Chorus: GPU–CPU Cooperative LLM Serving for Length-Skewed Decode

Dong Chen*
Hong Kong University of Science and Technology
Hong Kong, China
dchenbl@cse.ust.hk

Yukun Zhou*
Hong Kong University of Science and Technology
Hong Kong, China
yzhoufw@connect.ust.hk

Sheng Yao*
Hong Kong University of Science and Technology
Hong Kong, China
syaoao@connect.ust.hk

Ruichuan Chen
Nokia Bell Labs
Stuttgart, Germany
ruichuan.chen@nokia-bell-labs.com

Binhang Yuan
Hong Kong University of Science and Technology
Hong Kong, China
biyuan@ust.hk

Shuai Wang
Hong Kong University of Science and Technology
Hong Kong, China
shuaiw@cse.ust.hk

Luping Wang
Alibaba Group
Hangzhou, China
chamu.wlp@alibaba-inc.com

Cheng Huang
Alibaba Group
Hangzhou, China
xiaoluo.hc@alibaba-inc.com

Guodong Yang
Alibaba Group
Hangzhou, China
luren.ygd@alibaba-inc.com

Liping Zhang
Alibaba Group
California, USA
liping.z@alibaba-inc.com

Wei Wang†
Hong Kong University of Science and Technology
Hong Kong, China
weiwa@cse.ust.hk

Abstract

Online LLM serving is increasingly constrained by GPU memory capacity during decode. In length-skewed workloads, a small fraction of long requests keeps large key-value (KV) caches resident across many decode steps, exhausting GPU high-bandwidth memory (HBM), throttling admission, and reducing goodput for the short-request majority. At the same time, GPU servers provision substantial CPU and DRAM resources, and recent Intel CPUs add Advanced Matrix Extensions (AMX) for matrix-heavy kernels. The challenge is to use these host resources without placing CPU work on the per-token GPU critical path.

We present Chorus, a GPU–CPU cooperative serving system that uses AMX-equipped CPUs as a second decode lane for long requests with persistent KV state. Chorus keeps prefill and short-request decode on GPUs, offloads selected long-request decode once to distributed AMX CPU groups, and releases the corresponding GPU KV pages so the GPU lane can admit more short requests. To satisfy online service level objectives (SLOs), Chorus combines pressure-aware offload admission, tensor-parallel AMX decode that aggregates CPU compute and memory bandwidth, and AMX kernels specialized for small, mixed-length decode batches. Across popular LLMs and production-derived workloads, Chorus achieves the highest SLO-attaining goodput among baselines, improving peak goodput over SGLang by up to 1.26×. At the same offered load, Chorus delivers 1.3–11.4× SGLang's goodput when GPU-only serving enters time-to-first-token (TTFT) collapse, and improves over CPU-offload baselines by up to 4.3×.

CCS Concepts: • **Software and its engineering → Cloud computing**; • **Computer systems organization → Heterogeneous (hybrid) systems**.

Keywords: large language model serving, heterogeneous CPU–GPU systems, long-context inference, Intel AMX

ACM Reference Format:
Dong Chen, Yukun Zhou, Sheng Yao, Ruichuan Chen, Binhang Yuan, Shuai Wang, Luping Wang, Cheng Huang, Guodong Yang, Liping Zhang, and Wei Wang. 2026. Chorus: GPU–CPU Cooperative LLM Serving for Length-Skewed Decode. In *The 2026 ACM SIGOPS Annual Technical Conference (SIGOPS ATC 2026), November 15–18, 2026, Hong Kong, China.* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3828523.3857065

*Equal contribution to this work.
†Corresponding author: Wei Wang (weiwa@cse.ust.hk).



This work is licensed under a Creative Commons Attribution 4.0 International License.
SIGOPS ATC 2026, Hong Kong, China
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2818-1/2026/11
https://doi.org/10.1145/3828523.3857065

1 Introduction

LLM inference consists of two phases. *Prefill* processes the input prompt in parallel: it is dominated by large general matrix multiplications (GEMMs) and largely determines time-to-first-token (TTFT) [4, 29, 60]. *Decode* then generates output one token at a time: each step streams the model weights and rereads the accumulated key-value (KV) cache, so time-per-output-token (TPOT) is bound by memory bandwidth and by how long KV state stays resident in GPU memory [3, 4, 32, 56]. In this regime, GPU High-Bandwidth Memory (HBM) *capacity*, not compute or bandwidth, becomes the binding constraint. Model weights already consume much of HBM, and live KV caches grow with both context length and batch size (e.g., a 16K context for Qwen3-32B in BF16 occupies 4 GB). When HBM fills before streaming multiprocessors (SMs) or HBM bandwidth saturate, the serving system must admit fewer requests and shrink decode batches, losing goodput under interactive TTFT and TPOT SLOs.

How severely a deployment hits this capacity wall depends on its workload, and production workloads are heavily *length-skewed*: their traces consistently contain many short requests alongside a small tail of long requests with lengthy inputs or long generations [29, 38, 44, 47]. In our ServeGen [47] replay, the 95th/99th percentiles of total length reach roughly 5K/10K tokens for its Model-Large and 14K/29K tokens for DeepSeek-R1; Azure conversation and code traces show similar tails (Figure 1).

This skew translates directly into HBM pressure because long and short requests occupy memory for very different durations. A long request keeps a large KV cache resident across many decode steps, so a few long requests can pin HBM far longer than many short ones, throttling admission and batching for the short-request majority. Replaying a mixed ServeGen [47] workload on a decode GPU, we find that long-request windows drive KV-cache usage near the deployment's configured 90% high watermark while GPU HBM bandwidth stays at 38%—a clear indication that the GPU is capacity-starved, not bandwidth-saturated. In comparison, short-request windows, whose per-request residency is 4× smaller, sustain a 3.25× larger decode batch and 3.12× higher throughput on the same hardware (Figure 2). These results motivate isolating long and short requests in *separate resource pools*. Prior work already applies coarse-grained placement within GPU clusters: length-aware systems maintain separate GPU pools for long and short requests [25, 55]. PPD [23] provides another example: it routes append-prefill in later conversation turns between GPU prefill and decode workers based on KV locality, resource conditions, and SLOs. However, keeping long-request decode on GPUs remains costly: its long-lived KV state consumes scarce HBM capacity, constraining throughput and utilization.

In this paper, we explore offloading long-request decode to the CPUs that already sit idle in production LLM serving clusters. Such host resources are abundant yet largely unused: in our fleet snapshot, online 8-GPU servers keep CPU utilization in the 6–10% range, while a single server provides 8× the memory capacity of one H20 GPU (Figure 3a). Beyond being mostly idle, recent Intel Xeon CPUs add a new hardware feature called Advanced Matrix Extensions (AMX), on-core matrix engines that accelerate the GEMM-heavy kernels of decode [19, 20]. Together, the observed CPU headroom and AMX capability make these hosts a potential decode resource rather than a passive memory tier.

Putting these CPUs on the decode path, however, is harder than it appears, and two natural designs both fall short. The first keeps each request on the GPU and offloads selected operators or layers, such as attention or experts, to the CPU [7, 11, 17, 18, 37, 52]. For example, LIA partitions decoder sublayers between a single GPU and its host AMX CPU. This mixed placement exploits AMX compute but introduces CPU–GPU dependencies into every decode step. Our characterization shows why this per-step coupling is fragile online: on a single AMX node, attention is 11.0–12.8× slower than on an H20 and a B200, and non-kernel runtime overhead is 12–21× slower (§3.2). Thus, per-token CPU work and CPU–GPU transfers repeatedly stall the GPU loop. In our expert-offload measurement, the p_{99} GPU wait-for-CPU time reaches 196.3–215.9% of GPU expert execution time (Figure 3b), matching the SLO collapse we measure for an operator-offload baseline (§6.2). The second design avoids per-step coupling by moving whole requests to a single CPU, but then misses TPOT: extrapolating our measured DeepSeek-R1 layer breakdown gives ~132 ms/token on one AMX node, outside the acceptable range for interactive online serving [35, 42, 60].

We resolve both shortcomings in Chorus, a GPU–CPU cooperative serving system for length-skewed workloads. Its focus is distributed AMX offload to relieve GPU memory pressure from long-request decode within online SLOs. Chorus builds on two ideas that make CPU decode useful without putting CPUs on the GPU critical path. First, Chorus uses a *cooperative division of labor*: GPUs keep all prefill and short-request decode, where TTFT and short-request TPOT are most sensitive; AMX CPUs take only selected long-request decode; and each offload releases the request's GPU KV pages so the GPU lane can admit more short requests. Because Chorus offloads a long request *once* and lets the CPU group decode it independently, it avoids per-token CPU–GPU synchronization. Second, the CPU lane must be scaled to meet TPOT. Chorus uses *distributed AMX decode*, sharding each layer's attention heads, projections, and experts across AMX-equipped CPU nodes with tensor parallelism (TP). Wider TP aggregates CPU memory bandwidth and compute but adds communication and replicated work, while data parallelism (DP) replicates TP groups for throughput. Chorus profiles TP/DP layouts and selects the highest-throughput

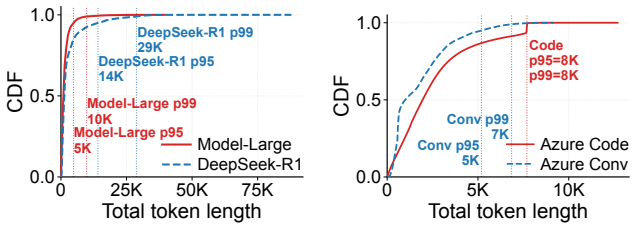
TPOT-feasible one (§6.5); this distributed layout brings long-request TPOT of the CPU lane into the interactive TPOT range used by online serving systems [35, 42, 60].

Turning these two ideas into a running system raises three challenges, each solved by one mechanism (§4). **(1) When to offload requests to AMX is not a fixed decode length cutoff.** A static threshold may over-offload and violate AMX TPOT, or under-offload and waste CPU capacity; Chorus instead schedules offloads under pressure, moving a long request only when GPU HBM is contended and some AMX group has TPOT-safe headroom. **(2) A single CPU node cannot meet TPOT.** Distributed AMX decode must aggregate CPU memory bandwidth and compute for each offloaded request, but wider groups add communication and reduce the number of parallel decode groups. Chorus therefore avoids over-wide groups where communication dominates. **(3) Offloaded decode requires a specialized AMX execution path.** The offloaded workload creates three small-batch bottlenecks: long-prefix attention is imbalanced across AMX cores, MoE routing scatters few tokens across experts, creating underfilled GEMMs, and TP places hidden-state AllReduce on every layer's critical path. Chorus addresses them with three AMX-path optimizations: *adaptive space-division multiplexing* partitions attention by cached-prefix blocks and merges partial results with online softmax; *guarded routed/shared expert fusion* remaps experts into a compact local space and fuses small-batch GEMMs; and *AVX-accelerated Chunk-pipelined AllReduce* overlaps chunk transfer with vectorized reduction for TP communication.

We evaluate Chorus on Qwen3-32B, DeepSeek-R1, and GLM-4.5 using production-derived ServeGen and AzureLLM traces [29, 38, 47], comparing with SGLang GPU-only serving [59], NEO [17], and KTransformers [7]. Across all six model/workload pairs, Chorus achieves the highest SLO-attaining goodput among baselines: up to 1.26× over SGLang at peak goodput, 1.3–11.4× SGLang's goodput when GPU-only serving enters TTFT collapse at the same offered load, and up to 4.3× over CPU-offload baselines. These results show that AMX-equipped hosts provide a practical second decode lane for length-skewed workloads without moving latency-sensitive work off GPUs.

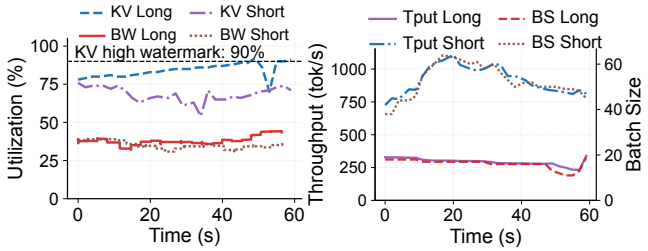
2 Background and Motivation

Length-skewed LLM serving. Production LLM serving workloads are *length-skewed*: most requests are short, while a small tail carries long contexts or long generations. Notably, in ServeGen [47], total-length p95/p99 are roughly 5K/10K tokens for Model-Large and 14K/29K tokens for DeepSeek-R1 (Figure 1a); Azure LLM conversation and code traces [29, 38] show a similar pattern, with total-length p95/p99 values of roughly 5K/7K and 8K/8K tokens, respectively (Figure 1b).



(a) ServeGen M-Large & DS-R1 **(b)** Azure Code & Conversation

Figure 1. Request-length distributions in real-world LLM production traces. Both ServeGen and Azure LLM contain a short-request majority and a long tail, motivating long-request decode as a distinct serving class.



(a) Resource utilization. **(b)** Throughput and batch size.

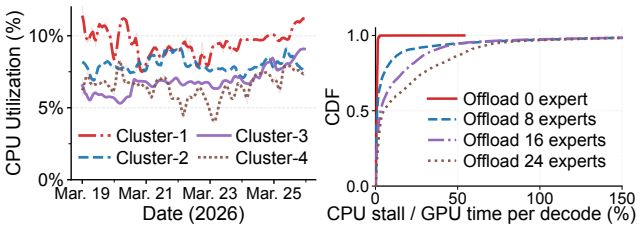
Figure 2. GPU decode in long- and short-request-dominant windows from the same ServeGen run. Long-request KV residency limits admission and throughput before GPU HBM bandwidth saturates.

Long-request decode is a residency problem. This skew stresses GPU memory through *residency time*, not just request count. For a decode batch, the KV footprint is:

$$M_{\mathrm{KV}} = 2 \times L \times n_{\mathrm{kv}} \times d_h \times S_{avg} \times B \times b, \tag{1}$$

where L is the number of layers, n_{kv} the number of KV heads, d_h the head dimension, S_{avg} the average sequence length, B the batch size, and b bytes per element. Because the footprint grows with sequence length, long requests consume more HBM while also keeping their KV state live across many decode steps. A few such requests can occupy HBM longer than many short ones and exhaust the batching headroom left for the short-request majority, limiting throughput to a low level. Therefore, the dominant cost of long-request decode is *persistent KV-cache residency*, not per-step compute.

To quantify the impact of this residency cost, we replay a mixed ServeGen workload on the decode GPU using the Qwen3-32B setup in §6. Figure 2 compares long- and short-request-dominant windows from the same run; all values are window averages. In the long-request-dominant window, KV-cache usage approaches the deployment's configured 90% high watermark, leaving little headroom for admitting new decode requests, while HBM bandwidth remains at 38%. The GPU is hence *capacity-starved*, not bandwidth-bound: long requests pin KV state for tens of seconds, hold the decode batch at 16, and limit throughput to 293 tokens/s. When the workload transitions to the short-request-dominant window,



(a) CPU utilization **(b)** CPU-induced GPU stall

Figure 3. Host-side opportunity and coupling cost. (a) CPU utilization on online 8-GPU AI servers remains mostly in the 6%–10% range across one week. (b) GPU wait-for-CPU time normalized by GPU expert execution time during decode. Expert offload creates frequent and long-tail GPU stalls.

per-request residency is 4× smaller, the batch size grows to 52, and throughput reaches 917 tokens/s under comparable SM and bandwidth utilization.

To prevent long-request decode from stalling the short-request majority, prior systems [25, 55] propose to serve long and short requests in separate GPU pools. While this design provides desirable isolation between the two request classes, it requires additional GPU provisioning, and those GPUs assigned to long-request serving still carry the HBM capacity pressure, leading to limited throughput and utilization. Relieving this pressure requires moving long-request decode—and thus its persistent KV state—off the GPU without slowing prefill or short-request decode. The key question is what resource should it be offloaded to.

CPUs as a second decode lane. We explore using the CPUs that already sit beside the GPU serving path as a *second decode lane* for long requests, based on two observations. First, the measured GPU inference clusters provide substantial, already-provisioned CPU and DRAM headroom. Figure 3a shows that on standard 8-GPU servers, CPU utilization stays mostly in the 6–10% range in a one-week trace collected from four LLM serving clusters of a real-world hyperscaler. Across the measured deployments, 5,800 GPU servers leave roughly 4.75M logical cores and a correspondingly large DRAM pool underutilized. Similar CPU idleness is also reported in other production serving clusters [43, 50]. Second, CPU hosts are no longer compute-poor for LLM decode: modern Intel server CPUs add Advanced Matrix Extensions (AMX), on-core matrix engines, which accelerate the GEMM-heavy kernels of decode, and prior work shows AMX can drive CPU-side token generation [1, 19]. Abundant idle capacity plus AMX capability make these hosts a potential active decode lane rather than a passive memory tier.

Prior work falls short. Existing CPU-assisted LLM serving designs, however, do not fit this role. Table 1 categorizes these studies into two design approaches and compares them on whether they meet the key requirements of efficient long-request decode. The first design, *memory-tier offload*, places

Table 1. CPU-assisted serving designs for online LLM decode. A design "passes" a column when it avoids steady-state CPU–GPU synchronization, scales CPU-side compute and memory bandwidth, or preserves online SLOs for the short-request majority.

Design	Avoids Step Sync	Scale Comp/BW	Online SLO
Memory-tier offload [37, 52]	Partial	No	Partial
Operator offload [6, 7, 13, 17, 18]	No	Yes	No
Chorus (Ours)	Yes	Yes	Yes

KV cache or weights in host DRAM to expand effective capacity [37, 52]; it relieves HBM but adds no decode compute path, and repeated movement of large states can instead make PCIe the bottleneck [49]. The second design, *operator offload*, selectively offloads attention, KV-heavy steps, or MoE experts to CPUs while keeping the remaining heavy-lift request compute on the GPU decode loop [6, 7, 13, 17]. LIA [18] is the closest example: for each input length and batch size, it partitions decoder sublayers between a single GPU and its host AMX CPU. This mixed placement exploits AMX compute but requires CPU–GPU synchronization at every decode step, leading to frequent GPU stalls. Figure 3b quantifies this synchronization overhead by measuring the time GPU spends on waiting for CPU over the time it spends on expert execution for each decode step; a value of 100% means the GPU waits for CPU-side work for as long as it spends executing the corresponding expert work. The no-offload baseline has only 2.4% p_{99} stall, but with 8, 16, and 24 CPU-offloaded experts, p_{99} stall reaches 215.9%, 197.8%, and 196.3%. In addition to these problems, both design approaches leave the same gap: neither isolates long-request tails from the short-request majority nor preserves online SLOs for short requests.

The discussions above narrow down the design space to *request-level, decode-only separation*: offload a long request decode to the CPU side once, release its GPU KV pages, and let it finish generating there without per-token synchronization. Whether this is achievable hinges on one open question: can AMX-equipped CPUs meet a long-request TPOT target at request granularity? We answer this question in §3 and derive the challenges and requirements that shape the design of Chorus's in §4.

3 CPU Characterization and Design Requirements

We next characterize whether AMX-equipped CPUs can support the request-level, decode-only separation motivated in §2. We profile DeepSeek-R1 decode on a modern Intel Xeon Granite Rapids (GNR) CPU using the SGLang-based AMX backend in Chorus (§3.1). By comparing operator-level latency across the CPU and NVIDIA H20 and B200 GPUs

Table 2. Hardware specifications: NVIDIA H20 and B200 GPUs vs. an Intel Xeon GNR (Granite Rapids) server with 96 inference cores. CPU compute is theoretical AMX peak†; sustained throughput is substantially lower (§3.2).

Type	Metric	H20 GPU	B200 GPU	Xeon GNR
Compute	BF16	148 TFLOPS	2,250 TFLOPS	314.6 TFLOPS†
	INT8	296 TOPS	4,500 TOPS	629.1 TOPS†
Memory	Type	HBM3	HBM3e	DDR5
	Capacity	96 GB	180 GB	768 GB
	Peak bandwidth	4.0 TB/s	8.0 TB/s	825 GB/s

†AMX peak = cores × ops/cycle × freq: 96×1024×3.2 GHz (BF16), 96×2048×3.2 GHz (INT8).

(Table 3), we attribute the performance gaps to hardware and runtime constraints (§3.2). These constraints expose three challenges for a practical AMX decode lane, which we translate into design requirements for Chorus (§3.3).

3.1 Platform and Experimental Setup

Table 2 summarizes the hardware specs of our platform. The CPU testbed is a single-socket Intel Xeon GNR server with 120 physical cores, 96 of which are dedicated to inference. Each core has an AMX tile unit for low-precision matrix multiplication and AVX-512 vector pipelines for softmax, normalization, activation, and gating. The 768 GB DDR5 capacity can hold the weights and KV state of the models we study, but peak DDR5 bandwidth is only about 20% of H20 HBM3 and 10% of B200 HBM3e.

We profile a 6-layer DeepSeek-R1 proxy: three dense, non-mixture-of-experts (Non-MoE) layers and three mixture-of-experts (MoE) layers. The proxy preserves the two layer types of the full 61-layer model, which has 3 Non-MoE layers and 58 MoE layers. All traces use sequence length $L = 16{,}384$ and batch size 1. CPU measurements use our SGLang-based AMX path with optimized long-prefix attention and fused decoder-layer kernels; GPU measurements use CUDA Graph replay to reduce framework dispatch overhead in the measured GPU path. The results below are single-node measurements; multi-node aggregation is considered only in §3.3.

3.2 Performance Gaps and Root Causes

Table 3 shows two patterns. First, the CPU–GPU gap varies by components on two GPU baselines. For weight-streaming **linear GEMMs**, the CPU is 2.6–2.7× slower than H20 and 4.1–5.0× slower than B200, consistent with the DDR5–HBM bandwidth gap. **MoE experts** and **LayerNorm** show similar CPU–GPU gaps of 3.3–3.9×. In contrast, **attention** is ∼11× slower than H20 and 12.8× slower than B200, while accounting for roughly one-third of CPU kernel time. **MoE routing** is the only component on par with both GPUs (17 vs. 19 µs), because its serial top-k logic benefits little from GPU-wide parallelism. Second, non-kernel **overhead** adds a

Table 3. Per-layer decode breakdown: Xeon GNR (96 cores) with our optimized AMX kernels vs. H20 and B200 GPUs for DeepSeek-R1 ($L = 16{,}384$, batch size BS = 1). Kernel rows report compute-kernel time only. The Overhead row aggregates all non-kernel costs including Python wrappers, dispatch, activation, and framework plumbing.

Component	MoE layer (µs)			Non-MoE layer (µs)		
	CPU	H20	B200	CPU	H20	B200
Attention	637	58	50	642	58	50
Linear (GEMM)	443	167	109	1,223	470	247
MoE Experts	720	219	186	–	–	–
MoE Routing	17	19	19	–	–	–
LayerNorm	37	10	11	38	10	11
Kernel	**1,854**	**474**	**374**	**1,903**	**538**	**307**
Overhead	316	16	15	194	16	16
Total	**2,170**	**490**	**390**	**2,097**	**554**	**324**

12–21× gap. CUDA Graph replay captures a fixed GPU execution sequence, reducing framework and dispatch overhead to a graph launch.The CPU incurs analogous costs per layer because it lacks a comparable graph-capture path. We next trace these gaps to hardware and runtime constraints.

Memory bandwidth. The first gap comes from weight streaming. At batch size BS = 1, every projection is a matrix–vector product with arithmetic intensity ∼1 OP/byte, so performance is limited primarily by memory bandwidth. DDR5 sustains ∼700–800 GB/s out of an 825 GB/s peak, whereas H20 sustains ∼2 TB/s out of a 4.0 TB/s peak. This 2.5–2.9× realized-bandwidth ratio matches the observed 2.6–2.7× GEMM gap, despite AMX's 2.1× theoretical peak-BF16 advantage over H20. B200 provides up to twice H20's peak memory bandwidth and reduces the same GEMMs further, yielding a 4.1–5.0× gap relative to the CPU. Memory bandwidth remains the primary bottleneck at other small batch sizes as well: prior profiling [33] shows that CPU computation spends substantial time stalled on L1 and L2 cache misses while waiting for data from main memory. Beyond weight projection, the ∼11× and 12.8× attention gaps relative to H20 and B200 exceed the peak-bandwidth ratios of ∼5× and ∼10×, respectively; the remaining gap reflects how the CPU executes attention within and across cores.

Intra-core serialization. On the CPU, each core executes the four attention phases in strict program order (Figure 4(a)). AMX tile instructions and AVX-512 vector instructions issue from the same instruction stream and cannot proceed concurrently, and there is no asynchronous memory access approach to decouple KV loading from compute. Compounding this, AMX's vector neural-network instruction (VNNI) operand format forces a Pack-KV step on AVX-512 before any $QK^\top$ can issue—work that has no GPU counterpart. The two

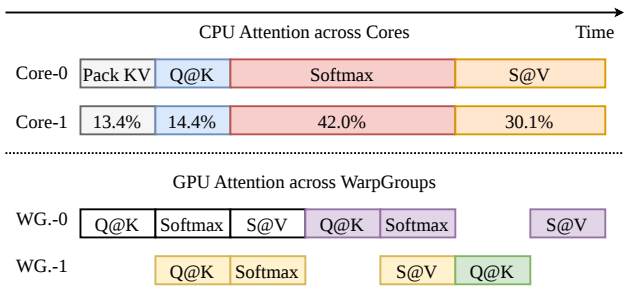


Figure 4. Attention core loop during decode. (a) CPU executes Pack-KV (AVX-512), $QK^\top$ (AMX), Softmax (AVX-512), and score $\times V$ (AMX) in strict serial order; AMX tiles sit idle during the AVX-512 phases. (b) GPU asynchronously pipelines KV streaming with matmul and softmax, keeping Tensor Cores continuously occupied.

AVX-512 phases (Pack-KV and Softmax) therefore become visible serial latency; together, they account for 55.4% of the per-core wall-clock time during which AMX sits idle. A GPU streaming multiprocessor (SM) hides these costs through asynchronous KV streaming and FlashAttention-3's Warp-Group interleaving [9], keeping Tensor Cores occupied.

Cross-core reduction. Saturating DDR5 during decode requires all 96 cores to participate in each attention call. The KV dimension is therefore sharded across cores: each core streams its shard once and emits a partial (max, sum, output) triple. A serial cross-core combine then merges these triples, costing 200 µs, or 29.5% of the end-to-end attention call. This combine is unavoidable on the CPU: replicating the full $L = 16{,}384$ KV stream on each core would inflate DDR5 traffic by orders of magnitude. The GPU avoids this tradeoff because FlashAttention-3 keeps the computation inside one SM's register file, with no cross-SM reduction on the critical path. Together with intra-core serialization, this combine explains why the attention gap exceeds the bandwidth-justified gap for both GPUs: from ~5× to 11× relative to H20 (a further ~2.3×), and from ~10× to 12.7–12.8× relative to B200. These asymmetries come from the CPU memory hierarchy, instruction set architecture (ISA), and core-coordination model, not from a single implementation artifact.

3.3 Challenges and Design Requirements

Our characterization narrows the role of AMX CPUs. They should not replace the GPU fast path: using the per-layer totals in Table 3, a full DeepSeek-R1 decode step with 58 MoE layers and 3 dense layers would take about ~132 ms/token on one AMX node, exceeding the interactive TPOT range [35, 42, 60]. Instead, AMX CPUs should serve as a selective decode lane for long requests whose KV state constrains GPU admission (§2). This role creates three design challenges and corresponding requirements (**R1-R3**).

Challenge-1: Choosing when to use the AMX lane. The decision cannot use a static request length cutoff. Offloading too many requests can overload AMX and violate offloaded-request TPOT, while offloading too few leaves long-lived KV state on the GPU and limits short-request admission. The offload decision must therefore depend on both sides of the tradeoff: GPU HBM pressure, which determines the benefit of releasing KV pages, and AMX headroom, which keeps the offload TPOT-safe. Thus, (**R1**) *Chorus must offload only when GPU memory is contended and an AMX group can accept the request without exceeding its TPOT-safe capacity.*

Challenge-2: Meeting TPOT after offload. Request-level offload removes per-token GPU–CPU synchronization: once a request is offloaded to AMX, slow CPU execution no longer stalls the GPU loop. The risk is instead that the offloaded request violates its TPOT SLO. Because one AMX node has 3.8–4.4× and 5.6–6.5× higher end-to-end per-layer latency than an H20 and a B200, respectively, Chorus must aggregate CPU memory bandwidth and compute across nodes. Tensor parallelism (TP) is the relevant form of resource aggregation because it shards one layer and reduces per-token latency; data parallelism and pipeline parallelism improve throughput rather than single-token latency. ~91% of MoE-layer kernel time can be parallelized via TP across attention heads, projections, MLP dimensions, and experts. However, the remaining serial work—dispatch, replicated kernels, and AllReduce communication—bounds useful TP width (§6.5). Thus, (**R2:**) *Chorus must use distributed TP that is wide enough to meet TPOT, but bounded to avoid communication- and serial-work dominated groups.*

Challenge-3: Specializing offloaded AMX decode. After request offload, an AMX group serves small, mixed-length decode batches over long cached prefixes, where attention work follows prefix length rather than request count. Selective MoE routing scatters few tokens across routed and shared experts, producing small GEMMs that underfill AMX tiles. TP also places hidden-state AllReduce on every layer's critical path. Generic CPU operators and communication are not optimized for these workloads. Thus, (**R3**) *Chorus must specialize the AMX decode path for long-prefix attention, small-batch expert execution, and TP communication.*

4 Chorus System Design

This section presents Chorus's design. Chorus uses AMX as a selective decode lane: prefill and short-request decode stay on GPUs, while long-request decode is offloaded only when doing so frees GPU HBM without violating offloaded-request TPOT. Chorus maps the three requirements from §3.3 to three mechanisms. *Adaptive decode offload* moves long requests only under GPU HBM pressure and TPOT-safe AMX capacity (**R1**; §4.2). *Distributed AMX decoding* aggregates CPU compute and memory bandwidth while bounding TP degree (**R2**; §4.3). *AMX decode-path optimizations* accelerate

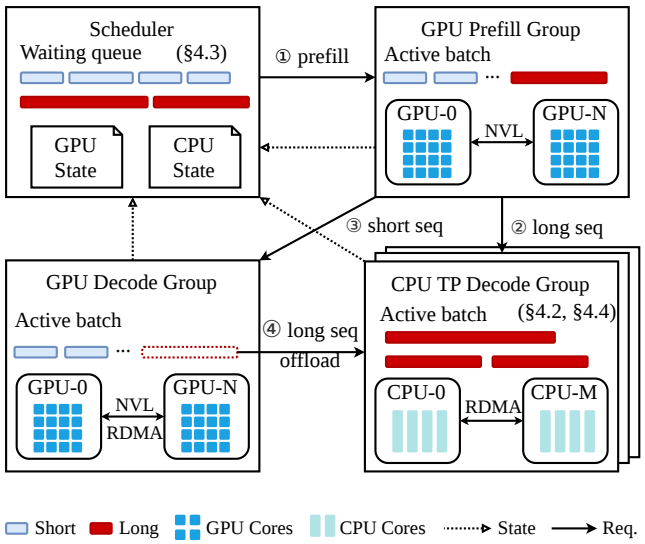


Figure 5. Overview of CHORUS. The scheduler keeps prefill and short-request decode on GPUs, routes eligible long requests to AMX after prefill or offloads active GPU-decode requests, and reclaims the released GPU KV pages.

the small-batch, long-prefix kernels and latency-critical TP communication created by offload (**R3**; §4.4).

4.1 System Overview

Architecture. Figure 5 shows CHORUS's four components. The *Global Scheduler* controls admission, decode placement, and offload. *Prefill GPU Groups* process every request and produce its initial KV cache. *Decode GPU Groups* are the default decode path and serve all short requests. *Decode AMX Groups* are multi-server CPU groups that independently execute selected long-request decode after offload.

The Global Scheduler maintains the waiting queue and tracks GPU KV-cache pressure, GPU queueing, AMX load, and recent TPOT attainment. It admits prefill only when a downstream decode group has capacity, then chooses the decode destination after prefill. Each Decode AMX Group is an independent decode engine with a *TPOT-safe admission quota*: the number of additional requests it can accept without exceeding the offloaded-request TPOT target. The quota is initialized from the group's profiled TP/DP layout (§4.3) and updated from recent tail-TPOT feedback.

End-to-end workflow. When a request arrives, it enters the waiting queue and is scheduled on a Prefill GPU Group (step ①). After prefill, the Global Scheduler chooses between GPU decode and AMX decode. CHORUS routes decode directly to AMX after prefill only when three conditions hold: Decode GPU HBM usage is high, the request currently holds a large KV cache, and a Decode AMX Group has nonzero TPOT-safe admission quota (step ②). Otherwise, the request enters the Decode GPU Group (step ③).

Algorithm 1 Adaptive AMX Decode Offload

Require: Source Decode GPU Group G; selected Decode AMX Group A
Require: Post-prefill candidates Q_{pf} destined for G (step ②); active requests Q_{dec} on G (step ④)
$Q \leftarrow Q_{\text{pf}} \cup Q_{\text{dec}}$
Estimate B_{safe} from A's recent tail TPOT profile
$B_{\text{cur}} \leftarrow$ requests currently decoded by A
$q \leftarrow \max(0, B_{\text{safe}} - B_{\text{cur}})$
if $q = 0$ **or** $|Q| = 0$ **or** G is not HBM-constrained **then**
  return no offload
end if
$k \leftarrow \min(q, |Q|)$
$\tau \leftarrow \max(L_{\min}, \text{KTHLARGESTKVLENGTH}(Q, k))$
for each r among the k largest KV-cache consumers in Q **do**
  if $L_{\text{kv}}(r) \geq \tau$ **and** $q > 0$ **then**
    Complete offload of r to A; release its GPU KV pages
    $q \leftarrow q - 1$
  end if
end for

CHORUS may also offload an active GPU-decode request when its KV cache grows, the Decode GPU becomes HBM-constrained, and an AMX group has nonzero TPOT-safe admission quota (step ④). CHORUS transfers the request's KV cache and sampling state, releases its GPU KV pages, and generates all subsequent tokens on AMX. Each request crosses the GPU–CPU boundary at most once, avoiding recurrent synchronization and offload thrashing. This rule forgoes moving a request back to GPU when pressure later falls; instead, TPOT-aware admission controls AMX load to keep the offloaded request within its TPOT target.

4.2 Adaptive Long-Request Decode Offload

Offload eligibility. CHORUS offloads requests only when doing so relieves an HBM-constrained Decode GPU without overloading the AMX lane. A Decode GPU is HBM-constrained when its KV-cache usage reaches the configured high watermark. A request is eligible when (1) its current KV-cache length exceeds a dynamic threshold τ, (2) the Decode GPU is HBM-constrained, and (3) the target AMX group exposes a nonzero TPOT-safe admission quota. A configurable minimum $L_{\min}$ excludes short requests. If any condition fails, decode remains on the GPU. Algorithm 1 summarizes the scheduler decisions at steps ② and ④.

Pressure-aware adaptation. CHORUS runs the policy for each HBM-constrained Decode GPU Group G after prefill or when GPU-decode state changes. It estimates each AMX group's offload quota and chooses the group with nonzero quota and the lowest recent tail TPOT as target A. The candidate set Q contains post-prefill requests destined for G and active requests already decoded there; B_{cur} is the number of requests currently decoded by A.

Over a sliding time- or request-based window, the scheduler tracks recent tail TPOT by AMX batch size and context-length range. It sets B_{safe} to the largest observed batch size whose tail TPOT remains within the target for the candidate context range, yielding offload quota $q = \max(0, B_{\text{safe}} - B_{\text{cur}})$.

Within this quota, Chorus prioritizes immediate HBM reclamation. Let $L_{\text{kv}}(r)$ denote request r's current KV-token count. For an active-decode candidate, this state resides on G and is released by offload; for a post-prefill candidate, it resides on the Prefill GPU Group and represents the allocation avoided on G. Chorus ranks Q by $L_{\text{kv}}(r)$; for $k = \min(q, |Q|)$, let $L_{(k)}$ be the k-th largest value. The dynamic threshold $\tau = \max(L_{\min}, L_{(k)})$ selects at most the k largest eligible KV-cache consumers. Thus, TPOT feedback sizes the offload quota, while KV-cache ranking spends it on requests that release or avoid the most Decode GPU HBM.

4.3 Distributed AMX Decoding

Distributed execution. Chorus forms Decode AMX Groups that jointly execute each offloaded batch with tensor parallelism (TP), aggregating enough compute and memory bandwidth to meet the TPOT target. Each group shards attention heads, head-coupled projections, dense MLP dimensions, and MoE experts across servers; LayerNorm, routing, and pre-sharding projections remain replicated. Group members execute the same offloaded batch in lockstep, with two AllReduce operations per layer to combine the attention and MLP/expert outputs. Each member receives only its KV heads during offload. This reduces per-server weight and KV traffic. The Global Scheduler treats each group as one decode engine, and each group retains the offloaded request until the auto-regressive decoding completes.

SLO-aware TP/DP layout. For P AMX servers and TP degree N, Chorus forms $\lfloor P/N \rfloor$ data-parallel groups. Wider TP lowers TPOT but reduces concurrency and adds communication overhead; narrower TP creates more groups but may not meet the TPOT target. Chorus therefore uses an offline layout-selection procedure for the target model and AMX pool: profile feasible TP degrees over expected context lengths and batch sizes, discard layouts that exceed TPOT, and select the TP/DP layout with the highest aggregate SLO-feasible throughput. At deployment, Chorus loads the selected model shards and exposes each group as an independent offload destination. These profiled limits provide the initial TPOT-safe admission quota used by §4.2; runtime TPOT feedback refines the quota as workload mix and batch composition change during serving.

4.4 AMX Optimizations for Offloaded Decode

Offload produces small, mixed-length decode batches with long cached prefixes, while distributed execution involves per-layer AllReduce. Generic CPU execution exposes three bottlenecks: equal attention partitioning creates stragglers, separate MoE expert kernels amplify small-batch overhead, and general-purpose collectives add synchronization latency. Chorus addresses them with adaptive space-division multiplexing (SDM), guarded routed/shared expert fusion, and AVX-accelerated chunk-pipelined AllReduce (ACAR).

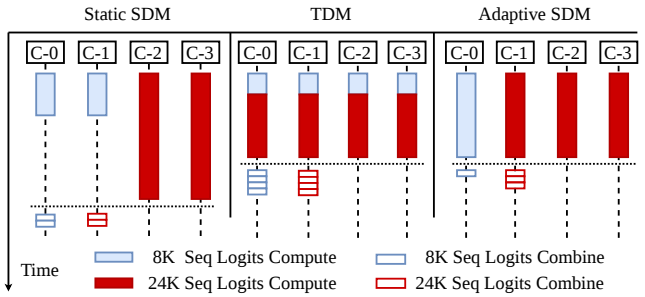


Figure 6. Attention partitioning across four CPU workers for 8K and 24K requests. Static space-division multiplexing (SDM) creates stragglers; time-division multiplexing (TDM) balances each request but serializes the batch; adaptive SDM balances token work across workers.

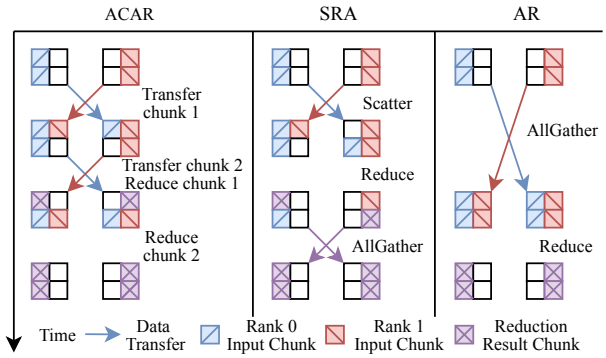


Figure 7. AllReduce designs. AVX-accelerated Chunk-pipelined AllReduce (ACAR) overlaps chunk transfer with AVX reduction; Reduce-Scatter AllGather (SRA) reduces distributed chunks before gathering them; AllGather Reduce (AR) gathers all chunks before local reduction.

Adaptive SDM for mixed-length attention. AMX attention must expose parallel work without duplicating KV-cache traffic. Splitting grouped-query attention (GQA) or Multi-head Latent Attention (MLA) across additional workers can replicate reads of KV state shared by multiple query heads. Chorus therefore parallelizes over cached-prefix blocks and merges partial attention outputs.

Assigning every request the same number of prefix splits balances request count, but not token work. Figure 6 illustrates this problem using four CPU workers and a two-request batch with 8K and 24K prefixes. Static SDM divides each request into two chunks, producing two 4K chunks and two 12K chunks; workers assigned the 12K chunks finish later and become stragglers. TDM avoids this imbalance by assigning all workers to one request at a time, but serializes the requests within the batch.

Adaptive SDM instead allocates a configured global split budget in proportion to each request's prefix cache length. For request i, this block count is $s_i = \lceil L_i/N_b \rceil$, where L_i is its cached-prefix length and N_b is the decode block size. The allocation respects thread and temporary-buffer limits. In the figure, this produces four balanced 8K chunks across the 8K and 24K requests. Each split emits a partial value vector and log-sum-exp statistic, which an online-softmax merge combines without materializing the full attention matrix.

Guarded routed/shared expert fusion. Existing CPU implementations execute routed and shared experts through separate AMX kernels. In the small batches produced by selective offload, each kernel performs little work, making the two launch paths and their intermediate-memory traffic a significant fraction of latency. Chorus instead remaps routed and shared experts into one compact local expert space and pads their token assignments into AMX-friendly `BLOCK_M` groups. A unified path executes the expert projections with batch-reduce GEMM and fuses SiLU gating, routing-weight application, and cross-expert accumulation.

The unified path helps only while batches remain small. At larger batches, separate kernels amortize their overhead through larger expert GEMMs and can expose more parallelism. Chorus therefore profiles the crossover batch size offline and selects the fused or separate path at runtime. We evaluate this crossover in §6.6.

AVX-accelerated Chunk-pipelined AllReduce. AllReduce lies on every TP layer's critical path for hidden-state synchronization. Existing UCC algorithms expose a latency–bandwidth trade-off: AllGather Reduce [40] (AR) uses one communication round but pays high transfer and reduction cost, while Reduce-Scatter AllGather [31] (SRA) lowers those costs at the expense of a second communication phase.

Figure 7 shows Chorus's AVX-accelerated Chunk-pipelined AllReduce (ACAR), tailored to hidden-state synchronization. ACAR places AllGather chunks with one-sided RDMA writes and applies vectorized AVX reduction as each chunk arrives, overlapping communication with reduction while avoiding SRA's extra communication phase.

5 Implementation

Chorus adds approximately 16K lines of code to SGLang [59], Mooncake [30], PyTorch Distributed, UCC [41], and UCX.[1] The SGLang router implements §4.2 using GPU KV-cache pressure, AMX load, and recent TPOT. Mooncake transfers each offloaded request's KV cache and sampling state among prefill GPU, decode GPU, and AMX engines over RDMA; after transfer, the source GPU releases the KV pages and the AMX group continues decode.

Decode AMX Groups run tensor-parallel inference through PyTorch Distributed over UCC/UCX. We add ACAR to UCC and integrate adaptive SDM attention, guarded routed/shared expert fusion, and fused dense/MoE decoder layers into the AMX backend (§4.4). Cgroups reserve CPU and memory for GPU serving and assign only spare capacity to AMX serving.

[1]Unless otherwise stated, we use SGLang commit 57cdcc0210483727bd2, Mooncake v0.3.6, UCC v1.6.0, and UCX v1.20.0-rc1.

6 Evaluation

6.1 Methodology

Our evaluation asks five questions: (Q1) How much does request-level AMX offload improve SLO-attaining goodput over SGLang and applicable operator-offload baselines? (Q2) Can TPOT-aware admission use AMX headroom without violating the TPOT target? (Q3) Can distributed AMX groups meet the long-request TPOT requirement? (Q4) How should a fixed CPU pool balance tensor parallelism (TP) and data parallelism (DP)? (Q5) How much do AMX kernel and communication optimizations reduce decode-path latency?

Testbed. Our testbed has two 8-GPU servers for GPU serving and eight 8-GPU servers used only as AMX hosts. Each GPU-serving server has eight NVIDIA H20 GPUs (96 GB HBM each), two 48-core Intel Xeon 8469 CPUs, and 1 TB DDR4; its GPUs share 900 GB/s bidirectional NVLink. Each AMX server has one 120-core Intel Xeon 6979P and 768 GB of 12-channel DDR5. We do not use the GPUs on these hosts or count them as serving capacity. Considering coexistence with host workloads, we reserve 20% of each AMX host's cores and use the remaining 96 cores for AMX decode. GPU-serving servers and AMX servers have respectively four and one 200 Gbps BlueField-3 DPUs, and communicate over RoCE v2.

Models and deployment. We evaluate one dense **Qwen3-32B** [53] and two MoE models: **GLM-4.5** [39] (355B total, 32B activated parameters) and **DeepSeek-R1** [10, 12] (671B total, 37B activated parameters). Qwen3-32B uses two TP GPUs for prefill and one GPU for decode. Each MoE model uses eight TP GPUs for prefill and eight for decode. All Chorus deployments use the eight AMX servers as one CPU pool. GPUs run BF16; the AMX path uses the fastest SGLang CPU serving format for each model: FP8 for Qwen3-32B and INT8 for GLM-4.5 and DeepSeek-R1 [34]. These precisions remain fixed across experiments and follow common low-precision CPU-serving practice [5, 7, 36, 48].

Workloads. We replay two production-derived traces with bursty arrivals and length skew. **ServeGen** captures time-varying input and output lengths, including long reasoning requests [47]. **AzureLLM** contains invocation times and input/output token counts from Azure inference services [29, 38]. Because AzureLLM predates recent long-generation reasoning workloads, we double its output lengths while preserving input lengths and arrival times [47]. Both traces retain a short-request majority and a long tail (Figure 1). Unless stated otherwise, each run replays 500 sampled requests

for at least 10 minutes; we sweep arrival rates to identify the highest SLO-attaining operating point.

Baselines. We compare against SGLang and two operator-offload systems. **SGLang (GPU-only)** [59] uses the same SGLang-based runtime and GPU prefill/decode deployment as Chorus, but disables AMX offload and keeps decode state and computation on GPUs. **NEO** [17] pipelines attention and KV-cache work on the local host CPU; its available implementation does not support the recent MoE models, so we evaluate it only on Qwen3-32B. **KTransformers** [7] offloads selected MoE experts to AMX CPUs while retaining the remaining model path on GPUs. We use its INT4 hybrid configuration on GLM-4.5 and DeepSeek-R1 and offload at most eight routed experts to avoid overloading its CPU path.

Metrics. Our primary metric is *SLO-attaining goodput*: output-token throughput from requests satisfying both TTFT and TPOT targets. For input length L, the TTFT target is calculated as $\min(\max(0.5, L/512), 10)$ seconds, and the TPOT target is 75 ms. The TPOT target is stricter than those in prior large-model serving studies [2, 50, 60] and close to public interactive-serving guidance [35, 42]. We report maximum SLO-attaining goodput over each arrival-rate sweep, together with mean TTFT and TPOT_{P95}. AMX-path studies report TPOT, per-operator latency, or communication latency, depending on the mechanism being isolated.

6.2 End-to-End SLO-Aware Serving

This experiment answers Q1 by comparing Chorus with SGLang and CPU-offload baselines under the same TTFT and TPOT targets. It tests whether selective long-request offload improves SLO-attaining goodput. Figures 8 and 9 provide complementary views of end-to-end performance: the former illustrates each system's TTFT and TPOT results, while the latter reports the consequent SLO-attaining goodput. Because a request contributes to goodput only when it meets both targets, the objective is not to minimize TTFT or TPOT independently, but to sustain both over the widest range of offered loads.

Chorus achieves the highest SLO-attaining goodput. Across all six model/workload pairs, Chorus reaches the best SLO-attaining goodput among the systems applicable to each model, improving peak goodput over SGLang by 1.07–1.26× (1.17× on average). On AzureLLM, for example, Chorus raises DeepSeek-R1 peak goodput from 397.3 to 470.4 tok/s. At the same offered load, the gap widens: at Chorus's best-goodput request rate, Chorus delivers 1.3–11.4× SGLang's goodput because SGLang has already entered TTFT collapse in several settings. The smallest peak gain occurs for DeepSeek-R1 on ServeGen, where sparse MoE activation keeps the GPU path efficient at moderate load; even there, Chorus shifts the best-goodput point to a 1.5× higher request rate (0.4 to 0.6 RPS) and reaches 422.1 tok/s.

Chorus also outperforms CPU-offload baselines. When serving Qwen3-32B, it improves peak goodput over NEO by 23.6% on ServeGen and 67.5% on AzureLLM; for MoE models, Chorus improves over KTransformers' best measured goodput by 89.7–428.5%. These comparisons show that relieving GPU memory pressure alone is insufficient if recurring CPU work pushes TPOT beyond its target.

SGLang loses goodput through TTFT, not TPOT. SGLang keeps decode on GPUs, so its TPOT_{P95} remains below 59 ms in all settings. Its bottleneck is admission: as offered load increases, long-request KV residency fills decode-GPU HBM, stalls new requests, and drives mean TTFT above 116 s for GLM-4.5 and DeepSeek-R1 and above 230 s for Qwen3-32B. Chorus mitigates this bottleneck by offloading selected long decodes to AMX, reclaiming GPU KV pages, and leaving prefill and short decode on the GPU fast path. It therefore keeps TPOT within the 75 ms target while delaying the TTFT collapse that limits SGLang goodput.

CPU-offload baselines lose goodput through TPOT. The CPU-offload baselines become TPOT-limited for the opposite reason: NEO and KTransformers relieve GPU memory pressure, but keep CPU work on each request's per-token execution path. For Qwen3-32B, NEO pipelines attention and CPU-resident KV-cache access with GPU computation, improving admission and lowering TTFT. However, only its lowest-load points satisfy the 75 ms TPOT target; TPOT_{P95} rises to 5.65 s on ServeGen and 1.88 s on AzureLLM at high load. KTransformers exhibits the same joint-SLO failure pattern through expert offload: only one measured point per workload is TPOT-feasible for GLM-4.5, and none is feasible for DeepSeek-R1, whose minimum TPOT_{P95} is already 89.4–120.8 ms. Thus, in these evaluated baselines, freeing GPU capacity does not improve SLO-attaining goodput when recurring CPU work causes TPOT violations. Chorus instead moves selected long-request decode to an independent AMX lane, keeping CPU work off the GPU token loop. Together, these failure modes explain Chorus's goodput advantage: it preserves a wider operating range in which both SLOs hold.

6.3 TPOT-Aware AMX Admission

This experiment answers Q2 by isolating Chorus's TPOT-aware inflight control. All policies use the same GLM-4.5 workload (RPS = 0.6), minimum offload length ($L_{\min}$ = 4K), and KV-cache ranking from §4.2; they differ only in how they set the offload quota. Over a 500-second window, Figure 11 compares **A**, which adapts its inflight limit from recent tail TPOT, with an aggressive fixed limit of 12 (**B**) and a conservative fixed limit of 3 (**C**).

Chorus effectively uses available AMX headroom without crossing the TPOT boundary. Policy B reaches the highest mean AMX throughput (90.2 tok/s) and batch size (8.7), but raises TPOT_{P95} to 98.2 ms, causing 6.6% of requests to violate the 75 ms TPOT target. Policy C avoids

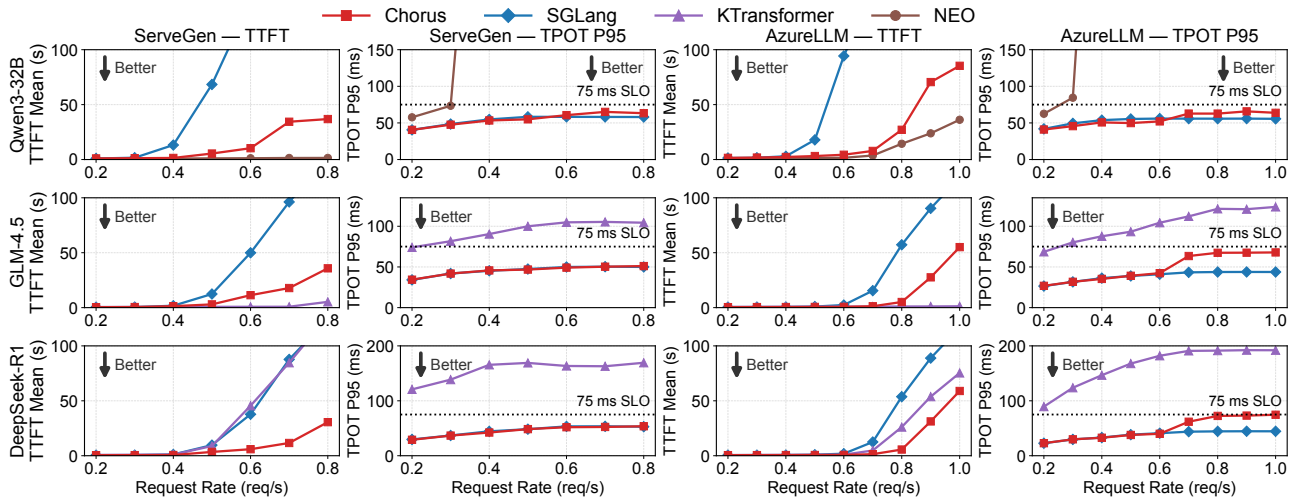


Figure 8. End-to-end mean TTFT and TPOT$_{P95}$ across models and workloads. NEO is evaluated on Qwen3-32B; KTransformers is evaluated on the MoE models.

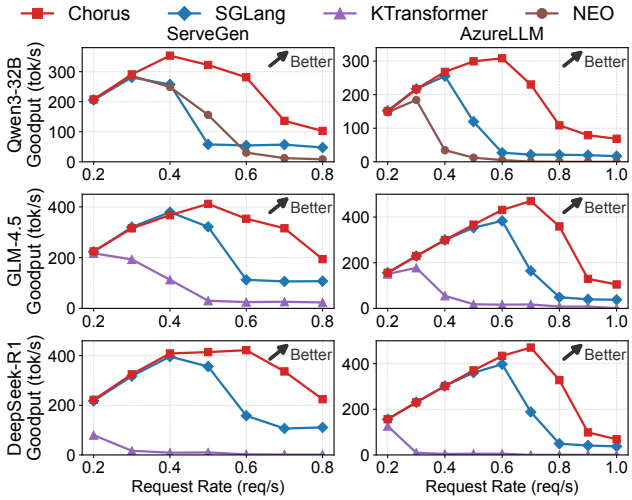


Figure 9. End-to-end SLO-attaining goodput across models and workloads. NEO is evaluated only on dense Qwen3-32B; KTransformers is evaluated only on MoE models.

violations but underuses AMX, reaching only 45.1 tok/s at a mean batch size of 2.5. Chorus sustains 74.7 tok/s at a mean batch size of 5.0—1.7× Policy C's throughput—while all offloaded requests meet the TPOT target (TPOT$_{P99}$ = 68.7 ms). Thus, TPOT-aware control captures available AMX headroom without the violations caused by aggressive fixed admission.

6.4 AMX Performance Microbenchmark

This experiment answers Q3 by testing whether distributed AMX decode can serve offloaded long-context requests within the TPOT target. Figure 10 reports average TPOT for three tested models as input length increases from 4K to 24K and batch size grows. Complementing §3, it measures whether Chorus's distributed AMX lane has enough latency headroom after a single CPU node proves too slow.

Long-context TPOT remains bounded. Across all three models, TPOT grows gradually with input length rather than collapsing at long context. At BS=1, increasing input length from 4K to 24K raises TPOT by only 8–25%: Qwen3-32B grows from 30.3 to 32.8 ms, GLM-4.5 from 35.7 to 39.0 ms, and DeepSeek-R1 from 32.1 to 40.1 ms. Distributed AMX serving keeps single-request TPOT well below the 75 ms target for the long-context requests that Chorus offloads.

Batch envelopes depend on model structure. Batching improves AMX throughput, but usable batch size depends on model structure. At 24K context, Qwen3-32B scales from BS=1 to BS=10 while remaining below the target, with TPOT rising from 32.8 to 72.3 ms. MoE models have tighter envelopes: GLM-4.5 reaches 79.1 ms at BS=6, and DeepSeek-R1 reaches 87.6 ms. DeepSeek-R1 is fast at BS=1 because sparse activation reduces single-token compute, but its TPOT grows faster with batch because larger batches activate more experts, increasing scattered expert dispatch and irregular expert-weight traffic.

The measured TPOT envelope defines safe admission. This microbenchmark explains the TPOT-safe envelope used by the admission policy evaluated above and defined in §4.2. For each model and context range, Chorus uses the largest below-target AMX batch. At 24K context, BS=4 is safe for all three models: Qwen3-32B reaches 45.1 ms, GLM-4.5 reaches 68.1 ms, and DeepSeek-R1 reaches 73.2 ms. Larger safe batches are model- and context-dependent; at 24K, GLM-4.5 exceeds 75 ms at BS=6, while DeepSeek-R1 exceeds it at BS=5. Chorus uses this profiled batch-size–context-length envelope as the initial quota, refines it with tail-TPOT feedback, and assigns the available slots to the largest KV-cache consumers to maximize GPU HBM released or avoided.

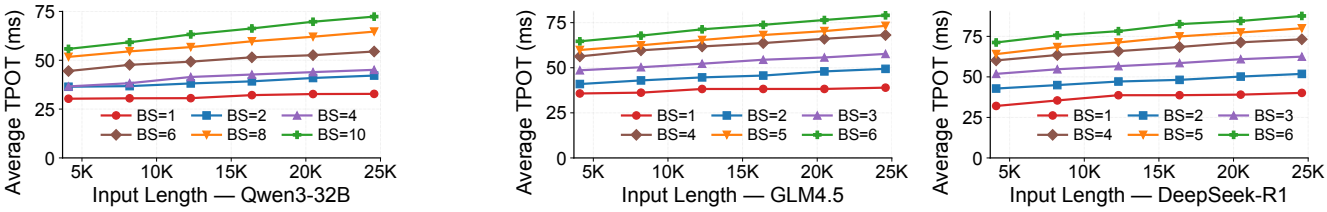


Figure 10. AMX TPOT microbenchmark across input length and batch size. Long-context TPOT remains bounded, but MoE models reach the TPOT limit at smaller batches than dense Qwen3-32B.

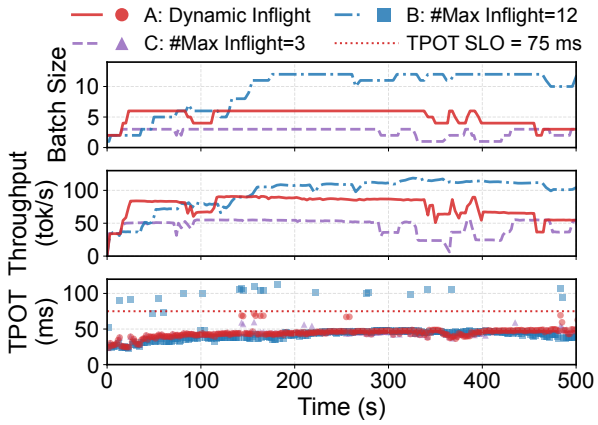


Figure 11. AMX admission over 500 seconds of the same GLM-4.5 workload (RPS = 0.6). TPOT-aware control improves over a conservative fixed cap without the violations caused by aggressive admission.

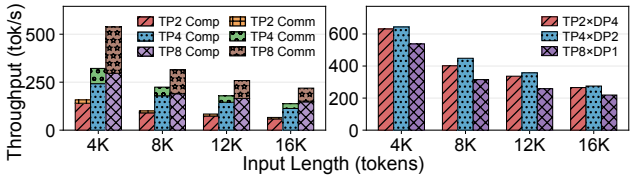


Figure 12. TP/DP tradeoff for Qwen3-8B-FP8 under $\text{TPOT}_{P99} \leq 50$ ms. The left panel stacks per-replica throughput by compute and communication share; the right panel shows eight-node aggregate throughput.

6.5 Choosing TP/DP Layouts for AMX Decode

This controlled experiment answers Q4 by validating Chorus's TP/DP layout rule (§4.3): choose the highest-throughput layout among those that satisfy TPOT, not the fastest standalone replica. We use Qwen3-8B-FP8 as a scaling probe, not to select layouts for the larger models in the end-to-end study. We impose a stricter 50 ms TPOT_{P99} target for the small model and keep only configurations that satisfy it.

Figure 12 shows the tradeoff. The left panel breaks single-replica throughput into compute and communication time shares. Wider TP adds compute and memory bandwidth, giving TP=8 the highest per-replica throughput from 4K to 16K inputs (537.8 and 219.2 tok/s), but also a larger communication share: 32.8–44.9% versus 10.1–12.4% for TP=2. At eight-node scale, TP=8 has no DP replicas, while TP=2, DP=4 has too little per-batch capacity. TP=4, DP=2 balances latency and concurrency, achieving the highest aggregate throughput at every input length: 643.7 tok/s at 4K and 275.2 tok/s at 16K. Offline profiling should maximize aggregate offloaded-decode throughput within the TPOT-feasible region.

6.6 AMX Decode-Path Optimizations

This experiment answers Q5 by isolating the three optimizations from §4.4: adaptive SDM attention, guarded MoE expert fusion, and ACAR. Each reduces latency on the offloaded AMX decode path.

Adaptive SDM attention. Figure 13 evaluates the adaptive SDM attention kernel on DeepSeek-R1 mixed-length batches containing one 16K–32K request and either one or three 8K requests. By partitioning cached prefixes in proportion to token work, adaptive SDM avoids static-SDM stragglers and TDM serialization. Across all settings, it reduces SGLang's CPU MLA latency from 509.8–940.0 μs to 221.6–329.2 μs, a 1.9–3.2× speedup, and outperforms static SDM and TDM by up to 1.5×. For the most imbalanced batch (32K/BS=4), it reduces static-SDM latency from 495.4 to 329.2 μs.

Guarded MoE expert fusion. Figure 14 evaluates the INT8 MoE execution of DeepSeek-R1 with separate and fused routed/shared expert kernels. At TP=4 and TP=8, fusion reduces BS=1–2 latency by 6.0–17.5%, where separate kernel launches and intermediate-memory traffic are difficult to amortize. The gain narrows at BS=3 and reverses at BS=4, where fusion is 3.6–5.1% slower. This crossover supports Chorus's guarded policy: fuse small batches and use separate expert kernels once larger GEMMs amortize the unfused path.

AVX-accelerated chunk-pipelined AllReduce. Figure 15 compares ACAR with UCC's AllReduce (AR) and reduce-scatter/allgather (SRA) over eight AMX servers. We set each UCC algorithm's radix to the eight-rank world size, avoiding additional recursive rounds. The faster UCC baseline is AR at 5 KB and SRA for larger messages. Across 5–100 KB, ACAR reduces latency from the best UCC latency of 26.5–74.9 μs to 11.6–67.3 μs, achieving a 1.1–2.3× speedup. These gains cover the small hidden-state messages issued by TP on the

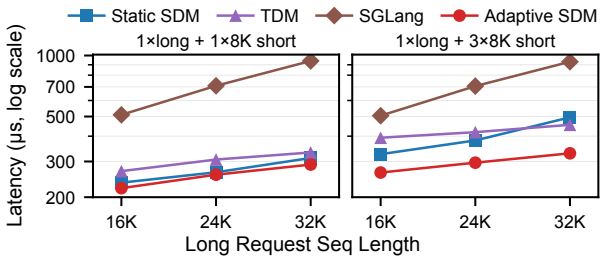


Figure 13. Decode MLA performance under mixed-length batches. Adaptive SDM balances cached-prefix work across cores, avoiding static-SDM stragglers and TDM serialization.

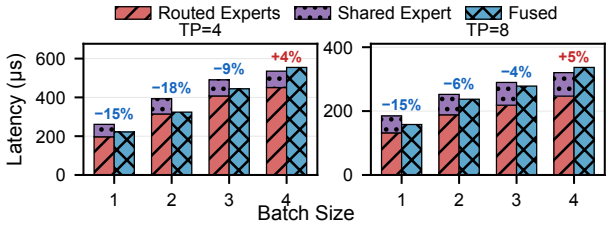


Figure 14. MoE expert fusion under TP-4 and TP-8. Fusion reduces small-batch overhead but regresses once larger batches amortize the unfused routed/shared expert kernels.

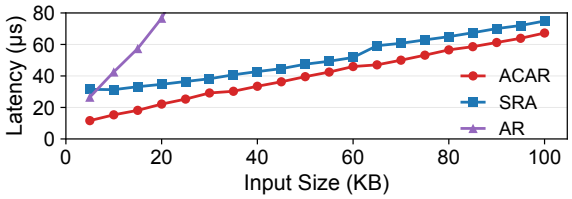


Figure 15. AllReduce latency. ACAR outperforms UCC's SRA and AR for AMX decode's small hidden-state messages.

AMX decode path, showing that ACAR reduces latency for the decode collectives on Chorus's CPU lane.

7 Related Work

Length-aware and memory-centric LLM serving. GPU serving systems improve utilization through iteration-level scheduling, paged KV-cache allocation, and prefill–decode coordination [2, 21, 54]. Other systems disaggregate prefill from decode or manage KV state across a distributed memory hierarchy [29, 30, 60]. PPD [23] uses SLO-aware placement to choose whether append-prefill in later conversation turns runs on GPU prefill or decode workers. More directly related, length-aware systems show the value of isolating long and short requests, typically through separate GPU pools [25, 55]. Chorus adopts the same isolation principle, but makes a different resource choice for the long-request lane: it removes long-request KV residency from GPU HBM rather than reserving additional GPU HBM for it. Memory-pressure techniques spill, reorganize, or compress KV state [8, 45, 49]. These approaches improve GPU scheduling or residency, but long-request decode still consumes accelerator memory or repeatedly accesses an external memory tier. Chorus instead moves both the state and computation of selected long requests to an independent CPU decode lane, preserving GPU HBM and compute for prefill and short-request decode.

CPU-assisted and heterogeneous inference. Host-assisted systems use CPU memory as a capacity tier for model weights and KV state [37, 52], or assign selected operators to CPUs, including attention and other KV-intensive work [6, 11, 13, 17] and MoE experts [7]. These designs establish the value of host resources, but adopt a different offload boundary. Memory-tier designs retain repeated state movement, whereas cooperative operator offload keeps CPU work in the GPU inference pipeline. LIA [18] partitions decoder sublayers between one GPU and its host AMX CPU, whereas Chorus performs a one-time, request-level offload to a distributed AMX group that completes the remaining decode independently of the GPU token loop.

CPU, AMX, and near-memory inference. Prior CPU inference systems improve quantized kernels, NUMA placement, and runtime scheduling for standalone or hybrid execution [22, 26, 46, 51, 57, 58]. Recent work further improves AMX efficiency through compressed execution, sparse kernels, cache-aware tiling, and accelerator sharing [1, 19, 20, 43]. Besides, near-memory accelerators offer another path to bandwidth-efficient long-context inference [14–16, 24, 27, 28]. Chorus is complementary to these execution substrates: it integrates long-prefix AMX kernels, cross-node CPU tensor parallelism, and pressure-aware request offload into an online GPU serving system. Future CPU or near-memory engines could replace the underlying decode backend without changing this request-level separation principle.

8 Conclusion

Length-skewed LLM workloads let a small tail of long requests pin KV state and constrain GPU admission for the short-request majority. Chorus relieves this pressure by keeping prefill and short-request decode on GPUs while offloading selected long-request decode to distributed AMX CPU groups. Pressure-aware offload, TP-scaled AMX decode, and long-prefix kernels keep CPU decode off the per-token GPU path while meeting TPOT targets. Across popular models and production-derived workloads, Chorus improves SLO-attaining goodput by up to 1.26× over GPU-only serving and 4.3× over CPU offload, showing that host resources can serve as a second decode lane for length-skewed serving.

Acknowledgments

We would like to thank our shepherd and the anonymous reviewers of ACM SIGOPS ATC 2026 for their insightful comments and constructive feedback, which helped improve this work. This work was supported by the HKUST-Alibaba Joint

Laboratory on Big Data and AI, RGC CRF Grant (Ref. #C6015-23G), RGC GRF Grant (Ref. #16217124), and NSFC/RGC CRS Grant (Ref. #CRS_HKUST601/24).

References

[1] Ahmed F AbouElhamayed, Jordan Dotzel, Yash Akhauri, Chi-Chih Chang, Sameh Gobriel, J Pablo Muñoz, Vui Seng Chua, Nilesh Jain, and Mohamed S Abdelfattah. 2025. Sparamx: Accelerating compressed llms token generation on amx-powered cpus. *arXiv preprint arXiv:2502.12444* (2025).

[2] Amey Agrawal, Nitin Kedia, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav S. Gulavani, Alexey Tumanov, and Ramachandran Ramjee. 2024. Taming Throughput-Latency Tradeoff in LLM Inference with Sarathi-Serve. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 2024)*. USENIX Association, 117–134. https://www.usenix.org/conference/osdi24/presentation/agrawal

[3] Anyscale. 2026. LLM Performance Optimization. https://docs.anyscale.com/llm/serving/performance-optimization. Accessed: June 5, 2026.

[4] Hannah Atmer, Yuan Yao, Thiemo Voigt, and Stefanos Kaxiras. 2025. Prefill vs. Decode Bottlenecks: SRAM-Frequency Tradeoffs and the Memory-Bandwidth Ceiling. *arXiv preprint arXiv:2512.22066* (2025).

[5] Aishwarya Bhandare, Vamsi Sripathi, Deepthi Karkada, Vivek Menon, Sunwoong Choi, Kushal Datta, and Vikram Saletore. 2019. Efficient 8-Bit Quantization of Transformer Neural Machine Language Translation Model. In *Proceedings of the 3rd Workshop on Neural Generation and Translation*. Association for Computational Linguistics, 1–5. https://aclanthology.org/D19-5601/

[6] Shiyi Cao, Shu Liu, Tyler Griggs, Peter Schafhalter, Xiaoxuan Liu, Ying Sheng, Joseph E. Gonzalez, Matei Zaharia, and Ion Stoica. 2025. 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* (Rotterdam, Netherlands) *(ASPLOS '25)*. Association for Computing Machinery, New York, NY, USA, 715–730. doi:10.1145/3669940.3707267

[7] Hongtao Chen, Weiyu Xie, Boxin Zhang, Jingqi Tang, Jiahao Wang, Jianwei Dong, Shaoyuan Chen, Ziwei Yuan, Chen Lin, Chengyu Qiu, Yuening Zhu, Qingliang Ou, Jiaqi Liao, Xianglin Chen, Zhiyuan Ai, Yongwei Wu, and Mingxing Zhang. 2025. KTransformers: Unleashing the Full Potential of CPU/GPU Hybrid Inference for MoE Models. In *Proceedings of the ACM SIGOPS 31st Symposium on Operating Systems Principles (SOSP 2025)*. ACM, 1014–1029. doi:10.1145/3731569.3764843

[8] Rongxin Cheng, Yuxin Lai, Xingda Wei, Rong Chen, and Haibo Chen. 2026. Kunserve: Parameter-centric memory management for efficient memory overloading handling in llm serving. In *Proceedings of the 21st European Conference on Computer Systems*. 1244–1260.

[9] Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. FLASHATTENTION: fast and memory-efficient exact attention with IO-awareness. In *Proceedings of the 36th International Conference on Neural Information Processing Systems* (New Orleans, LA, USA) *(NIPS '22)*. Curran Associates Inc., Red Hook, NY, USA, Article 1189, 16 pages.

[10] DeepSeek-AI, Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Haowei Zhang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Li, Hui Qu, J. L. Cai, Jian Liang, Jianzhong Guo, Jiaqi Ni, Jiashi Li, Jiawei Wang, Jin Chen, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, Junxiao Song, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Lei Xu, Leyi Xia, Liang Zhao, Litong Wang, Liyue Zhang, Meng Li, Miaojun Wang, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingming Li, Ning Tian, Panpan Huang, Peiyi Wang, Peng Zhang, Qiancheng Wang, Qihao Zhu, Qinyu Chen, Qiushi Du, R. J. Chen, R. L. Jin, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, Runxin Xu, Ruoyu Zhang, Ruyi Chen, S. S. Li, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shaoqing Wu, Shengfeng Ye, Shengfeng Ye, Shirong Ma, Shiyu Wang, Shuang Zhou, Shuiping Yu, Shunfeng Zhou, Shuting Pan, T. Wang, Tao Yun, Tian Pei, Tianyu Sun, W. L. Xiao, Wangding Zeng, Wanjia Zhao, Wei An, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, X. Q. Li, Xiangyue Jin, Xianzu Wang, Xiao Bi, Xiaodong Liu, Xiaohan Wang, Xiaojin Shen, Xiaokang Chen, Xiaokang Zhang, Xiaosha Chen, Xiaotao Nie, Xiaowen Sun, Xiaoxiang Wang, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xingkai Yu, Xinnan Song, Xinxia Shan, Xinyi Zhou, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, Y. K. Li, Y. Q. Wang, Y. X. Wei, Y. X. Zhu, Yang Zhang, Yanhong Xu, Yanhong Xu, Yanping Huang, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Li, Yaohui Wang, Yi Yu, Yi Zheng, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Ying Tang, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yu Wu, Yuan Ou, Yuchen Zhu, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yukun Zha, Yunfan Xiong, Yunxian Ma, Yuting Yan, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Z. F. Wu, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhen Huang, Zhen Zhang, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhibin Gou, Zhicheng Ma, Zhigang Yan, Zhihong Shao, Zhipeng Xu, Zhiyu Wu, Zhongyu Zhang, Zhuoshu Li, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Ziyi Gao, and Zizheng Pan. 2024. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437* (2024).

[11] Jiakun Fan, Yanglin Zhang, Xiangchen Li, and Dimitrios S Nikolopoulos. 2025. APEX: Asynchronous Parallel CPU-GPU Execution for Online LLM Inference on Constrained GPUs. *arXiv preprint arXiv:2506.03296* (2025).

[12] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Hanwei Xu, Honghui Ding, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jingchang Chen, Jingyang Yuan, Jinhao Tu, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaichao You, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingxu Zhou, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu

Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. 2025. DeepSeek-R1 incentivizes reasoning in LLMs through reinforcement learning. *Nature* 645, 8081 (2025), 633–638. doi:10.1038/s41586-025-09422-z

[13] Jiaao He and Jidong Zhai. 2024. Fastdecode: High-throughput gpu-efficient llm serving using heterogeneous pipelines. *arXiv preprint arXiv:2403.11421* (2024).

[14] Yintao He, Haiyu Mao, Christina Giannoula, Mohammad Sadrosadati, Juan Gómez-Luna, Huawei Li, Xiaowei Li, Ying Wang, and Onur Mutlu. 2025. Papi: Exploiting dynamic parallelism in large language model decoding with a processing-in-memory-enabled computing system. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*. 766–782.

[15] Guseul Heo, Sangyeop Lee, Jaehong Cho, Hyunmin Choi, Sanghyeon Lee, Hyungkyu Ham, Gwangsun Kim, Divya Mahajan, and Jongse Park. 2024. NeuPIMs: NPU-PIM Heterogeneous Acceleration for Batched LLM Inferencing. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3* (La Jolla, CA, USA) *(ASPLOS '24)*. Association for Computing Machinery, New York, NY, USA, 722–737. doi:10.1145/3620666.3651380

[16] Yang Hong, Junlong Yang, Bo Peng, and Jianguo Yao. 2026. REPA: Reconfigurable PIM for the Joint Acceleration of KV Cache Offloading and Processing. In *Proceedings of the 31st ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2* (USA) *(ASPLOS '26)*. Association for Computing Machinery, New York, NY, USA, 1622–1639. doi:10.1145/3779212.3790212

[17] Xuanlin Jiang, Yang Zhou, Shiyi Cao, Ion Stoica, and Minlan Yu. 2025. NEO: Saving GPU Memory Crisis with CPU Offloading for Online LLM Inference. In *Proceedings of Machine Learning and Systems*, M. Zaharia, G. Joshi, and Y. Lin (Eds.), Vol. 7. MLSys. https://proceedings.mlsys.org/paper_files/paper/2025/file/66a026c0d17040889b50f0dfa650e5e0-Paper-Conference.pdf

[18] Hyungyo Kim, Nachuan Wang, Qirong Xia, Jinghan Huang, Amir Yazdanbakhsh, and Nam Sung Kim. 2025. 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 (ISCA '25)*. Association for Computing Machinery, New York, NY, USA, 544–558. doi:10.1145/3695053.3731092

[19] Hyungyo Kim, Qirong Xia, Jinghan Huang, Nachuan Wang, Younjoo Lee, Jung Ho Ahn, Wajdi K Feghali, Ren Wang, and Nam Sung Kim. 2026. LiLo: Harnessing the on-Chip Accelerators in Intel CPUs for Compressed LLM Inference Acceleration. In *IEEE International Symposium on High Performance Computer Architecture (HPCA)*. https://ieeexplore.ieee.org/document/11408577/

[20] Hyungyo Kim, Gaohan Ye, Nachuan Wang, Amir Yazdanbakhsh, and Nam Sung Kim. 2024. Exploiting Intel Advanced Matrix Extensions (AMX) for Large Language Model Inference. *IEEE Computer Architecture Letters* (2024). https://ieeexplore.ieee.org/document/10538369/?arnumber=10538369

[21] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient Memory Management for Large Language Model Serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles (SOSP 2023)*. ACM, 611–626. doi:10.1145/3600006.3613165

[22] Suyi Li, Hanfeng Lu, Tianyuan Wu, Minchen Yu, Qizhen Weng, Xusheng Chen, Yizhou Shan, Binhang Yuan, and Wei Wang. 2025. Toppings: CPU-Assisted, Rank-Aware Adapter Serving for LLM Inference. In *2025 USENIX Annual Technical Conference (USENIX ATC 25)*. USENIX Association, Boston, MA, 613–629. https://www.usenix.org/conference/atc25/presentation/li-suyi-toppings

[23] Zongze Li, Jingyu Liu, Zhen Xu, Yineng Zhang, Tahseen Rabbani, and Ce Zhang. 2026. Not All Prefills Are Equal: PPD Disaggregation for Multi-turn LLM Serving. *arXiv preprint arXiv:2603.13358* (2026). Accepted at ICML 2026. https://arxiv.org/abs/2603.13358

[24] Qingyuan Liu, Liyan Chen, Yanning Yang, Haocheng Wang, Dong Du, Zhigang Mao, Naifeng Jing, Yubin Xia, and Haibo Chen. 2025. L3: DIMM-PIM Integrated Architecture and Coordination for Scalable Long-Context LLM Inference. *arXiv preprint arXiv:2504.17584* (2025).

[25] Xunzhuo Liu, Bowei He, Xue Liu, Andy Luo, Haichen Zhang, and Huamin Chen. 2026. Dual-pool token-budget routing for cost-efficient and reliable llm serving. *arXiv preprint arXiv:2604.08075* (2026).

[26] Seonjin Na, Geonhwa Jeong, Byung Hoon Ahn, Jeffrey Young, Tushar Krishna, and Hyesoon Kim. 2024. Understanding Performance Implications of LLM Inference on CPUs. In *IEEE International Symposium on Workload Characterization (IISWC)*. https://ieeexplore.ieee.org/document/10763564/?arnumber=10763564&tag=1

[27] Yudong Pan, Yintao He, Tianhua Han, Lian Liu, Shixin Zhao, Zhirong Chen, Mengdi Wang, Cangyuan Li, Yinhe Han, and Ying Wang. 2026. TriMoE: Augmenting GPU with AMX-Enabled CPU and DIMM-NDP for High-Throughput MoE Inference via Offloading. *arXiv preprint arXiv:2603.01058* (2026).

[28] Jaehyun Park, Jaewan Choi, Kwanhee Kyung, Michael Jaemin Kim, Yongsuk Kwon, Nam Sung Kim, and Jung Ho Ahn. 2024. AttAcc! Unleashing the Power of PIM for Batched Transformer-based Generative Model Inference. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*. 103–119. doi:10.1145/3620665.3640422

[29] Pratyush Patel, Esha Choukse, Chaojie Zhang, Aashaka Shah, Íñigo Goiri, Saeed Maleki, and Ricardo Bianchini. 2025. Splitwise: Efficient Generative LLM Inference Using Phase Splitting. In *Proceedings of the 51st Annual International Symposium on Computer Architecture* (Buenos Aires, Argentina) *(ISCA '24)*. IEEE Press, 118–132. doi:10.1109/ISCA59077.2024.00019

[30] Ruoyu Qin, Zheming Li, Weiran He, Jialei Cui, Feng Ren, Mingxing Zhang, Yongwei Wu, Weimin Zheng, and Xinran Xu. 2025. 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)*. USENIX Association, Santa Clara, CA, 155–170. https://www.usenix.org/conference/fast25/presentation/qin

[31] Rolf Rabenseifner. 2004. Optimization of Collective Reduction Operations. In *Computational Science - ICCS 2004*, Marian Bubak, Geert Dick van Albada, Peter M. A. Sloot, and Jack Dongarra (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 1–9.

[32] Pol G. Recasens, Ferran Agullo, Yue Zhu, Chen Wang, Eun Kyung Lee, Olivier Tardieu, Jordi Torres, and Josep Ll. Berral. 2025. Mind the Memory Gap: Unveiling GPU Bottlenecks in Large-Batch LLM Inference . In *2025 IEEE 18th International Conference on Cloud Computing (CLOUD)*. IEEE Computer Society, Los Alamitos, CA, USA, 277–287. doi:10.1109/CLOUD67622.2025.00036

[33] Kyeonghyeon Ryu, Yunhyeong Jeon, Gyeongseo Park, Minho Kim, and Daehoon Kim. 2026. InfAMAX: Bridging the Compute-Memory Gap in Intel AMX for Efficient LLM Inference . *IEEE Computer Architecture Letters* 25, 01 (Jan. 2026), 198–201. doi:10.1109/LCA.2026.3693303

[34] Sglang and Intel team. 2026. Cost Effective Deployment of DeepSeek R1 with Intel® Xeon® 6 CPU on SGLang. https://www.lmsys.org/blog/2025-07-14-intel-xeon-optimization. Accessed: 2026-06-05.

[35] Sglang team. 2026. Together with SGLang: Best Practices for Serving DeepSeek-R1 on H20-96G. https://www.lmsys.org/blog/2025-09-26-sglang-antgroup. Accessed: 2026-06-05.

[36] Haoran Shen, Naveen Mellempudi, Xin He, Qun Gao, Chang Wang, and Mengni Wang. 2024. Efficient Post-training Quantization with FP8 Formats. In *Proceedings of Machine Learning and Systems*, Vol. 6. https://proceedings.mlsys.org/paper_files/paper/2024/hash/

dea9b4b6f55ae611c54065d6fc750755-Abstract-Conference.html

[37] Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Beidi Chen, Percy Liang, Christopher Ré, Ion Stoica, and Ce Zhang. 2023. FlexGen: high-throughput generative inference of large language models with a single GPU. In *Proceedings of the 40th International Conference on Machine Learning* (Honolulu, Hawaii, USA) *(ICML'23)*. JMLR.org, Article 1288, 23 pages.

[38] Jovan Stojkovic, Chaojie Zhang, Íñigo Goiri, Josep Torrellas, and Esha Choukse. 2025. DynamoLLM: Designing LLM Inference Clusters for Performance and Energy Efficiency. In *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 1348–1362. doi:10.1109/HPCA61900.2025.00102

[39] GLM-4.5 Team, Aohan Zeng, Xin Lv, Qinkai Zheng, Zhenyu Hou, Bin Chen, Chengxing Xie, Cunxiang Wang, Da Yin, Hao Zeng, Jiajie Zhang, Kedong Wang, Lucen Zhong, Mingdao Liu, Rui Lu, Shulin Cao, Xiaohan Zhang, Xuancheng Huang, Yao Wei, Yean Cheng, Yifan An, Yilin Niu, Yuanhao Wen, Yushi Bai, Zhengxiao Du, Zihan Wang, Zilin Zhu, Bohan Zhang, Bosi Wen, Bowen Wu, Bowen Xu, Can Huang, Casey Zhao, Changpeng Cai, Chao Yu, Chen Li, Chendi Ge, Chenghua Huang, Chenhui Zhang, Chenxi Xu, Chenzheng Zhu, Chuang Li, Congfeng Yin, Daoyan Lin, Dayong Yang, Dazhi Jiang, Ding Ai, Erle Zhu, Fei Wang, Gengzheng Pan, Guo Wang, Hailong Sun, Haitao Li, Haiyang Li, Haiyi Hu, Hanyu Zhang, Hao Peng, Hao Tai, Haoke Zhang, Haoran Wang, Haoyu Yang, He Liu, He Zhao, Hongwei Liu, Hongxi Yan, Huan Liu, Huilong Chen, Ji Li, Jiajing Zhao, Jiamin Ren, Jian Jiao, Jiani Zhao, Jianyang Yan, Jiaqi Wang, Jiayi Gui, Jiayue Zhao, Jie Liu, Jijie Li, Jing Li, Jing Lu, Jingsen Wang, Jingwei Yuan, Jingxuan Li, Jingzhao Du, Jinhua Du, Jinxin Liu, Junkai Zhi, Junli Gao, Ke Wang, Lekang Yang, Liang Xu, Lin Fan, Lindong Wu, Lintao Ding, Lu Wang, Man Zhang, Minghao Li, Minghuan Xu, Mingming Zhao, Mingshu Zhai, Pengfan Du, Qian Dong, Shangde Lei, Shangqing Tu, Shangtong Yang, Shaoyou Lu, Shijie Li, Shuang Li, Shuang-Li, Shuxun Yang, Sibo Yi, Tianshu Yu, Wei Tian, Weihan Wang, Wenbo Yu, Weng Lam Tam, Wenjie Liang, Wentao Liu, Xiao Wang, Xiaohan Jia, Xiaotao Gu, Xiaoying Ling, Xin Wang, Xing Fan, Xingru Pan, Xinyuan Zhang, Xinze Zhang, Xiuqing Fu, Xunkai Zhang, Yabo Xu, Yandong Wu, Yida Lu, Yidong Wang, Yilin Zhou, Yiming Pan, Ying Zhang, Yingli Wang, Yingru Li, Yinpei Su, Yipeng Geng, Yitong Zhu, Yongkun Yang, Yuhang Li, Yuhao Wu, Yujiang Li, Yunan Liu, Yunqing Wang, Yuntao Li, Yuxuan Zhang, Zezhen Liu, Zhen Yang, Zhengda Zhou, Zhongpei Qiao, Zhuoer Feng, Zhuorui Liu, Zichen Zhang, Zihan Wang, Zijun Yao, Zikang Wang, Ziqiang Liu, Ziwei Chai, Zixuan Li, Zuodong Zhao, Wenguang Chen, Jidong Zhai, Bin Xu, Minlie Huang, Hongning Wang, Juanzi Li, Yuxiao Dong, and Jie Tang. 2025. Glm-4.5: Agentic, reasoning, and coding (arc) foundation models. *arXiv preprint arXiv:2508.06471* (2025).

[40] Rajeev Thakur, Rolf Rabenseifner, and William Gropp. 2005. Optimization of Collective Communication Operations in MPICH. *Int. J. High Perform. Comput. Appl.* 19, 1 (Feb. 2005), 49–66. doi:10.1177/1094342005051521

[41] Manjunath Gorentla Venkata, Valentine Petrov, Sergey Lebedev, Devendar Bureddy, Ferrol Aderholdt, Joshua Ladd, Gil Bloch, Mike Dubman, and Gilad Shainer. 2025. Unified Collective Communication: A Unified Library for CPU, GPU, and DPU Collectives. *IEEE Micro* 45, 2 (2025), 26–35. doi:10.1109/MM.2025.3534638

[42] vllm team. 2026. GuideLLM Service Level Objectives. https://github.com/vllm-project/guidellm/blob/main/docs/guides/service_level_objectives.md. Accessed: 2026-06-05.

[43] Xinkai Wang, Chao Li, Yiming Zhuansun, Jinyang Guo, Xiaofeng Hou, Jing Wang, Luping Wang, Weigao Chen, Cheng Huang, Guodong Yang, Liping Zhang, and Minyi Guo. 2026. AUM: Unleashing the Efficiency Potential of Shared Processors with Accelerator Units for LLM Serving. In *IEEE International Symposium on High Performance Computer Architecture (HPCA)*. https://2026.hpca-conf.org/details/hpca-2026-main-conference/36/AUM-Unleashing-the-Efficiency-Potential-of-Shared-Processors-with-Accelerator-Units-

[44] Yuxin Wang, Yuhan Chen, Zeyu Li, Xueze Kang, Yuchu Fang, Yeju Zhou, Yang Zheng, Zhenheng Tang, Xin He, Rui Guo, Xin Wang, Qiang Wang, Amelie Chi Zhou, and Xiaowen Chu. 2025. 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* (Toronto ON, Canada) *(KDD '25)*. Association for Computing Machinery, New York, NY, USA, 5831–5841. doi:10.1145/3711896.3737413

[45] Gao Wei, Xinyu Zhou, Peng Sun, Tianwei Zhang, and Yonggang Wen. 2025. Rethinking Key-Value Cache Compression Techniques for Large Language Model Serving. *Proceedings of Machine Learning and Systems* 7 (2025).

[46] Jianyu Wei, Shijie Cao, Ting Cao, Lingxiao Ma, Lei Wang, Yanyong Zhang, and Mao Yang. 2025. T-MAC: CPU Renaissance via Table Lookup for Low-Bit LLM Deployment on Edge. In *Proceedings of the Twentieth European Conference on Computer Systems* (Rotterdam, Netherlands) *(EuroSys '25)*. Association for Computing Machinery, New York, NY, USA, 278–292. doi:10.1145/3689031.3696099

[47] Yuxing Xiang, Xue Li, Kun Qian, Yan Zhang, Wenyuan Yu, Ennan Zhai, Xin Jin, and Jingren Zhou. 2026. ServeGen: Workload Characterization and Generation of Large Language Model Serving in Production. In *23rd USENIX Symposium on Networked Systems Design and Implementation (NSDI 26)*. USENIX Association, Renton, WA, 1845–1859. https://www.usenix.org/conference/nsdi26/presentation/xiang-servegen

[48] Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. 2023. SmoothQuant: accurate and efficient post-training quantization for large language models. In *Proceedings of the 40th International Conference on Machine Learning* (Honolulu, Hawaii, USA) *(ICML'23)*. JMLR.org, Article 1585, 13 pages.

[49] Zhiqiang Xie, Ziyi Xu, Mark Zhao, Yuwei An, Vikram Sharma Mailthody, Scott Mahlke, Michael Garland, and Christos Kozyrakis. 2025. Strata: Hierarchical context caching for long context language model serving. *arXiv preprint arXiv:2508.18572* (2025).

[50] Chuhao Xu, Zijun Li, Quan Chen, Han Zhao, Xueyan Tang, and Minyi Guo. 2026. Towards Resource-Efficient Serverless LLM Inference with SLINFER. In *2026 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 1–18. doi:10.1109/HPCA68181.2026.11408548

[51] Yuzhuang Xu, Xu Han, Yuxuan Li, and Wanxiang Che. 2026. ArcLight: A Lightweight LLM Inference Architecture for Many-Core CPUs. *arXiv preprint arXiv:2603.07770* (2026).

[52] Yi Xu, Ziming Mao, Xiangxi Mo, Shu Liu, and Ion Stoica. 2024. Pie: Pooling cpu memory for llm inference. *arXiv preprint arXiv:2411.09317* (2024).

[53] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388* (2025).

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

[55] Yitao Yuan, Chenqi Zhao, Bohan Zhao, Zane Cao, Yongchao He, and Wenfei Wu. 2025. CascadeInfer: Low-Latency and Load-Balanced LLM

Serving via Length-Aware Scheduling. *arXiv preprint arXiv:2512.19179* (2025).

[56] Sungmin Yun, Seonyong Park, Hwayong Nam, Younjoo Lee, Gunjun Lee, Kwanhee Kyung, Sangpyo Kim, Nam Sung Kim, Jongmin Kim, Hyungyo Kim, et al. 2025. The new LLM bottleneck: A systems perspective on latent attention and mixture-of-experts. *arXiv preprint arXiv:2507.15465* (2025).

[57] Tianyi Zhang, Jonah Yi, Bowen Yao, Zhaozhuo Xu, and Anshumali Shrivastava. 2024. NoMAD-attention: efficient LLM inference on CPUs through multiply-add-free attention. In *Proceedings of the 38th International Conference on Neural Information Processing Systems* (Vancouver, BC, Canada) *(NIPS '24)*. Curran Associates Inc., Red Hook, NY, USA, Article 3581, 25 pages.

[58] Juntao Zhao, Jiuru Li, and Chuan Wu. 2025. Sandwich: Separating Prefill-Decode Compilation for Efficient CPU LLM Serving. *arXiv preprint arXiv:2507.18454* (2025).

[59] Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. 2024. SGLang: efficient execution of structured language model programs. In *Proceedings of the 38th International Conference on Neural Information Processing Systems* (Vancouver, BC, Canada) *(NIPS '24)*. Curran Associates Inc., Red Hook, NY, USA, Article 2000, 27 pages.

[60] Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. 2024. DistServe: Disaggregating Prefill and Decoding for Goodput-optimized Large Language Model Serving. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 2024)*. USENIX Association, 193–210. https://www.usenix.org/conference/osdi24/presentation/zhong-yinmin