of-memory error prevention, introduces a novel transient *rotary* state and *active rotation* to proactively rotate requests between HBM and DRAM based on their SLO progress. (2) **DuplexKV** (§ 4.3), a high-performance KV cache rotation engine co-designed to support efficient request rotations over NVLink-C2C. It maximizes C2C bandwidth utilization by enabling full-duplex data-race-free transfers, merging fragmented KV cache segments into large efficient batches, and overlapping transfers with model execution.

We evaluate SuperInfer across various models and workloads on GH200. Under high request rates, SuperInfer substantially improves end-to-end SLO attainment, achieving up to 74.7% higher TTFT success rates than state-of-the-art systems, while maintaining comparable throughput and TBT. At low request rates, where memory is sufficient, SuperInfer matches baseline performance, confirming its benefits stem from efficient, SLO-aware memory management on Superchips. These improvements demonstrate that Superchips, paired with co-designed software, can effectively mitigate HOL blocking and enable responsive LLM serving.

In summary, this paper makes the following contributions: **(1)** We identify the limitations of existing LLM serving systems with PCIe-based offloading on Superchips, and analyze why they fail to utilize NVLink-C2C bandwidth and achieve SLO responsiveness. **(2)** We present SuperInfer, the first SLO-aware LLM serving system co-designed for Superchips, which consists of (i) RotaSched, an OS-inspired scheduler using active rotation guided by Virtual Lag Time (VLT), and (ii) DuplexKV, a full-duplex KV cache rotation engine that eliminates C2C under-utilization. **(3)** We demonstrate that SuperInfer achieves up to 74.7% higher TTFT SLO attainment rate under high request rates while maintaining comparable throughput and TBT latency.

2 Background and Related Work

2.1 LLM Inference and Offloading over PCIe GPUs

During LLM autoregressive generation, each request's KV cache grows linearly with sequence length (Radford et al., 2019), making GPU memory the dominant bottleneck: even high-end GPUs can hold only a limited number of concurrent requests. Researchers have explored sophisticated mechanisms to address this challenge.

Early systems like DeepSpeed-Inference (Aminabadi et al., 2022) and FlexGen (Sheng et al., 2023) treat host memory as an extension of GPU memory to fit models beyond on-device capacity. While enabling single-GPU inference for large models, their static offloading leads to high data transfer latency during dynamic serving workloads. Later, PagedAttention (Kwon et al., 2023) proposes paging-based memory managers that organize KV cache into small, fixed-size blocks, significantly reducing memory fragmentation and improving memory utilization. This design has become the de facto technique in recent LLM serving frameworks (Kwon et al., 2023; Zheng et al., 2024). However, paging scatters KV cache across non-contiguous small regions, making GPU-CPU transfer expensive. Subsequent systems attempt to overlap the transfer overheads with computation (Jiang et al., 2024b; Luo et al., 2025; Cao et al., 2025; Yu et al., 2024; Kim et al., 2025; Hu et al., 2024; Qin et al., 2025). FastDecode (He & Zhai, 2024), NEO (Jiang et al., 2024c), HeteGen (Zhao et al., 2024), CachedAttention (Gao et al., 2024), FlashGen (Jeong & Ahn, 2025), and NanoFlow (Zhu et al., 2025) propose offloading KV cache or part of the computation from GPU to CPU or SSDs, effectively increasing the batch size and therefore inference throughput. Although effective, these designs remain PCIe-bound, and do not exploit the high-bandwidth, full-duplex C2C links in emerging Superchips.

To overcome the PCIe bottleneck, Aqua (Vijaya Kumar et al., 2025) offloads KV cache to other GPUs via inter-GPU NVLink, but it relies on limited spare peer GPU memory rather than large CPU memory on Superchips. Alternatively, some systems reduce KV cache volume itself instead of transfer pipeline optimizations (Chen et al., 2025b; Song et al., 2024). For example, InfiniGen (Lee et al., 2024) prunes KV entries via dynamic importance estimation and transfers only a subset of KVs for each request. CacheGen (Liu et al., 2024b) compresses KV cache via layer-wise quantization and arithmetic coding. These methods reduce offloading overhead but are lossy, making them hard to generalize to unseen tasks and requests.

2.2 Offloading over Emerging Tightly Coupled GPU–CPU Architectures

Recent Superchip architectures (e.g., NVIDIA GH200, AMD MI300A) tightly integrate GPU and CPU via high-performance interconnects, offering an order of magnitude higher bandwidth than traditional PCIe. For example, GH200's NVLink-C2C provides 900GB/s bidirectional bandwidth (NVIDIA, 2024). Several recent studies have benchmarked (Fusco et al., 2024; Schieffer et al., 2024) or explored programming Superchips (Vellaisamy et al., 2025). SuperOffload (Lian et al., 2025) further examines Hopper, Grace, C2C joint optimizations, demonstrating potential for large-scale LLM training. However, few prior

systems explicitly target LLM inference on Superchips. The most relevant work is Pie (Xu et al., 2024), which enables GPU-CPU KV cache spilling on GH200 and adapts memory allocation on-the-fly. However, Pie is not optimized for tight latency SLOs under high load, nor does it apply SLO-aware scheduling. In contrast, SuperInfer introduces an active rotary scheduler and full-duplex transfer engine, jointly addressing SLO-awareness and C2C link utilization.

2.3 Scheduling for Latency-Aware Inference

While widely used techniques like continuous batching (Yu et al., 2022) improve throughput, they lack explicit latency control. Several works target fairness among requests or clients (Sheng et al., 2024; Wei et al., 2025), which complement but do not directly optimize for latency SLOs.

There has been a growing number of works explicitly managing request latency. Sarathi-Serve (Agrawal et al., 2024) introduces chunked prefill and a dynamic batching policy to improve SLO attainment under dynamic workloads, but it assumes GPU-residency of all requests. Ao et al. (2025) formulate LLM serving as an online scheduling problem and proposes WAIT/Nested-WAIT algorithms to balance throughput and latency under memory constraints. While effective within single-GPU memory, these works do not address latency management of LLM serving for heterogeneous GPU-CPU hierarchies or offloading. A larger set of systems mitigates latency via queue reordering and request prioritization. For example, SSJF (Qiu et al., 2024) and LTR (Fu et al., 2024) use length prediction to approximate Shortest-Job-First (SJF) or Shortest-Remaining-Time-First (SRTF) scheduling, reducing HOL blocking and improving responsiveness of short requests. Others treat SLO attainment as a search or scheduling optimization problem, such as SOLA (Hong et al., 2025), FastServe (Wu et al., 2023), SHEPHERD (Zhang et al., 2023), or target multi-SLO serving for mixed request types (Zhang et al., 2025; Huang et al., 2025; Borui et al., 2025; Li et al., 2025). Although effective, they are fundamentally limited by GPU memory capacity.

Alternatively, several works aim to reduce SLO violations by saving GPU memory or increasing utilization (Chen et al., 2025c; Patke et al., 2024). LightLLM (Gong et al., 2025) estimates future KV cache usage to avoid harmful preemption, which wastes GPU memory, but it is also limited by GPU memory capacity. TokenFlow (Chen et al., 2025a) and Select-N (Ma et al., 2025) explore SLO-aware scheduling with offloading to balance host memory usage and latency, which are PCIe-based and thus limited by its bandwidth.

3 Observations

This section presents the motivational studies. We first identify the SLO-aware offloading as a challenge (§ 3.1), then examine the limitations of PCIe for offloading (§ 3.2), finally analyze the unexpected NVLink-C2C bandwidth under-utilization of existing offloading techniques (§ 3.3).

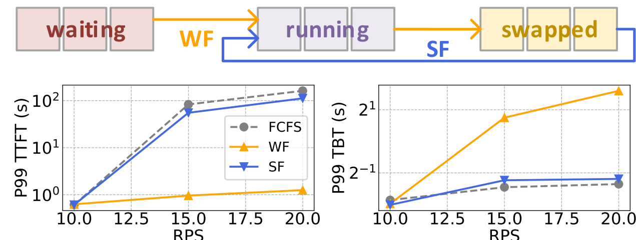

Figure 1. Two static offloading policies: Waiting-First (WF) and Swapped-First (SF), and comparison of their P99 TTFT and TBT to FCFS under varying request rates (Qwen2.5-32B, ShareGPT).

3.1 SLO-Aware Offloading as a Challenge

While sophisticated scheduling techniques exist, they cannot fundamentally overcome the GPU memory pressure imposed by hardware constraints (more details in Appendix A). A natural idea to mitigate memory pressure is offloading, which swaps inactive requests' KV cache to CPU memory. However, this introduces a challenge for SLO-aware serving (Fig. 1): prioritize waiting requests (*Waiting-First*, WF) by preempting running ones, or prioritize resuming swapped requests (*Swapped-First*, SF)? These static, SLO-unaware policies, common in production frameworks (Zheng et al., 2024; Kwon et al., 2023), fail to strike a balance. As Fig. 1 shows, compared to *First-Come-First-Serve* (FCFS), WF favors new arrivals, reducing TTFT but significantly increasing TBT, as running requests are paused for a long time during generation. Conversely, SF protects TBT but degrades to FCFS-like performance, failing to fully exploit the potential of offloading, as swapped requests are prioritized to resume immediately, thus swap space is underutilized. Consequently, neither policy effectively balances TTFT and TBT.

Insight #1: Offloading shows promise for alleviating memory pressure, but making it SLO-aware requires determining *which* requests to swap or run. Static policies inevitably bias toward one SLO at the expense of the other. An effective design must mitigate this issue via proactive scheduling.

3.2 Responsiveness Bottleneck Caused by Low Swap Bandwidth over PCIe

Even with ideal scheduling, SLO-aware offloading succeeds only if requests can be swapped fast enough between GPU and CPU. In practice, offloading responsiveness is constrained by the effective *swap bandwidth*, i.e., the amount of KV cache transferred per unit time, which, on PCIe-based GPUs, is fundamentally limited by the bandwidth of PCIe.

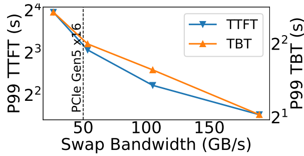

Figure 2. P99 TTFT and TBT latency vs. swap bandwidth for vLLM with offloading. (Qwen2.5-32B, ShareGPT, RPS=20).

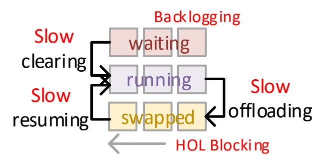

Figure 3. Limitations of low swap bandwidth for offloading: slow clearing of backlogged waiting requests and new HOL blocking in the swapped queue.

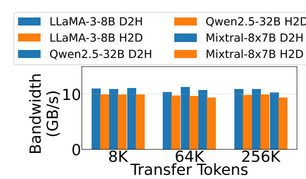

Figure 4. Measured bidirectional CPU-GPU KV cache transfer bandwidth of vLLM for three different models.

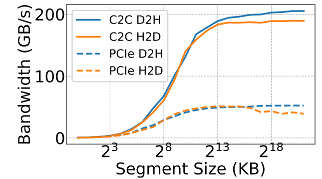

Figure 5. NVLink-C2C (GH200) vs. PCIe Gen5x16 (H200) full-duplex bandwidth by segment size.

As Fig. 2 shows, increasing swap bandwidth beyond the PCIe Gen5x16 uni-directional limit significantly reduces both TTFT and TBT. Tail TTFT is primarily caused by waiting queue delay, and tail TBT primarily results from the duration that requests are in the swapped state. Thus, PCIe's low swap bandwidth creates two major obstacles to reducing tail latencies (Fig. 3). The first, it leads to *request backlogging*, i.e., the accumulation of new arriving requests in the waiting queue due to GPU memory exhausting at high request rates. Low swap bandwidth makes clearing this backlog through offloading sluggish; paused requests remain in GPU memory longer due to slow offloading, while new arrivals continue to queue up, increasing TTFT SLO violations. Second, when a large number of requests are swapped, low swap bandwidth stalls request resumption due to slow KV cache transfer, leading to new HOL blocking in the swapped queue and limited responsiveness to TBT SLO violations. Consequently, PCIe fundamentally limits ability of offloading-based LLM serving systems to respond to dynamic SLO violations when system load is high.

Insight #2: Offloading-based SLO-aware LLM inference is fundamentally limited by low bandwidth of PCIe. It constrains the swap bandwidth and responsiveness, leading to slow clearing of backlogged waiting requests and new HOL blocking in the swapped queue.

3.3 Superchip Offers Opportunity but Not for Free

In principle, the GH200's high NVLink-C2C bandwidth could eliminate the swap bandwidth bottleneck described above, enabling near-instantaneous request preemption and resumption. However, our profiling of widely-used LLM serving systems shows severe bandwidth under-utilization.

As Fig. 4 shows, vLLM's offloading engine achieves only ∼10GB/s of effective bandwidth on GH200 (<5% of the theoretical peak) across three models, regardless of the total transfer token number. To understand this gap, we perform a fine-grained full-duplex transfer bandwidth characterization across different segment sizes for GH200 (see Appendix B for method). As Fig. 5 shows, C2C bandwidth saturates at ∼200+200GB/s. Further investigation shows this limit stems not from the C2C itself, but from the Grace DRAM subsystem. Each NUMA node in Grace provides 384GB/s of DRAM bandwidth (NVIDIA, 2024). Therefore, memory-intensive transfers can saturate DRAM bandwidth before reaching the C2C link's 450+450 GB/s theoretical peak. Moreover, Grace DRAM is half-duplex, which explains why concurrent device-to-host (D2H) and host-to-device (H2D) transfers are limited by ∼384GB/s in total, while individual D2H/H2D can each achieve higher bandwidth.

Despite these findings, the large disparity between achievable (∼200GB/s) and achieved (∼10GB/s) bandwidth raises a natural question: what prevents current LLM offloading mechanisms from fully harnessing GH200's NVLink-C2C?

Insight #3: Superchips unlock the bandwidth essential for responsive offloading, but current LLM serving stacks exploit <5% of it. Fully exploiting its architecture requires co-designed scheduling and memory-management mechanism for high-performance, SLO-aware LLM inference.

4 System Design

Based on insights from § 3, we propose SuperInfer, a high-performance LLM serving system optimized for memory-intensive, latency-sensitive tasks on GH200. This section details SuperInfer's overview (§ 4.1) and its two key components: RotaSched (§ 4.2) and DuplexKV (§ 4.3).

4.1 Overall Architecture

Fig. 6 illustrates the overall architecture of SuperInfer, which is composed of two co-designed components:

RotaSched (§ 4.2): An OS-inspired, SLO-aware preemptive scheduler implementing *active rotation* to fully utilize large swap space. Guided by a novel *Virtual Lag Time* (VLT) metric, it employs *Largest-VLT-First* (LVF) policy to prioritize lagging requests vulnerable to SLO violations, while preempting long-running ones into the *rotary* state for later resumption. This ensures fast request execution rotation with offloading and high responsiveness for SLO violations.

DuplexKV (§ 4.3): A high-performance KV cache rotation

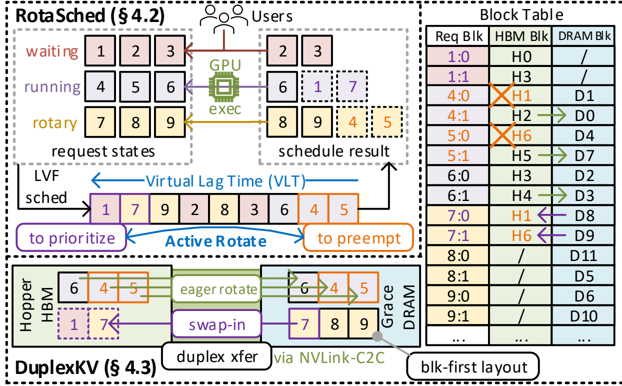

Figure 6. Overall architecture of SuperInfer. RotaSched maintains the requests states and perform Largest VLT First (LVF) scheduling based on Virtual Lag Time (VLT). DuplexKV manages the KV cache allocation across Hopper HBM and Grace DRAM with a block table, and transfers KV cache for swap-in and eager block rotation in a full-duplex manner. Numbers in squares denote different requests. "X:Y" denotes KV block Y of request X, "H:W" and "D:Z" denotes KV cache block in HBM and DRAM, respectively.

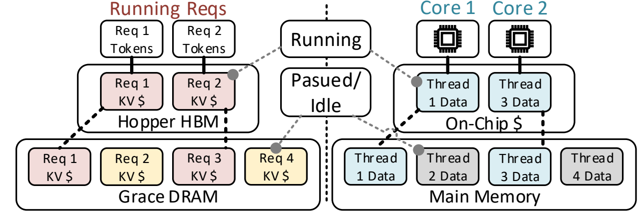

Figure 7. Analogy: LLM serving stack on GH200 (left) vs. OS on CPU (right). Requests → threads, Hopper HBM → on-chip cache, Grace DRAM → main memory, KV cache → thread data.

engine optimized for Superchips, enabling RotaSched's frequent, large-volume offloading. It achieves high bandwidth and low overhead via eager block rotation for data-race-free, full-duplex transfers, transformed KV cache layout with batched transfer kernel launches to merge small segments into efficient large transfers, and a cross-iteration pipeline to overlap transfers with computation.

4.2 RotaSched: OS-Inspired Rotary Scheduler for SLO-Aware Offloading

4.2.1 From Passive Preemption to Proactive Rotation

Most existing LLM serving systems (Kwon et al., 2023; Zheng et al., 2024) employ *passive preemption*: later-arrived requests are paused (by offloading/discarding their KV cache) only when KV cache demand exceeds available GPU memory. While preventing out-of-memory (OOM), it fails to actively manage SLOs. Crucially, no preemption occurs if GPU memory suffices for running requests, even if many waiting requests are close to violating their TTFT SLOs.

The GH200's tightly-coupled design offers a unique opportunity to rethink LLM latency and memory management. As Fig. 7 illustrates, an LLM serving stack on GH200 is analogous to an OS on CPU: requests act as threads, Hopper HBM as on-chip cache, Grace DRAM as main memory, and KV cache as thread data. This analogy inspires a shift in perspective: from static allocation of GPU memory to dynamic scheduling of GPU execution. Operating systems employ preemptive, time-slicing schedulers like CFS (Wong et al., 2008) and EEVDF (Stoica & Abdel-Wahab, 1995) to rotate execution among numerous threads without starvation.

Based on this insight, SuperInfer adapts the *rotation* principle for LLM serving to mitigate SLO violations. Instead of treating preemption as a last resort to prevent OOM, SuperInfer *actively rotates* requests across two execution tiers based on their SLO status. It introduces a *rotary* state: a transient execution state where a request's progress is temporarily paused on GPU and its KV cache is swapped to the CPU, waiting for next rotation. This enables an LLM inference scheduling structure on Superchips analogous to OS time-slicing, but driven by SLO rather than fixed quanta, as introduced next. Fig. 6 shows request state transition in SuperInfer.

4.2.2 Virtual Lag Time (VLT)

While the rotary state enables to alternate requests between GPU and CPU, the key question is *when* rotation should occur. Simply relying on queue length or memory usage would fall back to passive, SLO-unaware scheduling.

A prominent lag-based thread scheduler in OS is *Earliest Eligible Virtual Deadline First* (EEVDF) (Stoica & Abdel-Wahab, 1995). It tracks a *lag* value that shows whether a thread has received its fair share of CPU time, and prioritizes those with the earliest virtual deadlines. While offering valuable inspirations for managing a large number of concurrent requests, directly applying it to LLM serving is infeasible for two reasons. First, unlike lightweight context switches in OS supported by hardware-managed cache fetching, LLM request rotation involves costly GPU-CPU KV cache transfer. Second, LLM serving must jointly manage two distinct, asymmetrically sensitive latency targets (TTFT and TBT).

Building on EEVDF's spirit, SuperInfer introduces a metric called *Virtual Lag Time* (VLT) to measure a request's deviation from its SLO progress. VLT serves as the scheduling currency of RotaSched: requests with higher positive VLT are prioritized for execution, while those with smaller negative VLT become candidates for rotation. The following

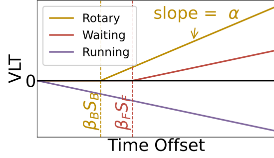

Figure 8. Visualization of VLT. The time offset refers to: $t_{\text{now}} - t_{\text{last}}$ (rotary), $t_{\text{now}} - t_{\text{arr}}$ (waiting), or $t_{\text{now}} - t_{\text{run}}$ (running).

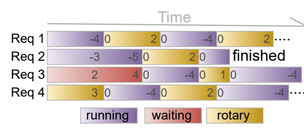

Figure 9. A conceptual example to show how LVF rotates executions. There are 4 requests in total, and HBM can only hold 2 requests. Numbers refer to VLT.

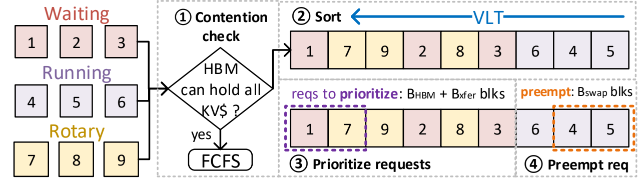

Figure 10. Demonstration of LVF scheduling algorithm. ①②③④ refer to steps described in § 4.2.3.

equation defines the VLT for each type of request:

$$VLT = \begin{cases} \alpha \cdot \text{ReLU}\left(t_{\text{now}} - t_{\text{last}} - \beta_B S_B\right), & \text{for rotary,} \\ \text{ReLU}\left(t_{\text{now}} - t_{\text{arr}} - \beta_F S_F\right), & \text{for waiting,} \\ -\left(t_{\text{now}} - t_{\text{run}}\right), & \text{for running,} \end{cases}$$

where S_B and S_F are the TTFT and TBT SLO, respectively. $\alpha \geq 0$ is sensitivity weight, $\beta_B, \beta_F \in \mathbb{R}$ are latency tolerance coefficients for TTFT and TBT, respectively. t_{now} is current system time, t_{last} is the time of last generated token, t_{arr} is request arrival time, t_{run} is the time a request begins in the running state. Fig. 8 visualizes VLT for three request types when $\alpha > 1$, $\beta_B > 0$, $\beta_F > 0$. For running requests, VLT is negative and decreases as they run, with smaller values indicating higher degree of "advance". Conversely, VLTs of waiting and rotary requests are initially zero when their inactive time is within tolerance. They turn positive once tolerance is exceeded, increasing further with longer waits, reflecting their degree of "lag".

VLT provides a mechanism to monitor request time usage and potential SLO violations, enabling the SLO-aware priorities. Intuitively, a larger VLT signifies more "lag," thus requires higher execution priority to mitigate SLO violations. Additionally, the adjustable parameters α, β_B, β_F offer flexibility for scenario-specific trade-offs. First, α defines the sensitivity ratio of TBT/TTFT to SLO violations; larger values indicate higher relative TBT sensitivity. Typically, TBT is more delay-sensitive (100ms is significant for TBT while manageable for TTFT), suggesting $\alpha \geq 1$. Second, β_B and β_F reflect SLO requirements and tolerances in scheduling, where larger values indicate higher tolerance for SLO violations, enabling scenario-specific configuration.

4.2.3 *Largest-VLT-First (LVF) Scheduling*

Building on VLT, we next introduce the *Largest-VLT-First* (LVF) scheduling policy, which leverages VLT to orchestrate proactive GPU-CPU request rotation. LVF prioritizes requests by VLT: those with higher positive VLT (indicating more vulnerable to SLO violations) are prioritized for execution, while those with lower negative VLT (indicating prolonged execution time) become preemption candidates to free HBM. Fig. 10 illustrates the LVF algorithm (Algorithm 1), which performs the following steps during each engine iteration: ① **Contention Check**: If HBM can hold KV cache of all requests, LVF skips subsequent steps and falls back to FCFS. ② **Sort**: LVF calculates VLTs for all requests and sorts them in descending order as a list $\mathcal{L}$. Waiting and rotary requests vulnerable to SLO violations are at the head. The tail holds running requests which are preemption candidates. ③ **Prioritize requests**: Given a transfer budget B_{xfer} and free HBM KV block number B_{HBM}, LVF picks rotary/waiting requests starting from the head of $\mathcal{L}$, until all $B_{\text{xfer}} + B_{\text{HBM}}$ blocks are used. Selected requests are prioritized for execution. ④ **Preempt requests**: B_{swap} extra blocks besides the current free HBM blocks are required to hold prioritized requests. LVF starts from the tail of $\mathcal{L}$ to preempt running requests until getting enough extra blocks.

This algorithm allows LVF to dynamically monitor potential SLO violations using VLT, and perform preemptive scheduling. By preempting (and swapping out) running requests, it makes HBM space for resuming SLO-vulnerable waiting or rotary requests. Thus, LVF enables the SLO-aware use of large swap spaces, avoiding the TBT violations in naive policies (§ 3.1). Fig. 9 shows a conceptual example of how LVF rotates execution among 4 requests when HBM can only hold 2 requests. Furthermore, the transfer budget B_{xfer} controls the number of blocks to be swapped each iteration, reflecting the swap bandwidth. NVLink-C2C enables a large B_{xfer} and a high swap bandwidth compared to PCIe-based systems, which is critical to system responsiveness for backlogged waiting requests, as discussed in § 3.2.

4.3 Efficient Offloading and Memory Management for Superchip NVLink-C2C

4.3.1 *Why NVLink-C2C Utilization Remains Low?*

As discussed in § 3.3, existing offloading mechanisms suffer from severe under-utilization on NVLink-C2C, achieving only ∼10GB/s effective throughput, far below the theoretical peak. To understand why, we examine how modern LLM serving frameworks manage KV cache.

Most recent LLM serving systems employ PagedAttention (Kwon et al., 2023) to manage KV cache. While eliminating memory fragmentation by allocating KV cache in fixed-sized blocks (e.g., 16 tokens per block), it also scatters

Algorithm 1 LVF Scheduling

Require: Request (running $\mathcal{Q}_{\text{R}}$, waiting $\mathcal{Q}_{\text{W}}$, rotary $\mathcal{Q}_{\text{S}}$), KV cache block number of request $\text{blk}(\cdot)$, transfer budget B_{xfer}, current free HBM block number B_{HBM}.
Ensure: Preempted requests $\mathcal{P}$, prioritized requests $\mathcal{R}$.

1: $\mathcal{P} \leftarrow \emptyset$, $\mathcal{R} \leftarrow \emptyset$, $B_{\text{left}} \leftarrow B_{\text{HBM}} + B_{\text{xfer}}$
2: $\mathcal{L} \leftarrow \mathcal{Q}_{\text{R}} \cup \mathcal{Q}_{\text{W}} \cup \mathcal{Q}_{\text{S}}$ ▷ All requests in a list
3: **if** $B_{\text{HBM}} \geq \sum_{r \in \mathcal{Q}_{\text{W}} \cup \mathcal{Q}_{\text{S}}} \text{blk}(r)$ **then** ▷ **Step ①**
4: **return** $\emptyset$, $\mathcal{Q}_{\text{W}} \cup \mathcal{Q}_{\text{S}}$ ▷ Fallback to FCFS
5: **end if**
6: Sort $\mathcal{L}$ in descending order of VLT. ▷ **Step ②**
7: **for** $r \in \mathcal{L}$ **do** ▷ Find prioritized requests ▷ **Step ③**
8: **if** $\text{VLT}(r) \geq 0$ & $\text{blk}(r) \leq B_{\text{left}}$ **then**
9: $\mathcal{R} \leftarrow \mathcal{R} \cup \{r\}$, $B_{\text{left}} \leftarrow B_{\text{left}} - \text{blk}(r)$
10: **end if**
11: **end for**
12: $B_{\text{swap}} \leftarrow B_{\text{xfer}} - B_{\text{left}}$ ▷ Extra block number ▷ **Step ④**
13: **for** $r \in \text{reversed}(\mathcal{L})$ **do** ▷ Find preempted request
14: **if** $\text{VLT}(r) < 0$ & $B_{\text{swap}} > 0$ **then**
15: $\mathcal{P} \leftarrow \mathcal{P} \cup \{r\}$, $B_{\text{swap}} \leftarrow B_{\text{swap}} - \text{blk}(r)$
16: **end if**
17: **end for**
18: **return** $(\mathcal{P}, \mathcal{R})$ ▷ Preempted and prioritized requests

each request's KV cache to non-contiguous GPU memory regions. Supposing a Transformer-based model has N_{L} layers, each maintaining their own KV cache. During inference, each layer's KV cache is divided into blocks (with KV of P tokens per block). KV entries within the same layer and block form the largest contiguous memory region, which we call a segment (Fig. 11). Segment size is $S_{\text{seg}} = P \times C$, where C denotes per-token KV size. Across N_{L} layers and N_{B} blocks, a request's full KV cache can be viewed as a layer-first 3D virtual tensor of shape $(N_{\text{L}}, N_{\text{B}}, S_{\text{seg}})$.

However, while KVs of the same segment are contiguous, segments from different layers or blocks are not, leading to non-contiguous memory layout. Since these segments are typically fine-grained and much smaller than transfer-efficient sizes, it results in poor C2C bandwidth utilization. For example, in Qwen2.5-32B ($N_{\text{L}} = 64$, $C = 4$, $P = 16$), each segment is $S_{\text{seg}} = 64$KB, while a full block's KV cache (all 64 layers) is 4MB. Thus, each 64KB segment must be transferred independently via a separate `cudaMemcpyAsync` kernel launch. As Fig. 5 shows, NVLink-C2C achieves high throughput ($\sim$200GB/s per direction) only when the transfer size $\geq$8MB. Below that threshold, bandwidth drops sharply, falling $<$10GB/s for $\leq$64KB segments. Therefore, while PagedAttention efficiently utilizes GPU memory, it paradoxically keeps the NVLink-C2C operating in its low-efficiency regime. Furthermore, the vast number of transfer operations also contributes to poor utilization. Each `cudaMemcpyAsync` is launched internally as a

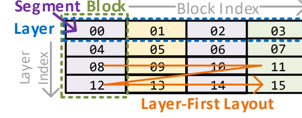

Figure 11. Layer, block, and segment structure of the PagedAttention KV cache. Colors denote different requests, and numbers denote relative segment addresses.

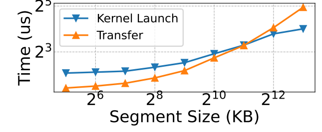

Figure 12. Kernel launch and transfer times for segments of different sizes via `cudaMemcpyAsync` (vLLM, Qwen2.5-32B, ShareGPT).

lightweight GPU kernel. When thousands of them are issued back-to-back, their accumulated launch overhead dominates execution time, further limiting effective bandwidth. Our profile on GH200 using NVLink-C2C (Fig. 12) confirms that the current offloading path under-utilizes NVLink-C2C also due to the software induced granularity and launch overhead. Specifically, launch time of `cudaMemcpyAsync` is larger than transfer time when $S_{\text{seg}} \leq 4$MB.

4.3.2 DuplexKV: Efficient KV Cache Rotation Engine

Our analysis shows that NVLink-C2C bandwidth underutilization stems not from hardware, but from thousands of small, non-contiguous `cudaMemcpyAsync` launches by current LLM serving stack. To fully exploit the C2C, SuperInfer introduces DuplexKV, a high-performance KV cache rotation engine that (i) performs data-layout transformation to reorganize fragmented KV segments into large, contiguous regions optimized for transfer; (ii) pipelines computation with simultaneous D2H/H2D transfers, maximizing utilization of NVLink-C2C's full-duplex bandwidth.

The challenge lies in swap-in/out dependencies. A straightforward bandwidth improvement is launching independent CUDA streams for concurrent H2D and D2H transfers. However, naively overlapping them introduces data races on shared HBM blocks. As Fig. 13 (left) shows, the destination HBM blocks for swap-in might coincide with the source blocks being freed by swap-out. This dependency forces the swap-in stream to wait for swap-out completion, causing serialization of both directions and C2C link underutilization. While many prior systems discuss overlapping transfer with computation (Gao et al., 2024; Jiang et al., 2024c), few explicitly examine overlapping of bidirectional H2D/D2H transfers; for example, vLLM (Kwon et al., 2023) and SGLang (Zheng et al., 2024) perform swap-in and swap-out serially. To our knowledge, no prior work has analyzed or addressed the data-race dependencies from bidirectional transfer concurrency in tightly coupled GPU-CPU systems like GH200. This motivates our new *eager block rotation* mechanism in SuperInfer, which eliminates these dependencies and enables full-duplex utilization of C2C.

Eager block rotation for data race free overlapping. To

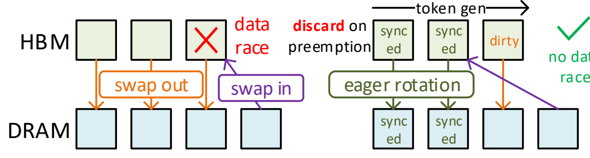

Figure 13. Left figure shows how data races occur when the swap-in destination block is the swap-out source block. Right figure shows our eager block rotation, which breaks the data dependency to enable concurrent swap-in and swap-out.

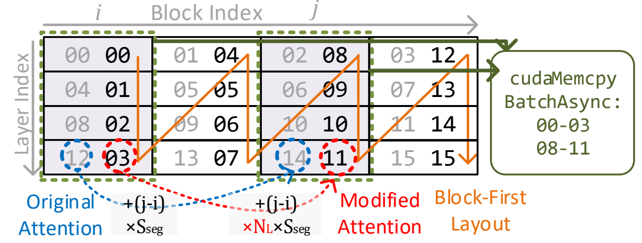

Figure 14. With block-first layout, small segments of the same block are merged into a larger contiguous region. Transfers of these regions can be done via a single `cudaMemcpyBatchAsync` kernel launch. Grey numbers are relative memory address for layer-first layout, black numbers are those for block-first layout.

decouple two streams, DuplexKV introduces an *eager block rotation* mechanism (Fig. 13 right). The key insight is that KV cache generation is incremental: each request appends new tokens to its designated block sequentially, and previously fully written blocks remain unchanged until request completes. Therefore, blocks are categorized: (i) *Dirty*: partially filled and updated during token generation, and (ii) *Synced*: fully filled and will not receive further writes until the request finishes. DuplexKV eagerly swap-out *synced* blocks from HBM and DRAM in the background, even before preemption. These early transfers mark the corresponding HBM blocks as synced. When the request is later preempted, only the last dirty block needs swap-out, while synced blocks in HBM can be discarded since valid copies already exist in DRAM. Crucially, this "discarded on preemption" mechanism relies on CPU-side duplication to maintain data correctness, ensuring that it imposes no additional HBM pressure. A lightweight block table (as shown in Fig. 6) tracks each block's state and residency.

Block-first layout and batched KV transfers. To address the inefficiency of small, scattered memory segments, we redesign the KV cache storage from *layer-first* to *block-first* ordering. As Fig. 14 shows, this layout makes all layers within the same block contiguous, merging N_{L} small per-layer segments into one large contiguous region (e.g., 64KB $\rightarrow$ 4MB for Qwen-2.5-32B). This transformation converts numerous fine-grained transfers into much reduced large DMA operations, which are in the high-bandwidth regime of NVLink-C2C. We also extend PagedAttention to adopt this block-first layout. As Fig. 14 shows, for original PagedAttention with layer-first KV cache layout, the stride between block-i and block-j is $(j-i)\cdot S_{\mathrm{seg}}$, while for block-first layout, it becomes $(j-i)\cdot N_L \cdot S_{\mathrm{seg}}$. We extend PagedAttention kernels to support this layout and stride, while preserving the paging abstraction. To further reduce kernel launch overhead, we merge individual `cudaMemcpyAsync` kernels of one direction (e.g., HBM $\rightarrow$ DRAM) into a single `cudaMemcpyBatchAsync` kernel. This batched invocation issues all transfer descriptors in one kernel launch, effectively eliminating per-kernel launch overhead.

Cross-iteration pipeline to overlap execution with rotation. To further reduce critical path latency, we also overlap schedule and KV cache rotation with model execution (e.g., LLM decoding), as Fig. 15 shows. It employs a cross-iteration pipeline: during iteration-t, GPU executes the batch prepared in iteration-$(t-1)$, while the scheduler and DuplexKV concurrently prepare the batch for iteration-$(t+1)$ on the host side. This pipeline hides data transfer and schedule latency behind compute, which minimizes GPU stall time and maximizing GPU utilization.

5 Evaluation

5.1 Evaluation Methodology

Models and Workloads. We evaluate SuperInfer over three models: LLaMA-3-8B (Dubey et al., 2024), Qwen2.5-32B (Yang et al., 2025) and Mixtral-8x7B (Jiang et al., 2024a). We use ShareGPT (ShareGPT Team, 2023) and LMSYS-Chat-1M (Zheng et al., 2023) datasets with controlled request arrival rates with sampled request arrival intervals following the Poisson distribution.

Metrics. We evaluate SuperInfer with the SLO attainment rate, defined as the percentage of requests that satisfy the SLO thresholds for TTFT and TBT.

Implementation. We implement SuperInfer in Python and C++ on top of vLLM (Kwon et al., 2023) (v0.6.6.post1), a widely used production-level LLM inference framework.

Hardware Testbeds. All experiments are performed on an NVIDIA GH200 NVL2 system. Each GH200 Superchip features 144GB of HBM and 480GB of DRAM. We utilize `numactl` to configure NUMA memory affinity, ensuring that all allocated memory resides on the same Superchip.

5.2 Main Result

We evaluate SuperInfer across all three models and all two datasets, comparing it against several baseline systems:

vLLM (Kwon et al., 2023) (v0.6.6.post1), with V1 engine (vLLM Community, 2025) enabled. It is a widely-used

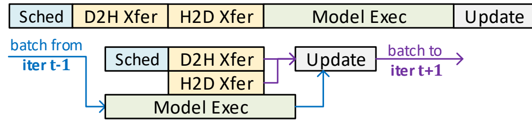

Figure 15. Comparing execution flow of vLLM (up) and SuperInfer (down). Schedule and KV cache transfers (2 CUDA streams) are overlapped with model execution in SuperInfer.

production-level LLM inference system.

TensorRT-LLM (NVIDIA, 2026) (v1.1.0). A highly optimized LLM inference framework by NVIDIA, incorporating a broad set of low-level and kernel optimizations.

LightLLM (Gong et al., 2025), **LTR** (Fu et al., 2024): Two representative SLO-aware scheduler. LightLLM balances trade-off between request queuing and harmful evictions by estimating future memory occupancy. LTR approximates SJF using learning-based request length ranking.

NEO (Jiang et al., 2024c): A representative work to offload partial KV cache and Attention computation to the CPU. Notably it does not support MoE models like Mixtral-8x7B.

For all baselines, PagedAttention (Kwon et al., 2023) and chunked prefill (Agrawal et al., 2024) are enabled. We exclude Pie (Xu et al., 2024), HeteGen (Zhao et al., 2024), and Select-N (Ma et al., 2025) due to the lack of publicly available code. FlexGen (Sheng et al., 2023) is excluded for lacking essential features (PagedAttention, chunked prefill), which prevents fair comparison. Approximate methods like InfiniGen (Lee et al., 2024) are also excluded.

For our evaluation, we configure the RotaSched's parameters as: $\alpha = 3$, $\beta_B = 0$, $\beta_F = 0.5$, $B_{\text{xfer}} = 2400$. TTFT and TBT SLOs are set to 5s and 100ms, respectively. Out of the 480 GB of Grace DRAM, we allocate 400 GB for KV cache offloading and conservatively reserve the remaining 80 GB for the OS and the SuperInfer runtime. This 80 GB serves as a conservative safety margin, as SuperInfer introduces no extra memory consumption beyond the offloading space comparing to vLLM in our setup ($\sim$12 GB), leaving sufficient headroom for stable execution. We report the TTFT and TBT SLO attainment rate under various request arrival rates.

Fig. 16 presents our results, revealing two key findings. First, SuperInfer significantly surpasses baselines in TTFT SLO attainment, especially at high request rates. This demonstrates its effectiveness of mitigating waiting queue HOL blocking via highly responsive active rotation, which proactively preempts running requests to prioritize waiting ones based on VLT (§ 4.2). Second, SuperInfer maintains TBT SLO attainment superior or comparable to baselines, indicating it leverages swap space in an SLO-aware manner for balanced TTFT and TBT improvement without introducing new rotary (swapped) queue HOL blocking (§ 3.1). Among baselines, LTR achieves the best TTFT SLO attainment but significantly sacrifices TBT, as its static deadline-based priority mechanism fails to achieve the balanced improvements. Notably, LightLLM shows unusual trends where TBT SLO attainment improves or stabilizes as the request rate increases, as it is designed to avoid harmful preemption caused by KV cache overflow and maintain stable TBT under high load (check Appendix C for further analysis). Furthermore, while TensorRT-LLM incorporates a broad set of advanced optimizations beyond KV-cache management and scheduling to achieve competitive TBT SLO attainment, its TTFT SLO attainment still degrades significantly at high request rates, due to its reliance on lazy preemption in request scheduling and its offloading strategy under memory pressure, which can delay prompt processing for newly arriving requests.

These results demonstrate SuperInfer's high performance and effectiveness in mitigating SLO violations. This is achieved through the co-design of the SLO-aware, preemptive RotaSched and bandwidth-efficient DuplexKV. Overall, SuperInfer exploits the full potential of the GH200 memory hierarchy and NVLink-C2C for SLO-aware offloading.

5.3 Analysis Results

5.3.1 How does each module of SuperInfer bring benefits?

To isolate the benefit of each module proposed in § 4, we follow settings in § 5.2 and evaluate the following configurations using the Qwen2.5-32B and ShareGPT dataset: **(1) vLLM**: with default FCFS scheduler; **(2) SuperInfer w/o DuplexKV**: SuperInfer with default KV cache offloading engine in vLLM, with two sub-variants for its transfer budget: **L** ($B_{\text{xfer}} = 300$), which reflects the severe bandwidth under-utilization, and **H** ($B_{\text{xfer}} = 2400$), which matches DuplexKV's budget; **(3) SuperInfer**: The full version (RotaSched +DuplexKV). Fig. 17 shows the results. Comparing vLLM and SuperInfer w/o DuplexKV (L) confirms that RotaSched effectively mitigates TTFT SLO violations while maintaining comparable TBT SLO attainment. However, forcing a large B_{xfer} with inefficient offloading engine as SuperInfer w/o DuplexKV (H) degrades performance, yielding significantly lower TBT SLO attainment. This occurs because slow transfers become the critical bottleneck, preventing full overlap with model execution (Fig. 15). Finally, the full SuperInfer successfully manages the large B_{xfer} using DuplexKV, achieving even greater TTFT improvements compared to SuperInfer w/o DuplexKV (L).

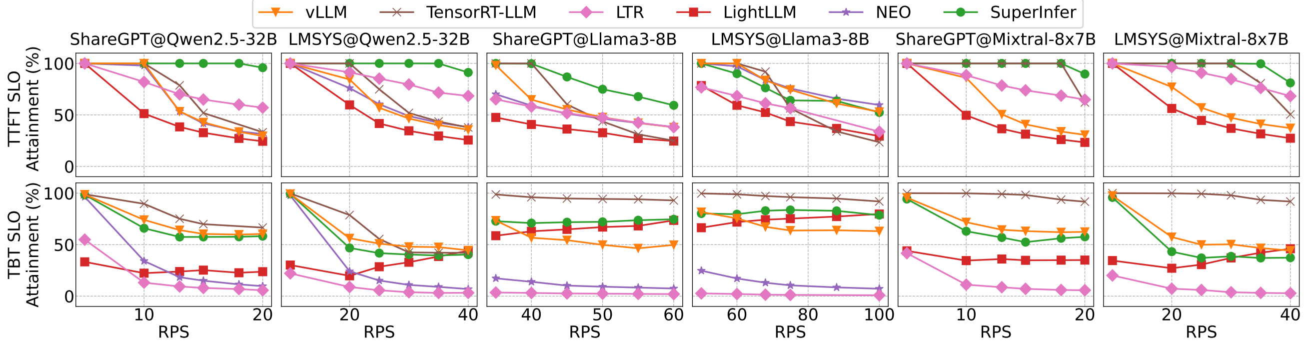

Figure 16. Comparison of SuperInfer against baselines across various models, datasets, and request rates (RPS). SuperInfer achieves significant improvements in TTFT SLO attainment over baselines, while preserving TBT SLO attainment comparable to others.

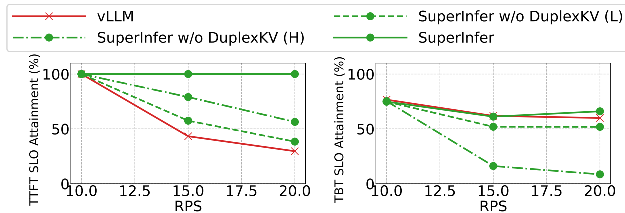

Figure 17. Comparing TTFT and TBT SLO attainment rates for: vLLM, SuperInfer w/o DuplexKV (L/H), SuperInfer.

5.3.2 *How does RotaSched tame tradeoff between TTFT and TBT through VLT?*

As defined in § 4.2.2, VLT includes sensitivity parameter α for TTFT/TBT trade-off, and tolerance parameters β_B, β_F to reflect SLOs and their tolerances. We follow settings in § 5.2 to evaluate their impacts using using Qwen2.5-32B and ShareGPT. Fig. 18 shows results with varying $\alpha \geq 1$ and fixed $\beta_B = \beta_F = 0$. It shows that larger α yields better TBT SLO attainment, as rotary requests get larger VLTs for prioritization. However, this comes at the cost of lower TTFT attainment, as waiting requests are relatively delayed. Thus, for TTFT-sensitive tasks (like summarization), we recommend setting $\alpha \leq 1$ to get the best TTFT improvement. $\alpha = 3$ offers a balanced sweet spot; further increases bring diminishing TBT benefits while significantly harming TTFT. Fig. 19 shows fixing $\alpha = 1$, $\beta_B = 0$ and increasing β_F leads to worse P99 TTFT, as new arrived requests have smaller VLTs and thus lower priority. For $\beta_F \geq 2$, P99 TTFT sharply increases with minimal TBT improvement, as new requests are significantly delayed. Conversely, as Fig. 20 shows, when fixing $\alpha = 1$, $\beta_F = 0$ and varying β_B in a large range, larger β_B worsen P99 TBT, as rotary requests get smaller VLTs and lower priority for resumption. Due to TBT's higher sensitivity to short delays, $\beta_B < 0$ is suggested for best TBT tails latency, albeit at the cost of relatively higher TTFT tail latencies. Overall, α, β_B and β_F provide flexibility to trade off between TBT and

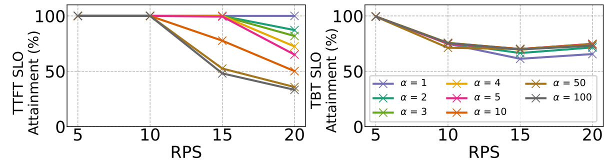

Figure 18. TTFT and TBT SLO attainment rate of SuperInfer under various α. Larger α leads to better TBT but worse TTFT.

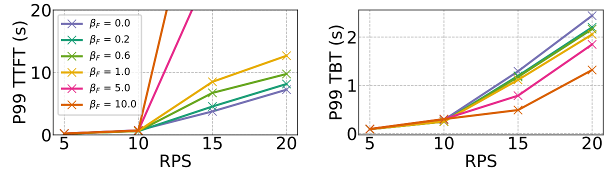

Figure 19. P99 TTFT and TBT of SuperInfer (various β_F values).

TTFT improvements, enabling scenario-specific deployment tuning.

It is important to note that while we provide a qualitative characterization to guide the understanding of parameter behavior under different settings, parameter selection can be tailored to specific applications. SuperInfer does not attempt to correlate or predict query distributions when setting these parameters. Such correlations are difficult to define in online LLM serving due to the dynamics and unpredictability of the workloads. Instead, SuperInfer is designed to proactively react to KV cache pressure and SLO violation risks. The exposed parameters (α, β_B and β_F) simply control how aggressively the system responds once such pressure is observed.

5.3.3 *How much bandwidth can DuplexKV achieve?*

To evaluate DuplexKV's optimizations for bandwidth utilization in § 4.3, we measure the bandwidth and end-to-end (E2E) transfer time of bidirectionally transferring 16GB (8GB or 32768 tokens per direction) of Qwen2.5-32B KV

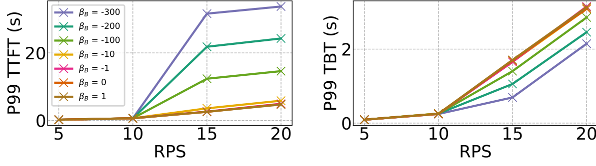

Figure 20. P99 TTFT and TBT of SuperInfer (various β_B values).

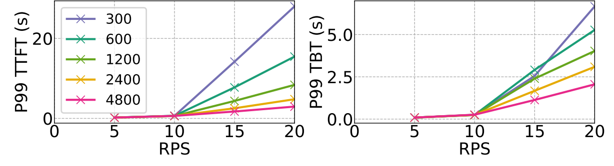

Figure 21. Comparing P99 TTFT and TBT of SuperInfer with various B_{xfer}. Higher B_{xfer} significantly reduce tail latencies.

Table 1. Measured bandwidth and E2E time of different transfer engines. U denotes uni-directional, B denotes bidirectional.

METHOD	D2H (GB/S)	H2D (GB/S)	E2E TIME (MS)
NAIVE	10.75(U)	9.86(U)	1556.15
MS	80.05(U)	133.51(U)	159.87
MS+MK	238.95(U)	269.69(U)	63.14
DUPLEXKV	**180.99(B)**	**179.37(B)**	**46.80**
IDEAL (DUPLEX)	192.00(B)	192.00(B)	41.66

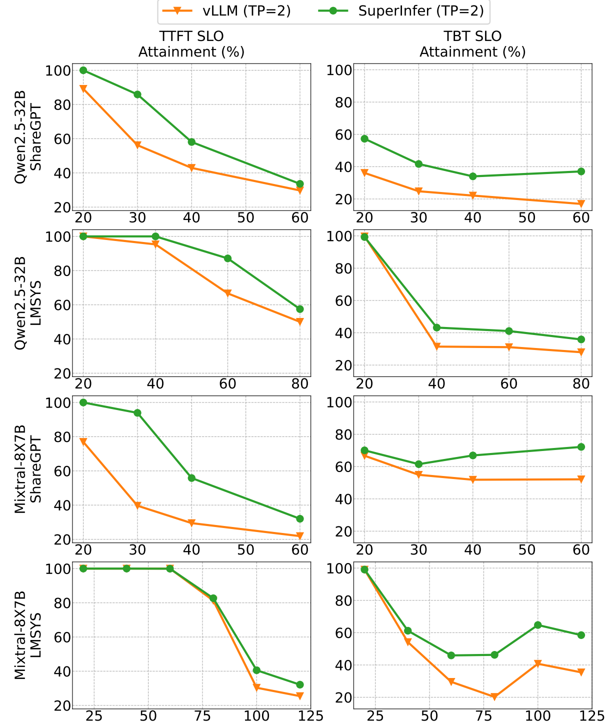

Figure 22. Comparison of SuperInfer and vLLM under TP=2 settings. The performance benefits of SuperInfer successfully extend to distributed configurations.

cache. We compare the following configurations: (1) **Naive**: uses per-segment (64KB) copy as vLLM; (2) **MS** (merged segments): uses our block-first layout to merge segments to 4MB, but still uses per-segment transfer; (3) **MS+MK** (merged kernels): builds on MS, with merged transfer kernel per direction. Crucially, due to data races, transfers are still performed serially (one direction at a time); (4) **DuplexKV**: the full version in § 4.3.2, MS+MK+eager block rotation, enabling concurrent full-duplex transfers; (5) **Ideal**: the theoretical limit of DRAM bandwidth (half-duplex 384GB/s, 192GB/s per direction concurrently). Table 1 shows the results. The naive method achieves only 5.6% of ideal bandwidth and results in 37.4× the ideal E2E time, due to its high inefficiency with small-segment transfers (§ 4.3.1). Instead, merging segments (MS) and kernels (MS+MK) incrementally improves realized uni-directional bandwidth within the limitation of DRAM (384GB/s), showing the importance of large transfer size and batch kernels. However, due to data races, they can only perform direction-serialized transfers, thus achieving only 3.6× and 1.4× the ideal time, respectively. In contrast, our proposed DuplexKV achieves near-ideal bidirectional bandwidth and E2E time consumption (1.1× the ideal), due to its eager block rotation to eliminate data race and enable concurrent bidirectional transfers.

5.3.4 *How important is the high swap bandwidth?*

In LVF scheduling (§ 4.2.3), the transfer budget (B_{xfer}) determines block number rotated per iteration, reflecting the swap bandwidth. As discussed in § 3.2, a high swap bandwidth is crucial to clear accumulated waiting requests timely and prevent new HOL blocking for the rotary (swapped) requests. We validate this by testing SuperInfer with Qwen2.5-32B and ShareGPT, varying B_{xfer} while keeping other settings same as § 5.2. The results in Fig. 21 show that a larger B_{xfer} significantly reduces the P99 TTFT and TBT, which mainly comes from the time requests spend in waiting or rotary states. Also, larger budgets yield greater improvements, confirming the necessity of high swap bandwidth for effective SLO-aware LLM serving with offloading.

A potential concern with SuperInfer is whether KV cache transfers might introduce pipeline stalls if they do not complete before model inference finishes. Upon reviewing the data collected across all experiments in § 5.2, we find that incomplete overlaps which lead to execution stalls occurred in only 0.021% of all engine iterations. A detailed breakdown reveals average latencies of 7.63 ms for scheduling, 15.8 ms for KV transfers, and 69.82 ms for model execution. Because the scheduling and transfer latencies are substantially shorter than the model execution time, their overheads are effectively hidden.

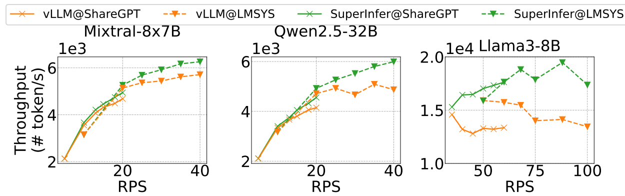

Figure 23. Throughput of vLLM and SuperInfer on three models.

5.3.5 *Does SuperInfer extend to multi-GPU settings?*

To confirm that SuperInfer's benefits persist in distributed environments, we evaluated it using Tensor Parallelism (TP=2) with Qwen2.5-32B and Mixtral-8x7B. GPUs are connected with NVLink (900GB/s). As Fig. 22 shows, across both models and datasets, SuperInfer consistently achieves higher TTFT and TBT SLO attainment rates compared to vLLM. This is because our RotaSched and DuplexKV are fundamentally orthogonal to standard parallelism strategies; they introduce no extra inter-GPU traffic, allowing SuperInfer to scale effectively in multi-GPU and distributed setups.

5.3.6 *How does SuperInfer affect the throughput?*

Fig. 23 shows measured throughput of vLLM and SuperInfer from § 5.2. SuperInfer achieves comparable or slightly better throughput than vLLM, with up to 29.2% improvement at high request rates. This is because in conventional offloading, memory-heavy long requests can monopolize GPU memory for a long time, preventing new requests from leveraging the concurrency benefit of chunked prefill (Agrawal et al., 2024), as prefill requests have few batching opportunities. In contrast, SuperInfer's fast rotation offers prefill requests more batching opportunities.

6 Conclusion

This paper presents SuperInfer, a high-performance, SLO-aware LLM serving system optimized for Superchips. SuperInfer transforms passive preemption into proactive, fine-grained request scheduling, via an OS-inspired active rotary scheduler and a co-designed DuplexKV rotation engine, achieving high NVLink-C2C utilization. Evaluations across diverse models and workloads show that SuperInfer substantially improves SLO attainment under high request rates while maintaining throughput. Our study demonstrates that Superchips unblock new opportunities for LLM serving, but realizing their full potential requires careful co-design of scheduling and memory movement in the software system.

Acknowledgments

We sincerely appreciate the anonymous reviewers and our shepherd. Their insightful feedback helps significantly improve the quality of the paper. This research was supported by the National Science Foundation (NSF) under Grant No. 2441601. The work utilized the Delta and DeltaAI system at the National Center for Supercomputing Applications (NCSA) and Jetstream2 at Indiana University through allocation CIS240055 from the Advanced Cyberinfrastructure Coordination Ecosystem: Services & Support (ACCESS) program, which is supported by National Science Foundation grants #2138259, #2138286, #2138307, #2137603, and #2138296. The Delta advanced computing resource is a collaborative effort between the University of Illinois Urbana-Champaign and NCSA, supported by the NSF (award OAC 2005572) and the State of Illinois. UIUC SSAIL Lab is supported by research funding and gift from Google, IBM, Amazon, and AMD, including the Google ML and Systems Junior Faculty Award.

REFERENCES

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

Agrawal, A., Kedia, N., Panwar, A., Mohan, J., Kwatra, N., Gulavani, B., Tumanov, A., and Ramjee, R. Taming {Throughput-Latency} tradeoff in {LLM} inference with {Sarathi-Serve}. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pp. 117–134, 2024.

Aminabadi, R. Y., Rajbhandari, S., Awan, A. A., Li, C., Li, D., Zheng, E., Ruwase, O., Smith, S., Zhang, M., Rasley, J., et al. Deepspeed-inference: enabling efficient inference of transformer models at unprecedented scale. In *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–15. IEEE, 2022.

Ao, R., Luo, G., Simchi-Levi, D., and Wang, X. Optimizing llm inference: Fluid-guided online scheduling with memory constraints. *arXiv preprint arXiv:2504.11320*, 2025.

Bai, J., Bai, S., Chu, Y., Cui, Z., Dang, K., Deng, X., Fan, Y., Ge, W., Han, Y., Huang, F., et al. Qwen technical report. *arXiv preprint arXiv:2309.16609*, 2023.

Borui, W., Juntao, Z., Chenyu, J., Chuanxiong, G., and Chuan, W. Efficient llm serving on hybrid real-time and best-effort requests. *arXiv preprint arXiv:2504.09590*, 2025.

Cao, S., Liu, S., Griggs, T., Schafhalter, P., Liu, X., Sheng, Y., Gonzalez, J. E., Zaharia, M., and Stoica, I. Moe-lightning: High-throughput moe inference on memory-constrained gpus. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*, pp. 715–730, 2025.

Chen, J., Du, C., Liu, R., Yao, S., Yan, D., Liao, J., Liu, S., Wu, F., and Chen, G. Tokenflow: Responsive llm text streaming serving under request burst via preemptive scheduling. *arXiv preprint arXiv:2510.02758*, 2025a.

Chen, W., He, S., Qu, H., Zhang, R., Yang, S., Chen, P., Zheng, Y., Huai, B., and Chen, G. {IMPRESS}: An {Importance-Informed}{Multi-Tier} prefix {KV} storage system for large language model inference. In *23rd USENIX Conference on File and Storage Technologies (FAST 25)*, pp. 187–201, 2025b.

Chen, W., Lu, C., Xu, H., Ye, K., and Xu, C. Multiplexing dynamic deep learning workloads with slo-awareness in gpu clusters. In *Proceedings of the Twentieth European Conference on Computer Systems*, pp. 589–604, 2025c.

Dubey, A., Jauhri, A., Pandey, A., Kadian, A., Al-Dahle, A., Letman, A., Mathur, A., Schelten, A., Yang, A., Fan, A., et al. The llama 3 herd of models. *arXiv e-prints*, pp. arXiv–2407, 2024.

Fu, Y., Zhu, S., Su, R., Qiao, A., Stoica, I., and Zhang, H. Efficient llm scheduling by learning to rank. *Advances in Neural Information Processing Systems*, 37:59006–59029, 2024.

Fusco, L., Khalilov, M., Chrapek, M., Chukkapalli, G., Schulthess, T., and Hoefler, T. Understanding data movement in tightly coupled heterogeneous systems: A case study with the grace hopper superchip. *arXiv preprint arXiv:2408.11556*, 2024.

Gao, B., He, Z., Sharma, P., Kang, Q., Jevdjic, D., Deng, J., Yang, X., Yu, Z., and Zuo, P. {Cost-Efficient} large language model serving for multi-turn conversations with {CachedAttention}. In *2024 USENIX Annual Technical Conference (USENIX ATC 24)*, pp. 111–126, 2024.

Gong, R., Bai, S., Wu, S., Fan, Y., Wang, Z., Li, X., Yang, H., and Liu, X. 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*, pp. 798–813, 2025.

He, J. and Zhai, J. Fastdecode: High-throughput gpu-efficient llm serving using heterogeneous pipelines. *arXiv preprint arXiv:2403.11421*, 2024.

Hong, K., Li, X., Chen, L., Mao, Q., Dai, G., Ning, X., Yan, S., Liang, Y., and Wang, Y. Sola: Optimizing slo attainment for large language model serving with state-aware scheduling. In *Eighth Conference on Machine Learning and Systems*, 2025.

Hu, C., Huang, H., Hu, J., Xu, J., Chen, X., Xie, T., Wang, C., Wang, S., Bao, Y., Sun, N., et al. Memserve: Context caching for disaggregated llm serving with elastic memory pool. *arXiv preprint arXiv:2406.17565*, 2024.

Huang, J., Xiong, Y., Yu, X., Huang, W., Li, E., Zeng, L., and Chen, X. Slo-aware scheduling for large language model inferences. *arXiv preprint arXiv:2504.14966*, 2025.

Jeong, J. and Ahn, J. Accelerating llm serving for multi-turn dialogues with efficient resource management. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, pp. 1–15, 2025.

Jiang, A. Q., Sablayrolles, A., Roux, A., Mensch, A., Savary, B., Bamford, C., Chaplot, D. S., Casas, D. d. l., Hanna, E. B., Bressand, F., et al. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024a.

Jiang, C., Gao, L., Zarch, H. E., and Annavaram, M. Kvpr: Efficient llm inference with i/o-aware kv cache partial recomputation. *arXiv preprint arXiv:2411.17089*, 2024b.

Jiang, X., Zhou, Y., Cao, S., Stoica, I., and Yu, M. Neo: Saving gpu memory crisis with cpu offloading for online llm inference. *arXiv preprint arXiv:2411.01142*, 2024c.

Kim, H., Wang, N., Xia, Q., Huang, J., Yazdanbakhsh, A., and Kim, N. S. Lia: A single-gpu llm inference acceleration with cooperative amx-enabled cpu-gpu computation and cxl offloading. In *Proceedings of the 52nd Annual International Symposium on Computer Architecture*, pp. 544–558, 2025.

Kwon, W., Li, Z., Zhuang, S., Sheng, Y., Zheng, L., Yu, C. H., Gonzalez, J., Zhang, H., and Stoica, I. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pp. 611–626, 2023.

Lee, W., Lee, J., Seo, J., and Sim, J. {InfiniGen}: Efficient generative inference of large language models with dynamic {KV} cache management. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pp. 155–172, 2024.

Li, Z., Chen, Z., Delacourt, R., Oliaro, G., Wang, Z., Chen, Q., Lin, S., Yang, A., Zhang, Z., Chen, Z., et al. Adaserve: Accelerating multi-slo llm serving with slo-customized speculative decoding. *arXiv preprint arXiv:2501.12162*, 2025.

Lian, X., Tanaka, M., Ruwase, O., and Zhang, M. Superoffload: Unleashing the power of large-scale llm training on superchips. *arXiv preprint arXiv:2509.21271*, 2025.

Liu, A., Feng, B., Xue, B., Wang, B., Wu, B., Lu, C., Zhao, C., Deng, C., Zhang, C., Ruan, C., et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024a.

Liu, Y., Li, H., Cheng, Y., Ray, S., Huang, Y., Zhang, Q., Du, K., Yao, J., Lu, S., Ananthanarayanan, G., et al. Cachegen: Kv cache compression and streaming for fast large language model serving. In *Proceedings of the ACM SIGCOMM 2024 Conference*, pp. 38–56, 2024b.

Luo, C., Cai, Z., Sun, H., Xiao, J., Yuan, B., Xiao, W., Hu, J., Zhao, J., Chen, B., and Anandkumar, A. Headinfer: Memory-efficient llm inference by head-wise offloading. *arXiv preprint arXiv:2502.12574*, 2025.

Ma, C., Ye, Z., Zhao, H., Yang, Z., Fu, T., Han, J., Zhang, J., Luo, Y., Wang, X., Wang, Z., et al. Memory offloading for large language model inference with latency slo guarantees. *arXiv preprint arXiv:2502.08182*, 2025.

NVIDIA. Nvidia grace hopper superchip architecture whitepaper, 2024. URL https://resources.nvidia.com/en-us-grace-cpu/nvidia-grace-hopper?ncid=no-ncid. [Online; accessed 2025-10-26].

NVIDIA. Nvidia/tensorrt-llm: Tensorrt llm provides users with an easy-to-use python api to define large language models (llms) and supports state-of-the-art optimizations to perform inference efficiently on nvidia gpus. tensorrt llm also contains components to create python and c++ runtimes that orchestrate the inference execution in a performant way., 2026. URL https://github.com/NVIDIA/TensorRT-LLM. [Online; accessed 2026-03-27].

Patke, A., Reddy, D., Jha, S., Qiu, H., Pinto, C., Narayanaswami, C., Kalbarczyk, Z., and Iyer, R. Queue management for slo-oriented large language model serving. In *Proceedings of the 2024 ACM Symposium on Cloud Computing*, pp. 18–35, 2024.

Qin, R., Li, Z., He, W., Cui, J., Ren, F., Zhang, M., Wu, Y., Zheng, W., and Xu, X. 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)*, pp. 155–170, 2025.

Qiu, H., Mao, W., Patke, A., Cui, S., Jha, S., Wang, C., Franke, H., Kalbarczyk, Z. T., Başar, T., and Iyer, R. K. Efficient interactive llm serving with proxy model-based sequence length prediction. *arXiv preprint arXiv:2404.08509*, 2024.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Schieffer, G., Wahlgren, J., Ren, J., Faj, J., and Peng, I. Harnessing integrated cpu-gpu system memory for hpc: a first look into grace hopper. In *Proceedings of the 53rd International Conference on Parallel Processing*, pp. 199–209, 2024.

ShareGPT Team. Sharegpt, 2023. URL https://sharegpt.com/.

Sheng, Y., Zheng, L., Yuan, B., Li, Z., Ryabinin, M., Chen, B., Liang, P., Ré, C., Stoica, I., and Zhang, C. Flexgen: High-throughput generative inference of large language models with a single gpu. In *International Conference on Machine Learning*, pp. 31094–31116. PMLR, 2023.

Sheng, Y., Cao, S., Li, D., Zhu, B., Li, Z., Zhuo, D., Gonzalez, J. E., and Stoica, I. Fairness in serving large language models. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pp. 965–988, 2024.

Song, Y., Mi, Z., Xie, H., and Chen, H. Powerinfer: Fast large language model serving with a consumer-grade gpu. In *Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles*, pp. 590–606, 2024.

Stoica, I. and Abdel-Wahab, H. Earliest eligible virtual deadline first: A flexible and accurate mechanism for proportional share resource allocation. *Old Dominion Univ., Norfolk, VA, Tech. Rep. TR-95-22*, 1995.

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

Vellaisamy, P., Labonte, T., Chakraborty, S., Turner, M., Sury, S., and Shen, J. P. Characterizing and optimizing llm inference workloads on cpu-gpu coupled architectures. *arXiv preprint arXiv:2504.11750*, 2025.

Vijaya Kumar, A., Antichi, G., and Singh, R. Aqua: Network-accelerated memory offloading for llms in scale-up gpu domains. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, pp. 48–62, 2025.

vLLM Community. vllm v1 - vllm, 2025. URL https://docs.vllm.ai/en/latest/usage/v1_guide.html. [Online; accessed 2025-10-27].

Wei, Z., Yen, J., Chen, J., Zhang, Z., Huang, Z., Chen, C., Yu, X., Gu, Y., Wu, C., Wang, Y., et al. Equinox: Holistic fair scheduling in serving large language models. *arXiv preprint arXiv:2508.16646*, 2025.

Wong, C. S., Tan, I., Kumari, R. D., and Wey, F. Towards achieving fairness in the linux scheduler. *ACM SIGOPS Operating Systems Review*, 42(5):34–43, 2008.

Wu, B., Zhong, Y., Zhang, Z., Liu, S., Liu, F., Sun, Y., Huang, G., Liu, X., and Jin, X. Fast distributed inference serving for large language models. *arXiv preprint arXiv:2305.05920*, 2023.

Xu, Y., Mao, Z., Mo, X., Liu, S., and Stoica, I. Pie: Pooling cpu memory for llm inference. *arXiv preprint arXiv:2411.09317*, 2024.

Yang, A., Yu, B., Li, C., Liu, D., Huang, F., Huang, H., Jiang, J., Tu, J., Zhang, J., Zhou, J., et al. Qwen2. 5-1m technical report. *arXiv preprint arXiv:2501.15383*, 2025.

Yu, C., Wang, T., Shao, Z., Zhu, L., Zhou, X., and Jiang, S. Twinpilots: A new computing paradigm for gpu-cpu parallel llm inference. In *Proceedings of the 17th ACM International Systems and Storage Conference*, pp. 91–103, 2024.

Yu, G.-I., Jeong, J. S., Kim, G.-W., Kim, S., and Chun, B.-G. Orca: A distributed serving system for {Transformer-Based} generative models. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pp. 521–538, 2022.

Zhang, H., Tang, Y., Khandelwal, A., and Stoica, I. {SHEPHERD}: Serving {DNNs} in the wild. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pp. 787–808, 2023.

Zhang, W., Wu, Z., Mu, Y., Liu, B., Lee, M., and Lai, F. Tempo: Application-aware llm serving with mixed slo requirements. *arXiv preprint arXiv:2504.20068*, 2025.

Zhao, X., Jia, B., Zhou, H., Liu, Z., Cheng, S., and You, Y. Hetegen: Heterogeneous parallel inference for large language models on resource-constrained devices. *arXiv preprint arXiv:2403.01164*, 2024.

Zheng, L., Chiang, W.-L., Sheng, Y., Li, T., Zhuang, S., Wu, Z., Zhuang, Y., Li, Z., Lin, Z., Xing, E. P., et al. Lmsys-chat-1m: A large-scale real-world llm conversation dataset. *arXiv preprint arXiv:2309.11998*, 2023.

Zheng, L., Yin, L., Xie, Z., Sun, C. L., Huang, J., Yu, C. H., Cao, S., Kozyrakis, C., Stoica, I., Gonzalez, J. E., et al. Sglang: Efficient execution of structured language model programs. *Advances in Neural Information Processing Systems*, 37:62557–62583, 2024.

Zhu, K., Gao, Y., Zhao, Y., Zhao, L., Zuo, G., Gu, Y., Xie, D., Ye, Z., Kamahori, K., Lin, C.-Y., et al. {NanoFlow}: Towards optimal large language model serving throughput. In *19th USENIX Symposium on Operating Systems Design and Implementation (OSDI 25)*, pp. 749–765, 2025.

A Comparing FCFS and SJF-Oracle

As discussed in § 3.1, recent LLM serving systems increasingly employ sophisticated scheduling techniques to manage latency. However, they cannot overcome the memory constraints posed by hardware. Once KV cache storage is exhausted, newly arrived requests have to backlog in the waiting queue, waiting for running ones to finish.

To illustrate, we measure vLLM on GH200 using Qwen2.5-32B (Yang et al., 2025) model and ShareGPT dataset (ShareGPT Team, 2023). We compare the *First-Come-First-Serve* (FCFS) and *Shortest-Job-First* with oracle generation length information (SJF-Oracle) policy. As shown in Fig. 24, both FCFS and SJF-Oracle fail to prevent TTFT SLO violations under memory pressure. Once KV cache storage is exhausted, the length of waiting queue spikes dramatically, which causes severe TTFT SLO violations.

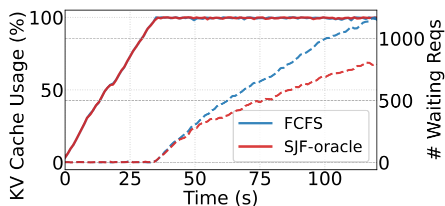

Figure 24. KV cache usage and waiting request number for vLLM with FCFS and SJF-oracle scheduler. Model: Qwen2.5-32B, dataset: ShareGPT, RPS=20.

B Bandwidth Measurement

We measure the CPU-GPU bandwidth for GH200 and H200 using NVIDIA's open-source `nvbandwidth` tool (v0.8). We focus on two specific bidirectional copy engine (CE) test cases:

- `host_to_device_bidirectional_memcpy_ce` (Test ID 2)
- `device_to_host_bidirectional_memcpy_ce` (Test ID 3)

The following command is used to run all tests:

```
CUDA_VISIBLE_DEVICES=0 numactl
    --cpunodebind=0 --membind=0
    ./nvbandwidth -t 2 3 -b <size_in_MB>
    -i 3
```

C Further Analysis of LightLLM

As discussed in § 5.2, LightLLM (Gong et al., 2025) shows an unusual TBT SLO attainment rate trend: as request rates increase, the TBT SLO attainment rate improves or stabilizes. To further explore the underlying root cause, we analyze the cumulative distribution function (CDF) of TBT for LightLLM.

As shown in Fig. 25, when request rate increase, the CDF of TBT of LightLLM shows almost no changes. Therefore, the TBT SLO attainment rate also stabilizes. That is because LightLLM's "Past-Future" scheduler is designed to avoid harmful request evictions by precisely estimating the peak future KV cache required by the running batch. Unlike aggressive schedulers that underestimate memory consumption and suffer from frequent, harmful evictions under high load, LightLLM's approach ensures that scheduled requests can complete successfully without the response interruptions that typically cause TBT SLO violations. This stable TBT performance, free from eviction-induced stalls, directly translates to a stabilized SLO attainment rate even as request rates increase.

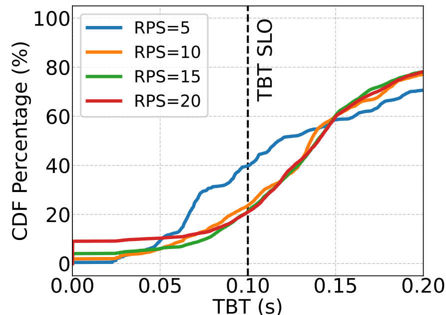

Figure 25. Cumulative distribution function (CDF) of TBT for LightLLM in § 5.2. When request rate increase, LightLLM

D vLLM on Unified Memory of GH200

Some readers may consider the hardware-managed Unified Memory (UM) of GH200 (NVIDIA, 2024) as a potential good solution for offloading. UM merges HBM and DRAM into a unified address space and automatically migrates blocks based on access, allowing for a unified large KV cache storage without explicit offloading management. However, our test result in Fig. 26 show that enabling UM for vLLM severely degrades performance, leading to significantly higher average TBT.

This performance collapse stems not from traditional UM overheads caused by page-fault-driven migration, but from a fundamental architecture and workload mismatch.

On conventional PCIe-based GPUs, UM is page-fault driven. An access to a non-resident page (e.g., in CPU memory) triggers a costly page fault and GPU stall, after which the driver migrates the page over PCIe. In contrast, the GH200's architecture features a cache-coherent NVLink-C2C interconnect and Address Translation Services (ATS) (NVIDIA,

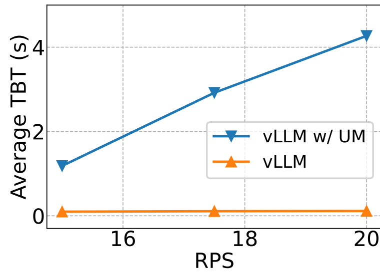

Figure 26. Comparing the vLLM and that with KV cache storage in GH200's Unified Memory (UM). vLLM on UM shows significant TBT degradation.

2024). This allows the Hopper GPU to directly access the Grace CPU's DRAM without incurring any page faults.

GH200 does support page migration, but instead of being page-fault driven, it uses *hardware access counters* to track the access frequency of pages from both the CPU and GPU. When a CPU-resident page is accessed from the GPU side, it initially keeps its residency in CPU memory. Only after enough frequent accesses will the GH200 trigger a background migration from CPU to GPU.

Although this migration is transparent to applications like vLLM and involves no page faults, it exposes a new bottleneck: the bandwidth cliff. The GPU's local HBM memory provides bandwidth up to 4 TB/s, whereas the CPU's DRAM subsystem, even when accessed directly over the 900 GB/s C2C link, is limited to its own lower bandwidth of 384 GB/s. This makes GPU access to CPU memory nearly an order of magnitude slower than access to HBM.

This bandwidth cliff is the root cause of vLLM's poor performance on UM. When a new request's KV cache blocks are resident in DRAM, the GPU's initial accesses are all serviced over the low-bandwidth C2C link through ATS, leading to poor performance in the Attention kernel. The request may finish processing long before the hardware counters accumulate enough frequency to ever trigger a migration. The system is thus perpetually "warming up" data that is already "cold" or no longer in active use.

Therefore, this hardware-managed UM is not suitable for LLM serving with offloading.

E ARTIFACT APPENDIX

E.1 Artifact check-list (meta-information)

- **Algorithm:** RotaSched: Largest-VLT-First (LVF) Scheduling; DuplexKV: Eager Block Rotation.
- **Program:** SuperInfer (implemented in Python and C++, atop vLLM v0.6.6.post1[1]).
- **Compilation:** CUDA 12.8 and GCC 13.3.0 for DuplexKV and CUDA kernels of SuperInfer.
- **Dataset:** ShareGPT[2]; LMSYS-Chat-1M[3].
- **Run-time environment:** Ubuntu 24.04.3 LTS with kernel 6.8.0-100-generic-64k, CUDA 12.8.
- **Hardware:** NVIDIA GH200 NVL2[4], with 2 Grace-Hopper pairs, each including 144GB HBM and 480GB DRAM connected via NVLink-C2C.
- **Execution:** A pre-configured Docker environment with automated bash scripts for orchestrating server and client processes, benchmarking baselines, parsing execution logs, and generating performance plots..
- **Metrics:** Time-To-First-Token (TTFT) SLO attainment rate; Time-Between-Tokens (TBT) SLO attainment rate; Throughput (tokens/s); Achieved NVLink-C2C transfer bandwidth (GB/s) and transfer time (ms).
- **Output:** Parsed logs and Python-generated plots.
- **Experiments:** End-to-end SLO attainment comparison against baselines (vLLM, LightLLM, LTR, NEO).
- **Disk Requirement:** ∼500 GB (to accommodate model weights for Qwen2.5-32B, Mixtral-8x7B, LLaMA-3-8B, dataset caches, and the Docker image).
- **Time Requirement:** Time for pulling docker image varies based on the network connection speed. The whole experiments takes about 30 hours. Because the full evaluation across all models and datasets is time-consuming, we also provide a "lite" experiment for convenience. This lite version evaluates a single model on a single dataset to quickly reproduce the core main results, which can be completed in approximately 5 hour.
- **Code licenses:** Apache 2.0[5].
- **Data licenses:** LMSYS-Chat-1M Dataset License Agreement[6] (LMSYS-Chat-1M) and Apache 2.0[5] (ShareGPT).
- **Archived DOI:** Artifact: `https://doi.org/10.5281/zenodo.18971768`. Code: `https://doi.org/10.5281/zenodo.19394229`.

E.2 Description

E.2.1 Hardware dependencies

- **Platform:** NVIDIA GH200 NVL2. This system features two Grace CPU and Hopper GPU pairs. Each Hopper GPU is equipped with 144GB HBM, and each Grace CPU contains 480GB DRAM (with a maximum bandwidth of 384GB/s). Within each pair, the Grace CPU and Hopper GPU are tightly coupled via an NVLink-C2C interconnect, providing 900GB/s of bidirectional bandwidth. Additionally, the two Hopper GPUs are interconnected via a 900GB/s NVLink.
- **Storage:** About 500GB of free disk space is required to accommodate the Docker image, model weights, and dataset cache.

E.2.2 Software dependencies

- **OS & Environment:** Ubuntu 24.04.3 LTS with kernel 6.8.0-100-generic-64k, CUDA 12.8 and GCC 13.3.0. Our provided Docker image is pre-packaged with this environment. The virtual environments for SuperInfer and all evaluated baselines are managed and isolated using Anaconda.
- **SuperInfer:** Implemented atop vLLM v0.6.6.post1[7], with PyTorch 2.5.1.
- **Baselines:** The evaluated baselines have the following specific software dependencies:
  - **vLLM:** Version v0.6.6.post1[7] with PyTorch 2.5.1.
  - **LTR[8]:** Git commit `13bbf6ff3dab661791d41362551b089e5f77c91c` with PyTorch v2.4.1.
  - **LightLLM[9]:** Version v1.1.0 with PyTorch 2.9.0.
  - **NEO[10]:** Git commit `33e4a0f7632688e4122de4c3c140196cebea6a5a` with PyTorch 2.7.0 and Triton 3.3.0[11].

E.2.3 Datasets

We use two open-source datasets to evaluate SuperInfer and the baselines:

[1] https://github.com/vllm-project/vllm/releases/tag/v0.6.6.post1

[2] https://huggingface.co/datasets/anon8231489123/ShareGPT_Vicuna_unfiltered/blob/main/ShareGPT_V3_unfiltered_cleaned_split.json

[3] https://huggingface.co/datasets/lmsys/lmsys-chat-1m

[4] https://www.nvidia.com/en-us/data-center/grace-hopper-superchip/

[5] https://choosealicense.com/licenses/apache-2.0/

[6] https://huggingface.co/datasets/lmsys/lmsys-chat-1m#lmsys-chat-1m-dataset-license-agreement

[7] https://github.com/vllm-project/vllm/releases/tag/v0.6.6.post1

[8] https://github.com/hao-ai-lab/vllm-ltr

[9] https://github.com/ModelTC/LightLLM

[10] https://github.com/NEO-MLSys25/NEO

[11] https://github.com/triton-lang/triton

1. **ShareGPT[12]:** This dataset contains highly diverse, real-world conversational turns between users and LLMs. In our evaluation, following prior works such as LTR, we extract only the first conversational turn (the initial human prompt and the LLM response) to serve as a single request.

2. **LMSYS-Chat-1M[13]:** A large-scale, real-world dataset comprising one million conversations collected in the wild from the LMSYS Chatbot Arena.

During testing, for each dataset, we randomly sample a total of $120 \times \text{RPS}$ requests (representing 120 seconds of traffic at a target request rate). These requests are then dispatched to the server with inter-request delays sampled from a Poisson distribution to simulate realistic arrival patterns.

E.2.4 Models

We use three open-source LLMs for our evaluations to cover diverse model scales and architectural designs:

- **LLaMA-3-8B[14]:** Represents widely adopted, small-scale dense models.
- **Mixtral-8x7B[15]:** Represents Mixture-of-Experts (MoE) architectures.
- **Qwen2.5-32B[16]:** Represents larger-scale dense models.

E.3 Installation

To simplify the environment setup and installation process, we provide a self-contained Docker image. This image comes prepackaged with all necessary runtime dependencies, the SuperInfer itself, the evaluated baselines. Model weights will be download automatically when running the scripts (Hugging Face token required).

```
docker pull monsoon235/
    superinfer_ae_public
docker run --gpus all -it -u 1003 -w /
    home/mingtao4 monsoon235/
    superinfer_ae_public zsh  # enter
    the environment
```

E.4 Experiment workflow and expected result

A full evaluation of the main results takes roughly 30 hours due to the extensive combination of 3 models, 2 datasets, 5 methods, and various request rates. For a quicker reproduction, you can run the provided "lite" experiment (∼5 hour), which evaluate all methods on just 1 model (Qwen2.5-32B) and 1 dataset (ShareGPT).

E.4.1 Lite Version

To execute the "lite" version of the experiment, please run the following commands:

```
ulimit -n 100000
conda activate vllm-v1
python /home/mingtao4/AE_scripts/
    AELite_AIO_0309_v1.py
```

This automated script evaluates SuperInfer alongside the baselines (vLLM, LTR, LightLLM, and NEO) using the Qwen2.5-32B model and the ShareGPT dataset. It sequentially launches the LLM server and the benchmark client for all data points. The generated logs are saved in `AE_results`.

To reproduce the figure in the paper, run Jupyter Notebook `/home/mingtao4/AE_scripts/AELite_mainresult.ipynb` to automatically parse these logs and reproduce the figure.

Expected Results: It should reproduce the first column (ShareGPT @ Qwen2.5-32B) of Fig. 16 in the paper.

E.4.2 Full Version

To execute the full version of the experiment, please run the following commands:

```
ulimit -n 100000
conda activate vllm-v1
python /home/mingtao4/AE_scripts/
    AE_AIO_0309_v1.py
```

This automated script evaluates SuperInfer alongside the baselines (vLLM, LTR, LightLLM, and NEO) using all models and datasets. It sequentially launches the LLM server and the benchmark client for all data points. The generated logs are saved in `AE_results`.

To reproduce the figure in the paper, run Jupyter Notebook `/home/mingtao4/AE_scripts/AE_mainresult.ipynb` to automatically parse these logs and reproduce the figure.

Expected Results: It should reproduce the full Fig. 16 in the paper.

[12] https://huggingface.co/datasets/anon8231489123/ShareGPT_Vicuna_unfiltered/blob/main/ShareGPT_V3_unfiltered_cleaned_split.json

[13] https://huggingface.co/datasets/lmsys/lmsys-chat-1m

[14] https://huggingface.co/meta-llama/Meta-Llama-3-8B

[15] https://huggingface.co/mistralai/Mixtral-8x7B-v0.1

[16] https://huggingface.co/Qwen/Qwen2.5-32B