

LLM Inference as a Moving Bottleneck: Memory, Scheduling, and Service-Level Objectives

Sean Fraser

14 October 2026

Abstract

This paper develops a bottleneck-oriented framework for reasoning about autoregressive large language model inference. The central difficulty is that prompt processing, token generation, cache growth, and request scheduling stress different resources, so an optimization that accelerates one phase can expose a different limiting factor. We synthesize ten primary research papers and derive explicit models of transformer computation, key-value cache capacity, decode arithmetic intensity, speculative decoding, and prefill-decode transfer. Evidence consists of verified literature and reproducible analytical calculations, not new GPU benchmarks. In an illustrative dense-model configuration, replacing 32 key-value heads with eight reduces a 16-request, 8,192-token FP16 cache from 64 GiB to 16 GiB; this capacity reduction does not itself establish a fourfold latency improvement.

Introduction

A language-model server can be fast at prompt processing and slow at generation, fit its weights in memory yet fail under concurrent requests, or report high aggregate throughput while users wait too long between tokens. These are not contradictory observations. They arise because inference is a sequence of interacting resource demands rather than a single matrix multiplication.

Three distinctions organize this paper. First, **prefill** processes a prompt in parallel over its token positions, whereas **decode** repeatedly extends each live sequence. Second, memory **capacity** limits which workloads can remain resident, while memory **bandwidth** limits how quickly their data can be consumed. Third, useful service capacity is constrained by latency objectives, not merely by tokens per second. PagedAttention addresses dynamic cache allocation [Kwon et al., 2023]; FlashAttention addresses attention data movement [Dao et al., 2022]; and DistServe addresses phase interference and resource allocation [Zhong et al., 2024]. Treating these as interchangeable forms of “faster attention” obscures what each changes.

Our contribution is a pedagogical synthesis, not a claim of a new inference algorithm. We connect a resource model to a practical optimization decision process. Specifically, we (i) derive phase-specific work and cache estimates; (ii) classify optimization mechanisms by resource and semantic effect; and (iii) supply reproducible numerical examples and an evaluation protocol that separates arithmetic estimates from empirical evidence.

The scope is dense, decoder-only transformers with conventional full causal attention and an explicit key-value (KV) cache. Grouped-query attention is included. Mixture-of-experts routing, latent attention, state-space architectures, multimodal encoders, and cross-attention require modified

equations. The literature selection emphasizes foundational mechanisms published in 2022–2025, extended with an October 2026 optical KV-network case study supplied by the reader; it is neither a systematic review nor an exhaustive account of developments through October 2026.

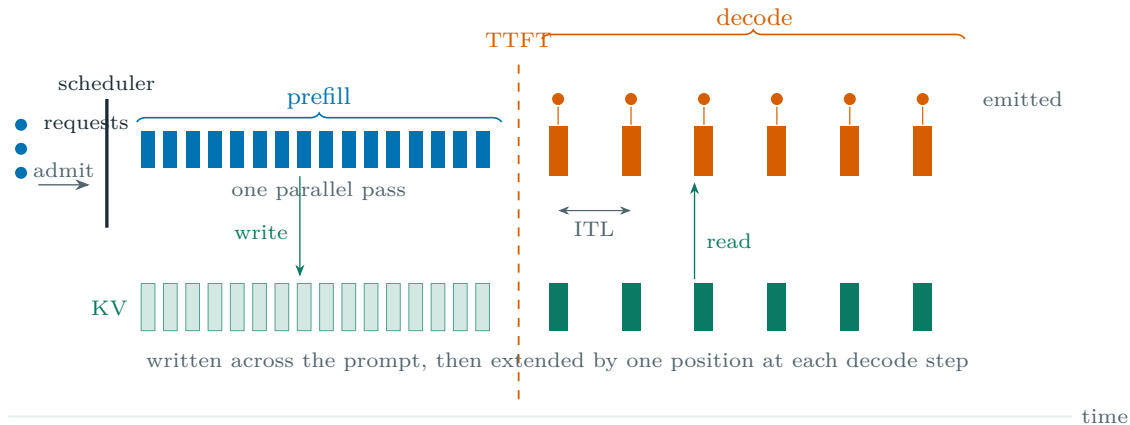


Figure 1: Serving timeline for one request. Prefill processes the prompt in one pass and writes the KV cache. Each decode step emits one token and appends one cache position. TTFT marks the phase boundary; ITL is the gap between emitted tokens. Control does not return to prefill. Synthesis of Sarathi-Serve, DistServe, and PagedAttention; not a deployed topology.

Research method and evidence boundaries

We selected eight foundational papers for complementary mechanisms: IO-aware attention, cache allocation, grouped-query attention, weight quantization, speculative decoding, chunked-prefill scheduling, phase disaggregation, and attention-kernel specialization. For these eight, the paper title and bibliographic metadata were retrieved from arXiv and the corresponding DataCite DOI record. Abstracts and full arXiv HTML text were saved as an evidence archive. BibTeX entries were obtained programmatically through DOI content negotiation, with a metadata-based fallback. The expanded disaggregation section adds Mooncake from the USENIX FAST 2025 publication page and Yoon and Yu’s October 2026 preprint from the reader-supplied version-1 PDF. The latter’s title and authors were checked against arXiv metadata; its DOI was marked pending registration, so it is not claimed to have passed a DataCite cross-check. The supplied PDF’s extracted text and checksum are archived. Results cited from that paper remain the authors’ reported measurements, simulations, or cost estimates, not independently reproduced results.

The analysis distinguishes three evidence classes:

- **Literature findings:** a mechanism or result explicitly attributed to its original paper. Reported speedups are not transferred to our illustrative workload.
- **Derived quantities:** values calculated from the equations and declared input parameters. These are deterministic arithmetic outputs, not measurements.
- **Proposed evaluation:** experiments needed to validate a deployment. These experiments were not run for this paper.

No model was trained, no inference GPU was benchmarked, and no latency trace was collected. Consequently, this paper makes no empirical claim about an engine ranking, production cost,

accuracy loss, or a hardware-specific throughput ceiling. Its falsifiable value is in whether the models correctly predict which resource should be measured next.

Autoregressive inference and its cost structure

Token dependency and attention

For a prompt x and generated tokens $y_1, \dots, y_T$, autoregressive inference factors the conditional distribution as

$$p_\theta(y_{1:T} \mid x) = \prod_{t=1}^T p_\theta(y_t \mid x, y_{<t}).$$

The probability factorization creates a sequential dependence across generated positions. It does not prohibit parallel verification of proposed tokens, which is the opportunity exploited by speculative decoding [Leviathan et al., 2022]. Within a causal attention layer,

$$\text{Attn}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_h}} + \mathcal{M}\right)V,$$

where $\mathcal{M}$ masks future positions and d_h is the head dimension. Cached keys and values avoid recomputing past token representations during ordinary autoregressive generation. They do not eliminate reading the historical state to compute new attention outputs.

Let L denote layers, d hidden width, h_q query heads, h_{kv} KV heads, $d_h = d/h_q$, and d_{ff} feed-forward width. For an ordinary two-matrix feed-forward block, the projection and feed-forward multiply-add work per token is approximately

$$F_{\text{linear}} \simeq 2L \left(2d^2 + 2d h_{kv} d_h + 2d d_{ff} \right).$$

We count a multiply-add as two floating-point operations. The two d^2 terms correspond to query and output projections; keys and values each project to $h_{kv} d_h$. For a gated feed-forward network, replace $2d d_{ff}$ inside the parentheses with $3d d_{ff}$. Embedding lookup, output vocabulary projection, normalization, nonlinearities, and positional transformations are omitted. Architecture-specific parameter counts must therefore replace these approximations when precision matters.

Prefill versus decode

For a batch of B equal-length prompts of S tokens, causal attention performs work proportional to $BLdS^2$. Using triangular causal evaluation gives the leading estimate

$$F_{\text{prefill}} \simeq BSF_{\text{linear}} + 2BLdS^2.$$

This leading coefficient assumes the kernel avoids masked work; a kernel computing a full square attention product performs more work. At decode, with S cached positions per sequence, one new query token per request has approximate work

$$F_{\text{decode}} \simeq BF_{\text{linear}} + 4BLdS.$$

The quadratic term is per prompt, not per generated token. These expressions explain why sufficiently large prefills can expose abundant matrix-level parallelism while a small decode batch reuses each weight across comparatively few token positions. They do **not** prove that all prefill is compute-bound or all decode is bandwidth-bound. Short prompts, long-context attention, kernel occupancy, batch size, and communication can change the dominant resource. Sarathi-Serve explicitly exploits the complementary utilization patterns of prefill and decode [Agrawal et al., 2024].

KV cache: capacity is not bandwidth

A cache accounting model

For equal-length sequences and a uniform KV element size b_{kv} bytes, the resident cache payload is

$$M_{KV} = 2BLS h_{kv} d_h b_{kv}.$$

The factor two accounts for keys and values. For unequal sequence lengths, replace BS with $\sum_i S_i$. This is logical payload, not allocator-reserved memory. A resident system must satisfy

$$M_W + M_{KV} + M_{\text{workspace}} + M_{\text{runtime}} \leq M_{\text{device}},$$

where M_W includes weight storage and any quantization metadata. Tensor-parallel cache storage depends on whether KV heads are sharded or replicated; dividing all memory terms by the device count is not generally valid.

GQA shares key and value heads across groups of query heads and therefore reduces the h_{kv} factor [Ainslie et al., 2023]. Holding other variables fixed, its ideal cache ratio relative to multi-head attention (MHA) is h_{kv}/h_q . This is an architectural choice, not a lossless runtime switch for an arbitrary pretrained checkpoint. Ainslie et al. study uptraining existing checkpoints as well as the quality–efficiency tradeoff [Ainslie et al., 2023].

Worked capacity example

We choose a hypothetical configuration: $L = 32$, $h_q = 32$, $d_h = 128$, FP16 KV with $b_{kv} = 2$, $B = 16$, and $S = 8192$. The choices illustrate scaling and do not identify a specific released model. The number of parameters used later is an independent illustrative weight assumption, not inferred from these dimensions.

KV layout	KV heads	Cache per token per request	Cache at $16 \times 8,192$ tokens
MHA	32	512 KiB	64 GiB
GQA	8	128 KiB	16 GiB
MQA	1	16 KiB	2 GiB

All values are calculated payloads; KiB and GiB use powers of two. The comparison supports a **capacity** claim only. It says nothing by itself about achieved token latency, model quality, or kernel support. An engine that replicates KV heads across tensor-parallel ranks can also realize a different physical saving.

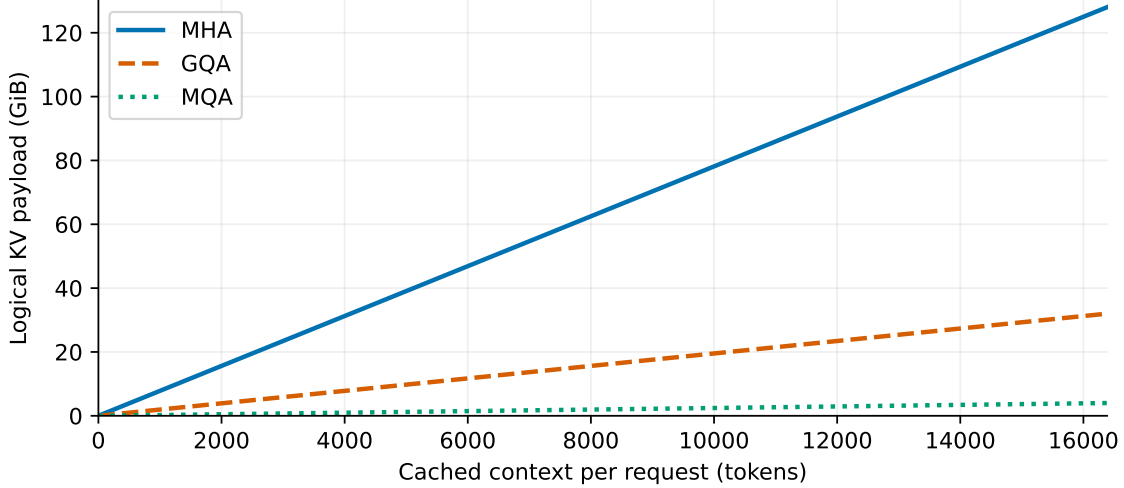


Figure 2: Calculated KV payload versus cached sequence length for the declared 16-request configuration. All curves come from the same dataset as the capacity table; axes start at zero. No measurements or fitted performance curves are shown.

Paging and prefix reuse

PagedAttention maps logical cache blocks to non-contiguous physical blocks, allowing dynamic allocation and sharing without reserving a contiguous maximum-length region for each request [Kwon et al., 2023]. It improves allocation efficiency; it does not reduce the mathematical number of KV elements required by an unshared sequence.

With block size P tokens, request i allocates $P\lceil S_i/P \rceil$ token slots. Its tail waste is less than P slots, so total tail slack is bounded by $B(P - 1)$ slots. This elementary bound excludes metadata, free-pool reservation, replication, and other fragmentation sources. Larger blocks reduce metadata and can improve access efficiency, but increase worst-case tail slack.

For R requests with an identical immutable prefix of S_p tokens, ideal sharing changes prefix storage from RmS_p to mS_p , where $m = 2Lh_{kv}d_hb_{kv}$. The maximum prefix-payload saving is therefore $(R - 1)mS_p$. Real systems must establish cache identity using all model-relevant inputs, not only visible text. Model revisions, adapters, position conventions, and tenant boundaries can invalidate reuse or make it unsafe. Eviction policies also determine whether a theoretically reusable prefix is present when needed.

A bandwidth–compute model for decoding

Let F be floating-point work, D bytes transferred through the bottleneck memory interface, C achievable compute rate, and β achievable bandwidth. A resource lower bound is

$$t_{\text{step}} \geq \max\left(\frac{F}{C}, \frac{D}{\beta}\right).$$

This roofline-style bound omits launch delays, synchronization, and imperfect overlap. It is not an equality or a latency forecast. The relevant C depends on datatype, matrix shape, and the available kernel; β is effective bandwidth, not necessarily a data-sheet peak.

For an illustrative dense model with N active parameters, assume each decode step streams all weights once and reads the full cache once. Approximate

$$D \simeq Nb_w + BmS, \quad I_{\text{weights+KV}} \simeq \frac{2BN}{Nb_w + BmS}.$$

Here b_w is the average bytes per weight and m is cache bytes per token per request. The numerator deliberately keeps only dense parameter work, so this intensity proxy undercounts long-context attention FLOPs. KV writes, activation traffic, dequantization metadata, repeated reads, and cache locality are also excluded. It is useful for separating weight amortization from cache traffic, not as a complete hardware model.

In the weight-only limit, $I \simeq 2B/b_w$: batching amortizes weight reads. At fixed nonzero context length, the proxy instead saturates at $2N/(mS)$ as B grows. At long contexts, including attention work changes that limiting intensity; the saturation derived here applies only to the stated dense-work proxy.

For $N = 8 \times 10^9$, $b_w = 2$, GQA $m = 131072$ bytes and $S = 8192$, weight traffic is 16 GB per step and KV reads contribute 1 GiB per live request. Equal modeled traffic occurs at

$$B_{\text{equal}} = \frac{Nb_w}{mS} \simeq 14.90.$$

GB uses decimal bytes; the equality is calculated in bytes. This is **not** the compute–bandwidth crossover. At $B = 16$, the assumed weight-plus-KV traffic is about 33.18 GB; at an assumed effective bandwidth of 2 TB/s it yields a memory-time floor of about 16.59 ms. The bandwidth is an illustrative input, not a measurement of any GPU. A service cannot promise that latency without measuring the omitted work and queueing.

Optimization mechanisms and their interactions

Attention kernels and quantization

FlashAttention tiles exact attention to reduce transfers between high-bandwidth memory and on-chip SRAM without materializing the full attention matrix [Dao et al., 2022]. “Exact” distinguishes it from an algorithmic approximation; floating-point accumulation order can still change numerical results. It does not remove the pairwise interactions of full attention, and its original paper’s training speedups must not be interpreted as universal decode speedups.

FlashInfer extends attention-kernel engineering toward heterogeneous cache layouts, customizable attention, and load balancing for dynamic serving workloads [Ye et al., 2025]. These concerns matter because a kernel optimized for one query-length and cache-layout regime need not perform well in

another. Kernel evaluation must therefore expose shape distributions and not only a single favorable matrix size.

Weight quantization reduces weight storage and potentially memory traffic. AWQ uses activation information to protect salient channels in low-bit weight-only quantization and couples the representation to inference kernels [Lin et al., 2023]. Actual acceleration depends on hardware instructions, packing, dequantization overhead, and whether the workload is weight-bandwidth limited. KV quantization is a distinct intervention: it changes b_{kv} rather than b_w and requires its own accuracy and kernel evaluation.

These methods compose unevenly. Weight compression can expose KV reads as the next bottleneck; paging can permit a larger batch that increases per-step KV traffic; prefix reuse can reduce prefill work while changing the relative load on decode capacity. Speedup factors from independent papers cannot be multiplied to predict a combined system.

Mechanism	Primary target	What it does not establish
FlashAttention	Attention IO and intermediates	Linear-time full attention
PagedAttention	KV allocation and sharing	Compression of every KV element
GQA/MQA	KV heads and cache size	Checkpoint-independent equivalence
Weight quantization	Weight bytes and kernels	Guaranteed accuracy or speed
Speculative decoding	Serial target-model steps	Guaranteed gain at saturation
Chunked prefill	Phase interference	Free prompt processing
Phase disaggregation	Resource coupling	Free KV transfer

Speculative decoding

A draft model proposes γ tokens; the target model verifies them in a parallel pass. For a proposal $z \sim q$, exact speculative sampling accepts with probability

$$a(z) = \min \left(1, \frac{p(z)}{q(z)} \right).$$

On rejection, the algorithm samples from the normalized positive residual $[p - q]_+$; after all proposals are accepted, it can produce one additional target token. These correction steps are necessary to preserve the target distribution [Leviathan et al., 2022]. Distributional equivalence does not require identical sampled strings under the same random seed or numerical implementation.

Under the simplifying assumption that every sequential proposal has the same conditional acceptance probability α , the expected tokens emitted per verification cycle are

$$\mathbb{E}[K] = \sum_{j=0}^{\gamma} \alpha^j = \frac{1 - \alpha^{\gamma+1}}{1 - \alpha},$$

with limit $\gamma + 1$ at $\alpha = 1$. If one ordinary target decode step takes t_T , drafting each proposal takes t_D , and verification takes $t_V(\gamma)$, an idealized speedup is

$$\mathcal{S} \simeq \frac{t_T \mathbb{E}[K]}{\gamma t_D + t_V(\gamma)}.$$

For an illustrative $\gamma = 4$, $\alpha = 0.8$, $t_D = 0.1t_T$, and $t_V = t_T$, this evaluates to approximately $2.40\times$. This is a toy calculation, not a reproduced benchmark. In practice acceptance depends on position and workload, verification grows with proposal count, and draft-model work competes for memory and compute. The primary paper reports acceleration in its evaluated T5-XXL setting [Leviathan et al., 2022]; it does not validate this hypothetical configuration.

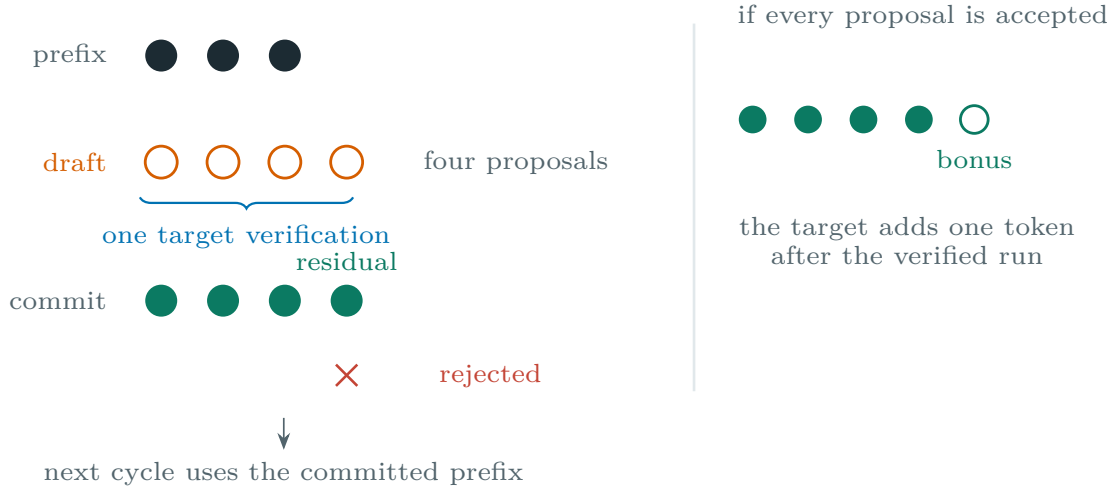


Figure 3: Speculative sampling drawn as token positions. The draft proposes four tokens and one target pass verifies them. Accepted positions are committed, a rejection is replaced by a residual sample, and full acceptance adds a bonus token. The next cycle starts from the committed prefix. Based on Leviathan et al.; not a particular implementation.

Scheduling and distributed serving

Continuous batching and chunked prefill

Iteration-level scheduling admits and retires requests as their sequences progress rather than waiting for an entire fixed batch to finish. PagedAttention’s serving system uses dynamic cache management to support efficient batching [Kwon et al., 2023]. The scheduler must nevertheless decide how much new prompt work to admit alongside ongoing generation.

A long unchunked prefill can delay the next decode step. Chunked prefill divides prompt work into bounded pieces and interleaves it with decodes. Sarathi-Serve uses chunked prefills and stall-free scheduling to address the throughput–latency tradeoff [Agrawal et al., 2024]. The chunk size is a control parameter: reducing it can shorten interference windows while increasing scheduling overhead or reducing matrix efficiency. A token budget alone does not equal a time budget because token costs depend on phase and context length.

A practical scheduler needs both memory admission and latency-aware work admission. Reserving headroom for future KV growth prevents a batch that fits now from immediately exhausting memory

as generation continues. Output-length uncertainty makes this a probabilistic capacity problem rather than an exact one-time allocation decision.

Distributed execution within a phase

Tensor parallelism shards operations across devices but adds collective communication. Pipeline parallelism partitions layers but introduces bubbles and scheduling constraints. Neither makes communication disappear; small token steps can be sensitive to collective latency even when total transferred bytes are modest. Consequently, a single-device roofline estimate is insufficient for a distributed deployment.

Disaggregated inference: the KV-transfer critical path

Separate execution pools, shared state

Prefill–decode (P/D) disaggregation assigns prompt processing and token generation to separate execution pools. This is different from tensor parallelism, which partitions an operation, or pipeline parallelism, which partitions layers. Each phase can itself use either form of parallelism. DistServe uses phase-specific resource allocation and parallelism to improve latency-constrained serving [Zhong et al., 2024]. Mooncake extends the idea toward a KVCache-centric system, combining separated execution pools with a global cache using CPU, DRAM, SSD, and NIC resources [Qin et al., 2025]. Compute disaggregation and cache-storage disaggregation are related but distinct design choices.

The benefit is independent phase provisioning and reduced direct prefill–decode interference. The cost is a state handoff: the selected decode instance must obtain the required KV state in a compatible layout before executing dependent attention. A cache hit on the prefill side does not imply that the decode side already has the state. Consequently, reusing computation can still require substantial network traffic [Yoon and Yu, 2026, Sections II-A–B].

Multi-turn agents change the transfer-to-compute ratio

Yoon and Yu’s *An Optical KV Cache Network for Disaggregated LLM Serving* studies a particularly demanding case: a coding agent appends a small tool result to a long context, while routing selects a decode instance without that context. The old prefix is available at prefill, but the new decode destination needs both the old prefix and newly computed KV. In their selected TraceLab turns fitting a 128K-token window, they report a median context of 66.7K tokens and median new input of 1.1K tokens [Yoon and Yu, 2026, Section II-B]. These are separate reported medians, not a measured median per-turn ratio.

Let S_p be cached prefix tokens, S_n new tokens, m KV bytes per token for the state being transferred, and χ indicate that the destination lacks the entire prefix. A simple payload accounting model is

$$M_{\text{move}} = m(S_n + \chi S_p), \quad \chi \in \{0, 1\}.$$

This assumes new-token KV must be transferred in either case and excludes metadata, replication, compression, and partial-prefix hits. With destination-miss probability p_{miss} , fixed lengths give $\mathbb{E}[M_{\text{move}}] = m(S_n + p_{\text{miss}} S_p)$. A heterogeneous workload requires averaging the joint distribution rather than multiplying independent mean lengths and miss rates. The equation isolates a routing

lever: reducing destination misses can remove more traffic than increasing link speed. Affinity can also create load imbalance, so locality and queueing must be evaluated together.

Overlap is a release-time problem

A transfer’s serialization lower bound remains M/β_{net} , but that is not necessarily its exposed contribution to TTFT. Let prefill start at zero and finish at T_p . A single, uncontended circuit starts reconfiguration at time $s \geq 0$, takes τ to configure, and carries M bytes that are all ready at time $r \geq 0$. Under constant effective rate R , no competing transfers, and no additional copies, its finish time is

$$T_{\text{finish}} = \max(r, s + \tau) + \frac{M}{R}, \quad \Delta_{KV} = \max(0, T_{\text{finish}} - T_p).$$

Here Δ_{KV} is only the exposed KV wait after prefill, not total TTFT. Queueing, scheduling, and first-token computation contribute additional terms. This definition assumes first-token emission waits for the handoff; a system that emits the first token from prefill can instead expose the delay in subsequent token delivery. Metric placement must be documented rather than assumed universal.

For an already resident prefix with $r = s = 0$, the ideal exposed wait is $[\tau + M/R - T_p]_+$. For a bulk pull initiated only after prefill, $s = r = T_p$, the entire $\tau + M/R$ is exposed. As a deterministic illustration, a 1 GiB prefix at an assumed 25 GB/s takes 42.95 ms to serialize. With 50 ms reconfiguration and 176 ms prefill, an early uncontended reservation can hide this prefix’s 92.95 ms setup-plus-transfer, whereas starting it after prefill cannot. These numbers are a constructed example, not a reproduction of the attached paper’s H100 measurements. New-token KV remains a separate dependency; hiding the prefix does not establish that the whole request handoff is hidden.

Layer-wise streaming has multiple release times. For ordered chunks with sizes M_j released at r_j , one idealized fixed-circuit recurrence is

$$f_0 = s + \tau, \quad f_j = \max(r_j, f_{j-1}) + \frac{M_j}{R}.$$

The final exposed wait is $[f_J - T_p]_+$. This recurrence explicitly models the fact that future-layer KV cannot be sent before computation produces it. It assumes an available destination and a fixed circuit; port conflicts, topology changes, and reshaping require additional scheduling constraints. Early setup can hide reconfiguration without hiding all serialization, especially when much of the payload becomes ready near prefill completion.

Optical circuits scheduled during prefill

The supplied paper proposes replacing the **GPU-side KV packet-switch path**, not the entire datacenter network, with optical circuit switching (OCS). Requests, output tokens, and control retain the host-NIC electrical packet network. Each GPU’s RDMA NIC connects through wavelength-division multiplexed (WDM) optics to the OCS; the modeled 400G link carries 45 GB/s at an assumed 90% efficiency [Yoon and Yu, 2026, Section III-A]. The paper distinguishes this assumed effective rate from a separately reported 32.5 GB/s NCCL measurement over 400G InfiniBand [Yoon and Yu, 2026, Section IV-B].

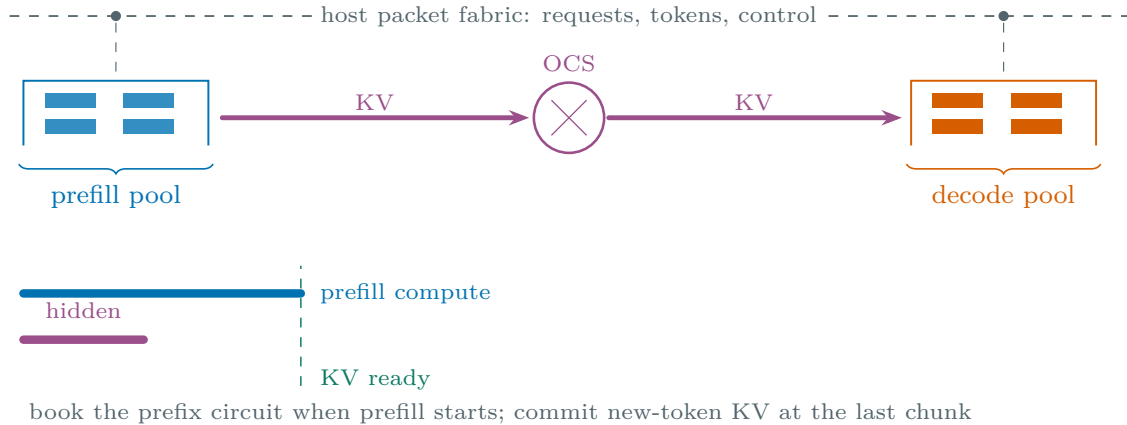


Figure 4: KV migration between prefill and decode pools. Host traffic stays on the packet fabric. Migrated KV crosses an optical circuit switch on a circuit booked while prefill is still running, so setup can finish before the cache is ready. Simplified from Yoon and Yu; each pool may contain more GPUs than shown. Not a claim that the packet fabric disappears.

Mechanical circuit setup is not assumed instantaneous. The scheduler books the cached-prefix **head** at prefill start, chooses a destination whose port becomes free early while preferring an existing circuit, and commits the new-token **tail** when the last prefill chunk starts. Algorithm 1 splits head and tail only when its estimated transfer-plus-setup time satisfies the paper’s admission condition; it is not an unconditional split. Late completion can extend a reservation into free port time or require rebooking. Last-chunk commitment reduces exposure to prediction errors accumulated earlier in the queue [Yoon and Yu, 2026, Algorithm 1 and Section III-B].

Optical packaging matters to the proposed economics. A fiber OCS switches fibers, not individual wavelengths. WDM can carry a 400G link on one fiber pair, while parallel-fiber DR4 consumes more OCS ports. The paper’s one-stage construction uses eight 384-port planes, one per GPU position in eight-GPU nodes, reaching 3,072 GPUs with transfers between corresponding positions [Yoon and Yu, 2026, Section III-A]. This topology constrains placement; it is not arbitrary non-blocking connectivity between every GPU position. The authors also identify optical loss-budget requirements: ordinary interface compliance alone does not guarantee their desired margin through the stated connectors and OCS path [Yoon and Yu, 2026, Table I and Section IV-C].

What the reported evidence establishes

The strongest reading of this work is a network-and-scheduler feasibility argument, not a complete production-inference benchmark. Its evidence has distinct layers:

Evidence	Paper reports	Scope of inference
Prefill calibration	33 Llama-3.1-70B runs on four H100 GPUs; 4.6% median model error	Calibration of simulated prefill timing, not a measured optical cluster

Evidence	Paper reports	Scope of inference
Controlled overlap test	Injected delays on two H100s with Llama-3.1-8B over NVLink	Tests timing overlap; does not implement the proposed optical fabric
Flow-level simulation	128/256-GPU slices; five paired 330-second trace windows	KV fabric and TTFT; decode instances are sinks without decode compute
Cost model	WDM OCS at 0.55–0.78 times a 2:1 electrical packet fabric over the stated 256–3,072-GPU range	Modeled KV-network cost under published-price and port-count assumptions, not total cluster cost

All entries are attributed results, not experiments repeated here [Yoon and Yu, 2026, Sections IV-A–C]. The simulation uses tenant-specific TTFT SLOs at five times median unloaded prefill time. This is not the joint TTFT/TPOT goodput objective defined later in this paper, and it cannot validate decode ITL, full service throughput, or overall energy savings.

The packet baseline is decisive. At 128 GPUs the authors report OCS within 0.4 percentage points of the agentic SLO attainment of non-blocking packet switching with shortest-remaining-processing-time scheduling across their evaluated grid and modes. Adaptive routing with fair sharing remains within 0.5 points of OCS even at 3:1 oversubscription, whereas gains over ECMP are larger [Yoon and Yu, 2026, Section IV-B]. Thus the result is not “optics always beats packets”: routing collisions, scheduling, and modeled price jointly explain the comparison.

Likewise, tolerance of slow switching is conditional. Without circuit reuse, the paper reports all tenants’ P99 TTFT and SLO attainment within 5% of the non-blocking SRPT reference up to 50 ms reconfiguration at 60% prefill load. Reuse permits longer delays in a selected simulated configuration; neither result is a universal 50–100 ms latency allowance [Yoon and Yu, 2026, Section IV-C].

Deployment implications

Disaggregation moves a bottleneck rather than removing it. Evaluate cache-aware decode placement against load-only placement, measure old-prefix and new-token traffic separately, and expose port-wait, setup, serialization, and tail-release delays. Compare adaptive routing and transfer scheduling before attributing an ECMP baseline’s loss to packet switching itself. Provisioning decisions should include both execution pools, any cache-storage tier, and the network boundary—not only the cheaper component.

The optical study also sharpens what a follow-up experiment must test: real OCS setup under concurrent traffic, prediction errors and tenant bursts, transceiver margin, failure recovery, cache-layout compatibility, and end-to-end decode compute with joint TTFT and ITL constraints. Until those are measured, its simulation supports a promising design point under declared assumptions rather than a production-wide cost or performance guarantee.

Service objectives and a reproducible evaluation protocol

Let a_i be request arrival time, f_i first-token time, c_i completion time, and O_i the number of emitted tokens. Define

$$\text{TTFT}_i = f_i - a_i, \quad \text{TPOT}_i = \frac{c_i - f_i}{O_i - 1} \quad (O_i > 1).$$

TPOT is a request-level average; it hides individual stalls. Report the distribution of inter-token latency (ITL) as well. For observation interval H , a request-goodput definition is

$$G = \frac{1}{H} \sum_{i \in \mathcal{C}} \mathbf{1}[\text{TTFT}_i \leq \tau_f \wedge \text{TPOT}_i \leq \tau_p],$$

where $\mathcal{C}$ contains requests completed in the interval. Boundary effects require a steady-state window and explicit accounting for rejected, failed, and unfinished requests. The thresholds τ_f and τ_p are application choices. This formulation follows the latency-constrained emphasis of DistServe rather than replacing it with unconstrained token throughput [Zhong et al., 2024].

A deployment study should preregister the following comparisons:

1. **Capacity hypothesis:** KV-head count and context length predict payload scaling. Compare equation-based payload, allocated bytes, and runtime-reserved bytes separately; explain the gap.
2. **Bandwidth hypothesis:** batching amortizes weights until cache traffic, compute, or communication dominates. Sweep batch and context while measuring memory traffic, achieved compute, per-step latency, and ITL tails.
3. **Scheduling hypothesis:** chunked prefill improves decode tails under long-prompt arrivals. Compare matched offered load with fixed and swept chunk sizes, including the cost to TTFT.
4. **Speculation hypothesis:** accepted tokens per verification outweigh drafting and verification overhead. Log proposal lengths, acceptance-by-position, all target/draft times, and output-quality controls.
5. **Disaggregation hypothesis:** reduced interference offsets unhidden KV-transfer cost at matched provisioned cost. Sweep network contention, session affinity, destination-cache misses, prefix/new-token lengths, transfer release times, and circuit setup. Compare ECMP with adaptive routing and transfer scheduling; measure both phases, transfer queues, and joint TTFT/ITL attainment.

The workload must preserve joint prompt/output-length distributions, burstiness, prefix-sharing frequency, and cancellation behavior. Use open-loop arrivals to expose overload rather than a client that silently reduces offered load when the server slows. Record hardware topology, model and tokenizer revision, engine commit, datatype, quantization settings, cache format, scheduler configuration, and parallelism degrees.

For repeated runs, report the number of runs, random seeds, uncertainty estimates, and the exact unit of resampling. Requests sharing a burst or cache state are not independent; block resampling or run-level analysis is preferable to naive token-level confidence intervals. Do not attach invented error bars to the deterministic calculations in this paper. Quality evaluation is mandatory when quantization or architectural modification changes model outputs; distribution-preserving speculation instead requires implementation correctness tests under the intended sampling settings.

Discussion: a bottleneck-first decision procedure

Start with workload and service objectives. If the model cannot remain resident, resolve weight, cache, and workspace capacity before kernel tuning. If small-batch decode streams weights, test weight compression or batching, then verify that their quality and latency costs remain acceptable. If growing contexts dominate cache traffic, examine KV precision, attention architecture, reuse, and context policy. If long prefills disrupt token delivery, compare chunking against phase isolation before deploying an additional network boundary.

This procedure is a diagnostic order, not a universal product prescription. It asks which resource changed after an intervention. A successful optimization frequently makes its own mechanism less important: reduced weight traffic exposes KV reads; more efficient allocation increases the feasible batch; a larger batch shifts pressure toward compute or communication. Evaluation should therefore measure the next bottleneck, not just confirm that the original one shrank.

The literature supports this separation of concerns. FlashAttention and FlashInfer optimize attention execution at different levels of specialization [Dao et al., 2022, Ye et al., 2025]. PagedAttention changes cache management [Kwon et al., 2023], while GQA changes cache representation through the model architecture [Ainslie et al., 2023]. AWQ changes weight representation and its kernels [Lin et al., 2023]. Speculation changes how many output positions a target verification pass can commit [Leviathan et al., 2022]. Sarathi-Serve and DistServe address scheduling and resource coupling through different architectural choices [Agrawal et al., 2024, Zhong et al., 2024]. This taxonomy is more transferable than a leaderboard compiled from incomparable experimental settings.

Limitations and broader impact

The analytical models omit many implementation details: attention rereads, cache residency in lower memory tiers, quantization scales, vocabulary projections, dynamic frequencies, heterogeneous sequence lengths, collective algorithms, and failure recovery. The cache formula is not applicable without modification to latent attention, recurrent-state models, or mixtures of local and global attention. The hypothetical parameters are not a measured model configuration, and no cost or energy savings have been established.

The literature set was selected by mechanism rather than a preregistered systematic-review protocol. Its absence of a technique is not evidence of that technique’s inferiority. Metadata agreement between arXiv and DataCite verifies bibliographic identity, not independent replication of a paper’s experiments. Public speedup claims remain conditional on the original implementations and workloads.

Efficient inference can broaden access and reduce resource use per request, but lower unit cost can also increase total consumption and facilitate abusive automation. Shared prefixes and retained KV state create privacy and tenant-isolation concerns. Capacity efficiency is not a reason to weaken authentication, cache isolation, retention limits, or abuse monitoring. AI assistance was used in drafting, literature synthesis, calculations, and document assembly; the author should review all claims and derivations before academic submission. It has not undergone peer review.

Conclusion

LLM inference is best understood as a changing balance among compute, memory capacity, memory traffic, scheduling, and communication. The equations and examples show why cache allocation, head sharing, low-bit weights, speculative sampling, and phase isolation solve different problems. A fourfold reduction in logical cache payload is an arithmetic fact under fixed assumptions; a fourfold reduction in latency is an experimental claim that requires separate evidence. Future validation should measure modeled and observed traffic together, study real arrival traces, and compare systems at matched cost and service-level attainment.

Reproducibility appendix

The companion bundle includes the Markdown master, generated LaTeX, DOI-derived bibliography, source metadata, fetched evidence text, analysis script, numerical CSV, vector figures, editable diagrams.net XML, and build instructions. `analysis.py` regenerates the table and quantitative figure from a common parameter set. The PDF is compiled with LaTeX; the HTML copy uses native MathML so formulas render without a network dependency. The evidence files are for traceability and do not constitute independent experimental replication.

Assumptions for the worked examples

Quantity	Value	Status
Layers / query heads / head dimension	32 / 32 / 128	Hypothetical architecture
Batch / context	16 / 8,192 tokens	Illustrative workload
KV bytes per element	2	FP16 payload assumption
Weight parameters / bytes per parameter	8 billion / 2	Independent dense-weight assumption
Effective device / network bandwidth	2 TB/s / 25 GB/s	Assumed, not measured
Speculative proposals / acceptance	4 / 0.8	Constant-acceptance toy model
Draft-token / verification time	0.1 / 1 target-step units	Assumed, not measured

Derivation of speculative expected progress

Let A be the number of consecutive accepted proposals before rejection, capped at γ . The cycle emits $K = A + 1$ tokens, counting either a correction token or the bonus token. The tail-sum identity gives $\mathbb{E}[K] = \sum_{j=0}^{\gamma} \Pr(A \geq j)$. Under constant conditional acceptance, $\Pr(A \geq j) = \alpha^j$, giving the geometric expression above. Real acceptance rates should be measured conditionally by proposal position rather than substituted from an unconditional average without checking this assumption.

References

Amey Agrawal, Nitin Kedia, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav S. Gulavani, Alexey Tumanov, and Ramachandran Ramjee. Taming throughput-latency tradeoff in llm inference with sarathi-serve, 2024. URL <https://arxiv.org/abs/2403.02310>.

- Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. Gqa: Training generalized multi-query transformer models from multi-head checkpoints, 2023. URL <https://arxiv.org/abs/2305.13245>.
- Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness, 2022. URL <https://arxiv.org/abs/2205.14135>.
- Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention, 2023. URL <https://arxiv.org/abs/2309.06180>.
- Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding, 2022. URL <https://arxiv.org/abs/2211.17192>.
- Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. Awq: Activation-aware weight quantization for llm compression and acceleration, 2023. URL <https://arxiv.org/abs/2306.00978>.
- Ruoyu Qin, Zheming Li, Weiran He, Jiale Cui, Feng Ren, Mingxing Zhang, Yongwei Wu, Weimin Zheng, and Xinran Xu. 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)*, pages 155–170, Santa Clara, CA, February 2025. USENIX Association. ISBN 978-1-939133-45-8. URL <https://www.usenix.org/conference/fast25/presentation/qin>.
- Zihao Ye, Lequn Chen, Ruihang Lai, Wuwei Lin, Yineng Zhang, Stephanie Wang, Tianqi Chen, Baris Kasikci, Vinod Grover, Arvind Krishnamurthy, and Luis Ceze. Flashinfer: Efficient and customizable attention engine for llm inference serving, 2025. URL <https://arxiv.org/abs/2501.01005>.
- Seongwon Yoon and Shimeng Yu. An optical kv cache network for disaggregated llm serving, 2026. URL <https://arxiv.org/abs/2610.18444v1>. Version 1, 13 October 2026; user-supplied PDF.
- Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. Distserve: Disaggregating prefill and decoding for goodput-optimized large language model serving, 2024. URL <https://arxiv.org/abs/2401.09670>.