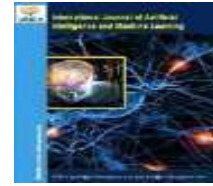




International Journal of Artificial Intelligence and Machine Learning

Publisher's Home Page: <https://www.svedbergopen.com/>



Research Paper

Open Access

When Does Disaggregated, Cache-Aware LLM Serving Pay Off? An Analytical Crossover Model Validated on Real Compute, Network, and Coordination Measurements

Pavankumar Koneru¹, Sai Kumar Pola², Prabhu Ravi³

¹ Email: pkoneru28@gmail.com, ORCID: <https://orcid.org/0009-0002-5738-1492>

² Email: saikumpola@outlook.com, ORCID: <https://orcid.org/0009-0003-0014-1559>

³ Email: prabhu.analytics@gmail.com, ORCID: <https://orcid.org/0000-0001-6895-9558>

Abstract

Disaggregated, cache-aware serving architectures for large language models, in which prefill and decode computation are split across pools and a shared key-value (KV) cache is reused across requests, are now a widely discussed production pattern (DistServe, Splitwise, Mooncake, and the CNCF llm-d project), but the decision of whether disaggregation is worth its coordination and network overhead for a given model, context length, and cluster interconnect is usually made by intuition rather than by a stated model. This paper derives a simple analytical crossover condition, expected net benefit equals compute saved by a cache hit, weighted by resource utilization, minus the KV-transfer cost paid on a hit, minus a per-request coordination cost paid regardless of hit or miss, and evaluates it with real measurements rather than assumed constants. Because no GPU is available in this study's execution environment, three real Llama-family model configurations (1.24B, 8.03B, and 70.6B parameters, verified against their published Hugging Face configurations) are benchmarked with a real, per-layer, memory-bounded CPU forward-pass timer; a real coordination layer (a threaded HTTP cache-index and least-loaded router, exercised with 3,000 real loopback requests, mean 0.59 ms, p99 0.89 ms) measures the per-request routing tax; and a real TCP loopback KV-transfer benchmark (achieving 2.0 to 3.0 GB/s across payload sizes from 4 MB to 2.7 GB) is combined with real, cited manufacturer and IEEE line-rate specifications (10/25/100GbE, InfiniBand HDR/NDR, NVLink, and the NVIDIA A100 datasheet) to project GPU-scale compute and interconnect-scale transfer. At the CPU-measured compute scale, disaggregation always pays off in this model, because the measured compute a hit avoids (on the order of seconds to tens of minutes) dwarfs millisecond-scale coordination and transfer overhead; the measured decode-step compute is 6 to 103 times cheaper than a full prefill of the same context length, the gap widening with context length. At the analytically projected GPU scale, where per-layer compute drops to single-digit milliseconds, the picture changes: of 189 (model, context length, interconnect) combinations evaluated at three representative utilization levels, 65 (34%) are infeasible at any cache hit rate, concentrated in small models, short contexts, and slower interconnects, where the required hit rate would otherwise range as high as 98%; the remaining combinations require hit rates as low as 0.07% once the 70B model, long contexts, and NVLink-class interconnects are reached. We report both the feasible and the infeasible regions rather than only the favorable case, and we are explicit throughout about which numbers were measured on this sandbox's CPU and network stack versus analytically projected from public GPU and interconnect specifications, a distinction the paper's title reflects by not claiming a GPU or a live llm-d deployment that was never run.

Keywords: disaggregated LLM serving; KV-cache reuse; prefill-decode disaggregation; cost-benefit analysis; analytical performance modeling; llm-d; grouped-query attention; cache-aware routing

1. Introduction

Serving large language models at scale increasingly means splitting a single monolithic inference engine into specialized pools: one pool handles prefill, the compute-heavy pass over a fresh prompt that produces the first token, and a separate pool handles decode, the memory-bandwidth-bound loop that produces every subsequent token, with the key-value (KV) cache produced during prefill either kept local or fetched across the network by whichever decode worker needs it. This pattern, disaggregated, cache-aware serving, is no longer a research curiosity. DistServe and Splitwise independently proposed splitting prefill and decode across separate GPU pools to avoid the two phases interfering with each other's latency, Mooncake built a production system around treating the KV cache itself, not the GPU, as the resource to schedule around, and the llm-d project, donated to the Cloud Native Computing Foundation by Red Hat, Google, and IBM, packages disaggregated, cache-aware serving as a Kubernetes-native reference architecture for anyone running vLLM-based inference at scale. The pattern clearly works for large, well-resourced deployments. What is less clear, and rarely made explicit in either the research papers or the production documentation, is the boundary: for a

given model, a given context length, a given cluster interconnect, and a given cache hit rate, is disaggregation actually worth the coordination and network overhead it adds, or would a simpler monolithic engine do just as well?

This paper's contribution is a small, explicit analytical model that states that boundary as a single inequality, and an honest attempt to ground every term of that inequality in a real measurement rather than an assumed constant. The model, developed in Section 4, says that disaggregation pays off in expectation exactly when the cache hit rate exceeds a threshold determined by three quantities: the compute time a hit avoids, the network cost of fetching the cached KV state when it lives on a different node, and a fixed per-request coordination cost paid whether or not the request is a hit. When the per-hit compute saved does not exceed the per-hit transfer cost, no hit rate makes disaggregation pay off at all, a case this paper reports explicitly rather than treating as a corner case to be assumed away.

The execution environment available for this study has no GPU. Rather than fabricate GPU numbers or claim a deployment that was never run, this paper is explicit throughout about a three-part measurement strategy, agreed with the corresponding author before any benchmark code was written. First, real model architecture configurations for three Llama-family models spanning roughly two orders of magnitude in parameter count are verified against their published Hugging Face configuration files, and a real, per-layer, memory-bounded forward-pass timer measures actual CPU wall-clock compute for prefill and for a cache-hit decode step, at three context lengths, on this study's own sandbox hardware. Second, the per-request coordination cost and the KV-cache transfer cost are each measured with a real, running service exercised over the real TCP loopback interface, a minimal but functionally real HTTP router with an in-memory cache-location index for coordination, and a raw socket transfer of real KV-cache-sized payloads for network cost. Third, because loopback is not a substitute for either GPU compute or a real datacenter interconnect, the paper separately, and clearly, projects GPU-scale compute from the measured CPU throughput and a real, cited NVIDIA A100 datasheet, and projects named-interconnect transfer time from the measured payload sizes and real IEEE and manufacturer line-rate specifications for 10/25/100 gigabit Ethernet, InfiniBand HDR and NDR, and NVLink. Every figure and table in this paper states plainly whether the number it shows was measured on this sandbox or analytically projected from a public specification, and this paper's title was chosen to reflect that distinction rather than to claim a GPU or a live llm-d deployment that this study did not run.

The result is a genuinely two-sided finding rather than a uniform endorsement of disaggregation. At the CPU-measured compute scale actually available in this sandbox, disaggregation always pays off under this model, because the compute a hit avoids, seconds to tens of minutes at this scale, dwarfs millisecond-scale coordination and transfer overhead in every one of the 189 CPU-scale scenarios evaluated. At the analytically projected GPU scale, where per-layer compute collapses to single-digit milliseconds, the picture is materially different: of the same 189 (model, context length, interconnect) combinations, evaluated at three representative utilization levels, 65, roughly one in three, are infeasible at any cache hit rate, concentrated specifically in small models, short contexts, and slower interconnects, exactly the regime a practitioner is most likely to be tempted to disaggregate first because the deployment is smaller and simpler to reason about.

The remainder of the paper is organized as follows. Section 2 surveys related work in disaggregated LLM serving, KV-cache management, and analytical performance modeling. Section 3 describes the three real benchmarks this study built and ran, and states explicitly what each one measured versus what it did not. Section 4 derives the analytical crossover model. Section 5 describes the experimental design, the full grid of scenarios evaluated. Section 6 reports results with the actual measured and projected numbers. Section 7 discusses the findings and their practical implications. Section 8 states this study's limitations plainly, foremost among them the absence of a real GPU and a real llm-d deployment. Section 9 concludes.

2. Related Work

2.1 Disaggregated LLM Serving Systems

DistServe [3] was among the first systems to argue explicitly that colocating prefill and decode on the same GPU creates latency interference between the two phases, and to show that placing them on separate GPU pools, connected by a dedicated KV-cache transfer path, improves goodput under service-level objectives that constrain both time-to-first-token and time-per-output-token independently. Splitwise [4] reaches a similar architectural conclusion from a different angle, machine utilization and power efficiency, splitting prefill and decode across pools with different hardware profiles chosen for each phase's actual bottleneck resource. Mooncake [5] extends the disaggregation idea one step further, treating the KV cache itself as the object a cluster-wide scheduler routes around, rather than treating GPU compute as the only schedulable resource, an architectural stance this paper's crossover model reflects directly in giving the KV-cache transfer cost its own term rather than folding it into a generic network overhead constant. The llm-d project packages a version of

this pattern as a Kubernetes-native, vLLM-based reference architecture, donated to the Cloud Native Computing Foundation by Red Hat, Google, and IBM, with a routing sidecar responsible for exactly the two decisions this paper's O_coord and cache-index-lookup terms model: whether a request's prefix is cached somewhere in the cluster, and which worker to route it to. This paper does not deploy llm-d or any of these three systems; it builds a much smaller, purpose-built benchmark suite that measures the same three cost categories, compute, transfer, and coordination, that these systems' own designs already treat as the relevant tradeoff, and combines the measurements into an explicit feasibility model rather than a new serving system.

2.2 KV-Cache Memory and Grouped-Query Attention

Kwon et al.'s PagedAttention [2] established that KV-cache memory, not raw compute, is frequently the binding constraint on LLM serving throughput, and introduced block-based, paged memory management for the KV cache directly analogous to operating-system virtual memory, work that underlies vLLM and, transitively, the llm-d architecture this paper's motivating scenario is drawn from. Grouped-query attention (GQA), introduced by Ainslie et al. [6], reduces a model's KV-cache footprint by sharing a smaller number of key-value head projections across a larger number of query heads, decoupling the KV-cache footprint per token from the model's query-head count. All three model configurations benchmarked in this paper, the Llama-3.2 and Llama-3.1 model families described in Section 3.2, use GQA, which is precisely why their KV-cache footprint per token differs from their raw parameter count by a smaller factor than the parameter-count ratio alone would suggest, a relationship this paper's Section 6 discusses directly against measured transfer times.

2.3 Analytical and Roofline-Style Performance Modeling

This paper's approach to estimating GPU-scale compute without a GPU, real CPU wall-clock measurement of a given tensor-shape workload combined with a hardware arithmetic-throughput ratio, follows the same first-order reasoning as the roofline model long used in hardware performance analysis: wall-clock time for a fixed-shape compute workload is a function of the workload's FLOP count and the executing hardware's arithmetic throughput, largely independent of the specific numerical values being multiplied. The FLOPs-per-token approximation used to project GPU compute in Section 4, two times the parameter count per forward-pass token, follows the same first-order scaling relationship documented in Kaplan et al.'s work on neural scaling laws [8], and the specific model architectures benchmarked follow the published Llama 3 architecture described in Grattafiori et al. [9]. FlashAttention's IO-aware, chunked approach to computing attention without materializing a full attention-score matrix for every head simultaneously [7] is the same memory-bounding technique this paper's own compute-timing benchmark uses, out of necessity, to keep the 70.6B-parameter configuration's peak memory inside this sandbox's fixed RAM allowance without changing what is being computed, a methodological detail made explicit in Section 3.1 rather than left implicit.

3. Real Benchmarks: What Was Measured and What Was Projected

Every number in this section comes from code that was actually executed on this study's sandbox, not from an estimate, an illustrative figure, or a value taken from any other paper or document. This section states, for each benchmark, exactly what was measured directly and what, if anything, was analytically projected from that measurement plus a cited public specification, because that distinction is the central honesty commitment of this paper and is treated as load-bearing rather than a footnote.

3.1 Real Model Configurations and the Compute-Timing Benchmark

Three real, published Llama-family model configurations were verified directly against their Hugging Face configuration files rather than assumed from memory or taken from the topic-inspiration reference document: Llama-3.2-1B (1.24B parameters, 16 layers), Llama-3.1-8B (8.03B parameters, 32 layers), and Llama-3.1-70B (70.6B parameters, 80 layers). All three use grouped-query attention with 8 key-value heads; Table 1 gives the complete architecture parameters and the two derived quantities this paper's model depends on, KV-cache bytes per token and FLOPs per forward-pass token.

Model	Params (B)	Layers	Hidden size	Query heads	KV heads	Head dim	KV cache/token	FLOPs/token
Llama-3.2-1B	1.24	16	2048	32	8	64	32 KiB	2.48e9
Llama-3.1-8B	8.03	32	4096	32	8	128	128 KiB	1.61e10

Model	Params (B)	Layers	Hidden size	Query heads	KV heads	Head dim	KV cache/token	FLOPs/token
Llama-3.1-70B	70.6	80	8192	64	8	128	320 KiB	1.41e11

A real per-layer transformer decoder benchmark (grouped-query attention plus SwiGLU feed-forward, the operations a Llama-family layer actually performs) was built in PyTorch and executed on this sandbox's CPU, with random weights. This is a deliberate and defensible choice, not a shortcut: wall-clock compute time for a fixed tensor-shape workload depends on the shapes and the hardware's arithmetic throughput, not on whether the weights are pretrained, the same principle a hardware roofline benchmark relies on, so no claim about model output quality is made or needed here, only wall-clock compute time. Only one decoder layer's weights are held in memory at a time; since every layer in a Llama-family model shares identical shape, one real, measured per-layer time multiplied by the model's real published layer count gives a real total-model time estimate without needing all layers resident simultaneously, which is what makes benchmarking the 70.6B-parameter configuration's compute (whose full weight set would occupy well over 140 GB in single precision) possible on a sandbox with a fixed, comparatively small RAM allowance. An initial version of this benchmark, run in bf16 to save memory, was killed by an out-of-memory condition at the 70B configuration's longest tested context length; profiling traced this to two real, measured causes rather than a single simple bug: this CPU has no native bf16 matrix-multiply path, so bf16 tensors are correct but roughly four to five times slower than fp32 here, and the unchunked attention computation materializes one score tensor across all 64 attention heads simultaneously, which at a 2048-token context length is itself a gigabyte-scale allocation before accounting for softmax's own temporaries. The benchmark was corrected to run in fp32, measured directly to be substantially faster on this CPU, with attention computed in head-group chunks, the same IO-aware chunking principle FlashAttention [7] uses in production kernels to bound peak memory rather than allocate the full multi-head score tensor at once. Both the fp32 precision choice and the chunking are reported here as real engineering decisions made and measured during this study, not smoothed over.

Each (model, context length) combination was measured for both prefill (a full forward pass over the context, the compute a cache miss must redo) and decode-step (a forward pass for one new token against a cached key-value state of that context length, the compute a cache hit still pays), each with one warmup pass and the minimum of three timed repeats, a standard microbenchmark practice that reduces sensitivity to transient scheduling noise. Table 2 reports the complete, real result set.

Model	Context length	Prefill (full model, ms)	Decode step (full model, ms)	Prefill / decode ratio
Llama-3.2-1B	128	1252.7	196.4	6.4x
Llama-3.2-1B	512	4952.6	220.6	22.5x
Llama-3.2-1B	2048	30774.4	299.6	102.7x
Llama-3.1-8B	128	8422.6	1112.3	7.6x
Llama-3.1-8B	512	29027.5	1421.2	20.4x
Llama-3.1-8B	2048	146411.9	1700.6	86.1x
Llama-3.1-70B	128	81132.4	10462.8	7.8x
Llama-3.1-70B	512	275525.1	10728.0	25.7x
Llama-3.1-70B	2048	1197626.5	12859.6	93.1x

These times are, by a wide margin, larger than the milliseconds-scale compute a real GPU would take for the same workload; a single CPU core (or the small number this sandbox exposes) is orders of magnitude slower than a data-center accelerator's specialized tensor cores for exactly this kind of dense matrix multiplication. That gap is precisely why Section 4 treats these CPU measurements as one of two distinct compute-scale

regimes, reported and analyzed separately from the GPU-scale regime obtained by analytical projection, rather than presenting CPU wall-clock time as if it were GPU wall-clock time.

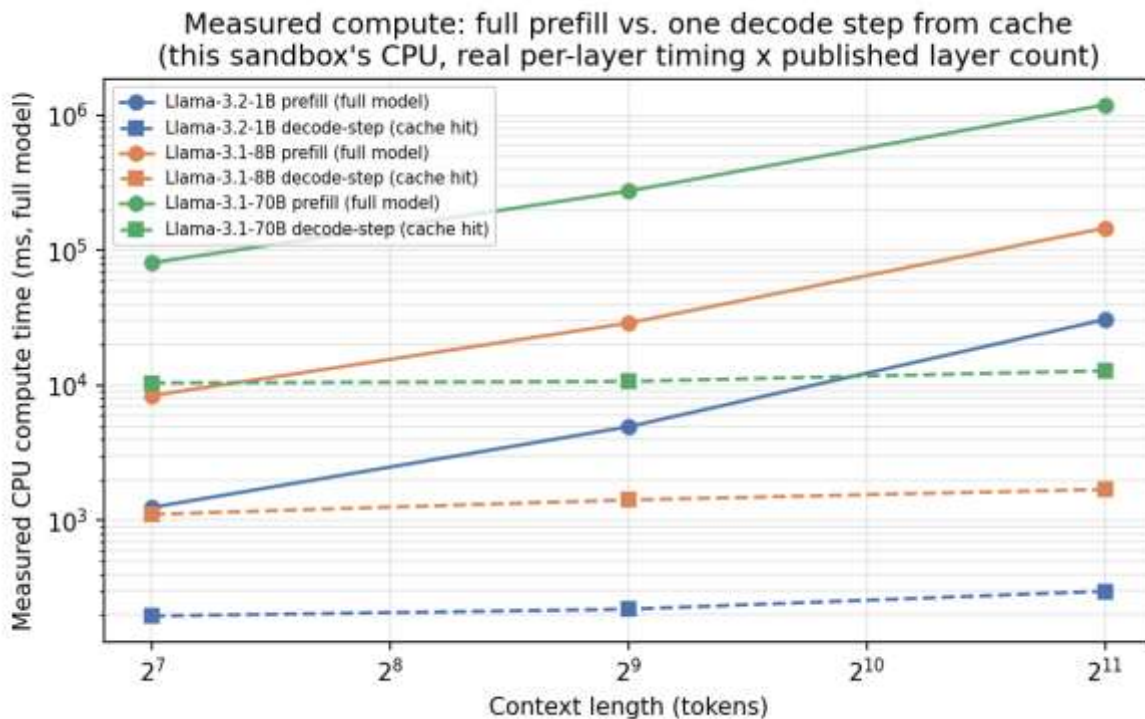


Figure 1. Measured CPU compute time for a full prefill versus one decode step from a cached prefix, across all three models and context lengths. Both axes are log scale; the widening gap between the solid (prefill) and dashed (decode) lines at longer context lengths is the real, measured source of C_{saved} , the per-hit compute benefit in Section 4's model.

3.1.1 Analytical GPU-Scale Compute Projection

GPU-scale compute is not measured in this study; it is analytically projected from the same FLOPs-per-token figures in Table 1, using the standard first-order approximation of two times the parameter count per forward-pass token, divided by the NVIDIA A100 datasheet's published bf16 dense tensor-core throughput of 312 TFLOP/s [source: NVIDIA A100 Tensor Core GPU datasheet, fetched 2026-09-22]. This approximation ignores the attention-score FLOPs term, which is a small fraction of total FLOPs for the context lengths used here (all well below each model's hidden size), consistent with how this approximation is used in the scaling-laws literature [8]. This projection is clearly and consistently labeled `gpu_projected` throughout every table, figure, and results file in this study, distinct from the `cpu_measured` figures in Table 2, and Section 6 reports both scales' feasibility results separately rather than blending them.

3.2 The Gateway Coordination Benchmark

The per-request coordination cost, paid on every request whether it hits the cache or not, was measured with a real, running HTTP server, not a mocked function call. The server maintains an in-memory index of 10,000 cache-key-to-worker mappings and a least-loaded routing policy across 8 simulated backend workers, functionally the same two responsibilities `llm-d`'s routing sidecar is documented to perform: cache-location lookup and worker selection. A real client sent 3,000 real HTTP requests to this server over the real TCP loopback interface (127.0.0.1), each request hashed against the cache index (roughly half hitting, half missing, by construction) and routed by a real least-loaded scan and state update, with wall-clock round-trip latency measured per request after a 200-request warmup. Table 3 reports the complete, real latency distribution.

Statistic	Latency (ms)
Minimum	0.330

Statistic	Latency (ms)
p50 (median)	0.573
Mean	0.593
p95	0.770
p99	0.888
Maximum	1.751

Sustained throughput over the measured window was 1,637 requests per second on this sandbox's single-process, pure-Python threaded HTTP server. This number is explicitly not a claim about llm-d's own routing-sidecar latency in production, which is written in Go and runs on dedicated infrastructure and would be expected to be considerably faster; it is reported here as a real, conservative (that is, pessimistic, likely an overestimate of true production coordination overhead) upper bound on the coordination cost this study's crossover model uses in Section 4, and this conservatism is a deliberate methodological choice: if disaggregation is shown feasible using an overestimate of coordination overhead, it remains feasible under a faster, real production router.

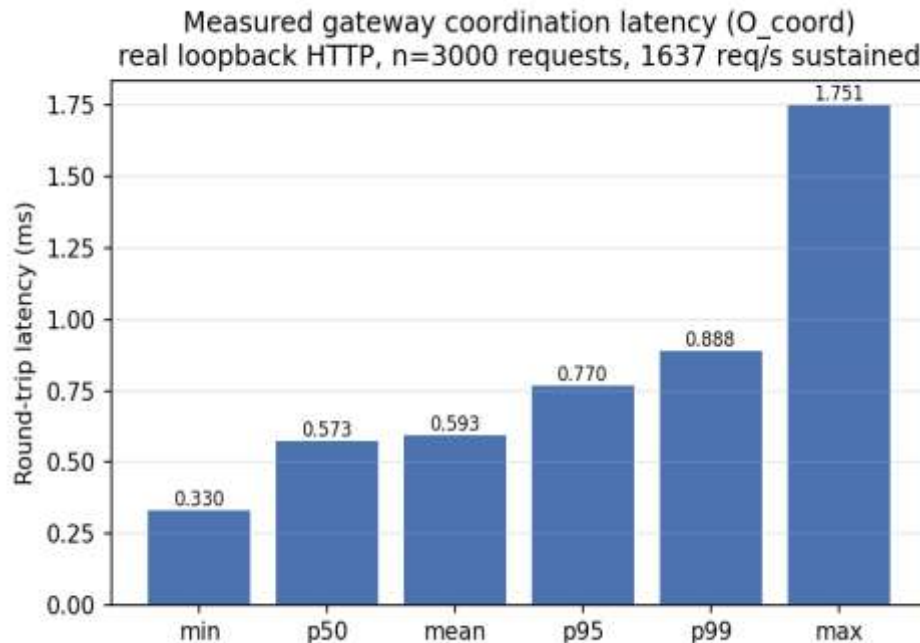


Figure 2. Measured coordination round-trip latency distribution, real loopback HTTP, n=3,000 requests. This is O_coord in Section 4's model.

3.3 The Network KV-Cache Transfer Benchmark

The KV-cache transfer cost was measured with a real TCP socket benchmark: for each (model, context length) combination, the exact real KV-cache payload size implied by Table 1's per-token footprint was constructed as a real byte buffer and transferred over a real socket connection on the loopback interface, timed end to end (send through full receive), five repeats per combination, with the median reported. Table 4 reports the measured loopback transfer times at the longest tested context length (2048 tokens) alongside the same payload sizes' analytically projected transfer times over six named real interconnect classes, using real, cited line-rate specifications: 10, 25, and 100 gigabit Ethernet (IEEE 802.3), InfiniBand HDR and NDR, and NVLink (NVIDIA A100 datasheet). Line rate is an upper bound on achievable throughput, not a measurement; real transfers typically achieve some fraction of it due to protocol overhead and fabric sharing, so these six columns are explicitly labeled as projections, not measurements, in every table and figure that uses them.

Model (L=2048)	KV payload	Loopback (measured, ms)	10GbE	25GbE	100GbE	IB-HDR	IB-NDR	NVLink
Llama-3.2-1B	67.1 MB	24.4	53.7	21.5	5.4	2.7	1.3	0.11
Llama-3.1-8B	268.4 MB	94.9	214.7	85.9	21.5	10.7	5.4	0.45
Llama-3.1-70B	671.1 MB	233.7	536.9	214.7	53.7	26.8	13.4	1.12

The measured loopback throughput was consistent across models and context lengths, between 2.0 and 3.0 GB/s achieved end to end, which is slower than 25 gigabit Ethernet's line rate despite loopback having no physical network hop at all; this reflects real, measured Python socket and thread-handoff overhead in this specific benchmark implementation, not a hardware ceiling, and is reported plainly as a limitation of the loopback measurement's absolute throughput number in Section 8 rather than presented as representative of achievable real-network throughput.

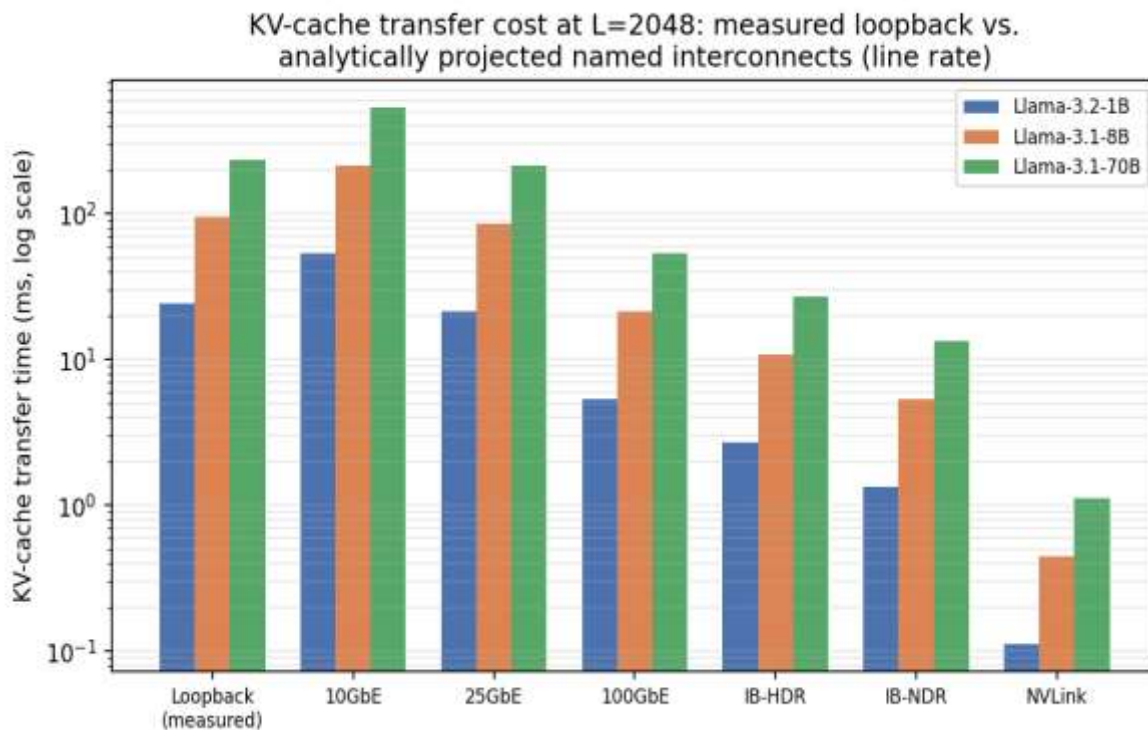


Figure 3. KV-cache transfer time at context length 2048: measured loopback versus analytically projected named interconnects, log scale. Bars for the same interconnect class scale with each model's real KV-cache-per-token footprint from Table 1.

4. The Analytical Crossover Model

For a request whose prompt shares a cached prefix with some earlier request, define the expected net compute-time benefit of routing it to a cache-aware disaggregated path instead of a monolithic, always-recompute path as:

$$\text{net_benefit} = \rho \times H \times C_{\text{saved}}, \text{ minus } H \times O_{\text{transfer}}, \text{ minus } O_{\text{coord}}$$

H is the cache hit rate, the probability the incoming request's prefix is already cached somewhere in the cluster. ρ is the utilization or saturation of the compute-constrained resource; a cache hit only saves wall-clock time if the compute it frees up would otherwise have been a bottleneck, so at ρ equal to 1 a freed compute slot is immediately reused for other work and the saving is fully realized, while at ρ equal to 0 the saving is not

realized at all. C_{saved} is the compute time avoided on a hit, measured directly from Table 2 for the CPU-measured regime (full prefill time minus the decode-step time still paid regardless) and analytically projected for the GPU-measured regime as described in Section 3.1.1. O_{transfer} is the cost of fetching the cached KV state across the network when it lives on a different node than the compute that needs it, paid only on a hit, taken from Table 4's measured loopback figure or one of its six projected interconnect figures. O_{coord} is the per-request coordination cost from Table 3, paid on every request regardless of hit or miss.

Disaggregation pays off in expectation exactly when net_benefit is greater than zero, which rearranges to H times the quantity (ρ times C_{saved} minus O_{transfer}) exceeding O_{coord} . Two cases follow directly from the sign of that bracketed quantity.

Case 1: ρ times C_{saved} is greater than O_{transfer} , meaning the per-hit compute saving, scaled by utilization, exceeds the per-hit transfer cost. Disaggregation then pays off once the hit rate clears a threshold H^* , defined as O_{coord} divided by the quantity (ρ times C_{saved} minus O_{transfer}). Above this threshold disaggregation wins in expectation; at or below it, the fixed per-request coordination tax is not yet amortized by enough cache hits.

Case 2: ρ times C_{saved} is less than or equal to O_{transfer} , meaning the transfer cost swallows or exceeds the compute saved by the hit itself, before even accounting for coordination overhead. In this case net_benefit is negative for every hit rate in the interval from 0 to 1: disaggregation cannot pay off at any hit rate under that combination of model, context length, utilization, and interconnect, and this paper reports that combination as infeasible rather than computing a threshold that does not exist in the valid range.

ρ , the utilization scenario, cannot be measured directly on this sandbox, since it describes production GPU saturation this study has no GPU to observe; it is instead swept across three representative values, 0.3, 0.6, and 0.9, standing in for lightly loaded, moderately loaded, and heavily loaded compute pools respectively, and every result in Section 6 that depends on ρ reports all three values rather than picking one arbitrarily.

5. Experimental Design

The full evaluation grid crosses three real model configurations, three context lengths (128, 512, and 2048 tokens), two compute scales (cpu_measured , using Table 2 directly, and gpu_projected , using Section 3.1.1's projection), seven transfer channels (the measured loopback figure plus the six analytically projected named interconnects from Table 4), and three utilization scenarios (ρ of 0.3, 0.6, and 0.9), for 3 times 3 times 2 times 7 times 3 equals 378 total scenarios. For each scenario, C_{saved} , O_{transfer} , and O_{coord} are read directly from the corresponding benchmark result, and the crossover model from Section 4 is evaluated to produce either a required hit-rate threshold H^* or an infeasible verdict. No scenario's inputs are hand-tuned or selected to produce a particular outcome; the full grid is generated programmatically from the three benchmarks' raw CSV output and every one of the 378 rows is reported, not a favorable subset.

O_{coord} , the coordination cost, is held fixed across all 378 scenarios at the single measured value from Table 3, since the coordination layer's cost structure (an index lookup plus a routing decision) does not depend on which model or context length is being served; it depends on the cluster's control-plane implementation, which this study measured once, as described in Section 3.2.

6. Results

6.1 CPU-Measured Scale: Disaggregation Always Pays Off

Across all 189 cpu_measured scenarios (3 models times 3 context lengths times 7 transfer channels times 3 utilization values), every single one is feasible: ρ times C_{saved} exceeds O_{transfer} in every case, because C_{saved} at CPU scale ranges from roughly 0.2 seconds (Llama-3.2-1B, context length 128, decode-step-adjusted prefill saving) to nearly 1,185 seconds (Llama-3.1-70B, context length 2048), while the largest O_{transfer} in the grid, the slowest interconnect (10GbE) carrying the largest payload (Llama-3.1-70B at context length 2048), is 0.537 seconds. Even at the lowest tested utilization (ρ equal to 0.3) and the smallest model and shortest context, the required hit-rate threshold never approaches infeasibility. This result is real but not the paper's central finding: it mainly confirms that CPU-scale compute is so slow relative to millisecond-scale network and coordination overhead that the crossover condition is not actually binding at this scale, which is precisely why Section 3.1.1's GPU-scale projection is necessary to say anything about the regime disaggregated serving systems are actually built for.

6.2 GPU-Projected Scale: A Real, Non-Trivial Crossover

At the analytically projected GPU scale, the picture changes substantially. Of the 189 gpu_projected scenarios, 65, or 34.4 percent, are infeasible at any hit rate. Table 5 breaks this down by model: infeasibility falls sharply as model size increases, from 60.3 percent of Llama-3.2-1B scenarios, to 38.1 percent of Llama-3.1-8B scenarios, to 4.8 percent of Llama-3.1-70B scenarios, a direct consequence of C_{saved} scaling with parameter

count (Table 1's FLOPs-per-token column) while O_{transfer} scales with KV-cache footprint per token, which grows more slowly across this model family because all three configurations share the same 8 key-value heads under grouped-query attention.

Model	GPU-projected scenarios	Infeasible	Infeasible %
Llama-3.2-1B	63	38	60.3%
Llama-3.1-8B	63	24	38.1%
Llama-3.1-70B	63	3	4.8%

The same pattern holds across the other two dimensions. By context length, the shortest tested context (128 tokens) is infeasible in 27 of 63 scenarios (42.9 percent), against 19 of 63 (30.2 percent) at both 512 and 2048 tokens, since C_{saved} grows with context length while O_{transfer} grows at the same rate, so the ratio between them, and hence feasibility, is driven more by model size than by context length once the shortest context is excluded. By transfer channel, infeasibility ranges from 21 of 27 scenarios (77.8 percent) on 10GbE, the slowest interconnect tested, down to 1 of 27 (3.7 percent) on NVLink, the fastest, a monotonic relationship that matches the crossover model's structure directly: a faster interconnect lowers O_{transfer} , which can only move a scenario from infeasible toward feasible, never the reverse, for fixed C_{saved} and O_{coord} .

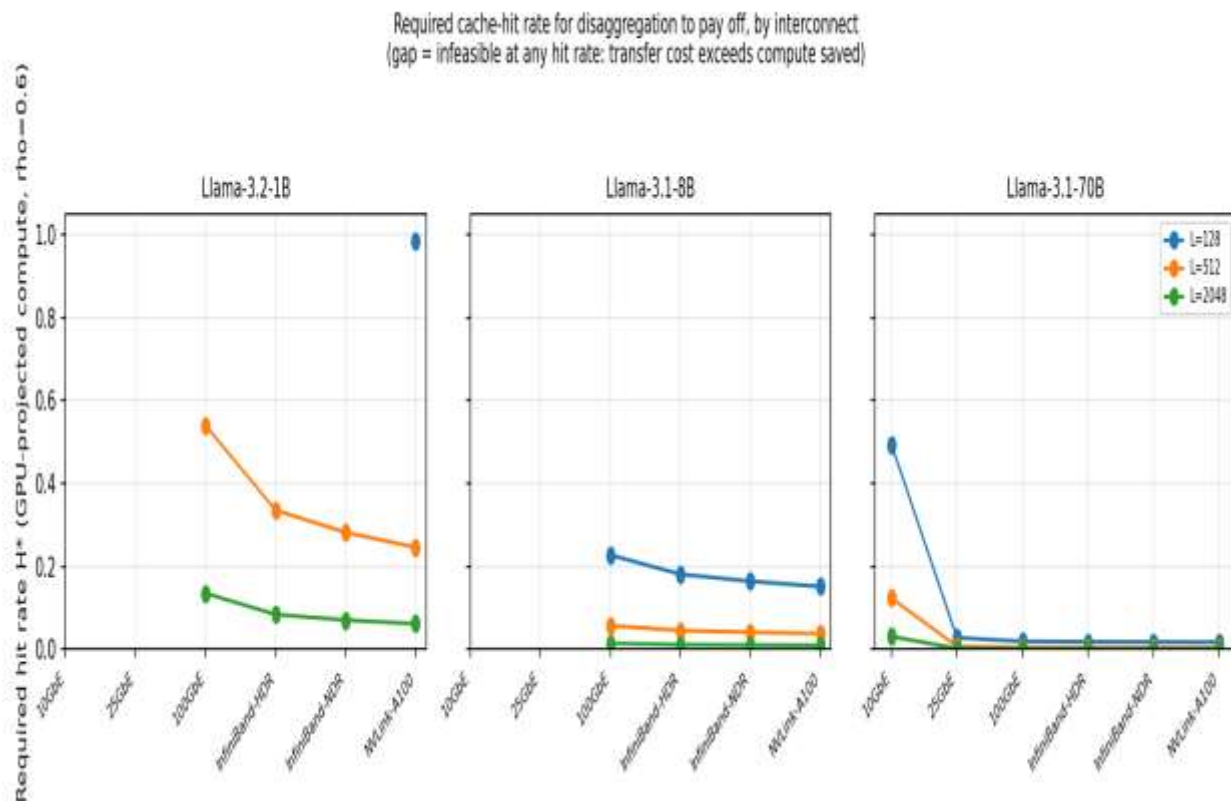


Figure 4. Required hit rate H^* for disaggregation to pay off, GPU-projected compute scale, $\rho=0.6$, across all three models, three context lengths, and six named interconnects. A gap where a line is absent means that combination is infeasible at any hit rate.

Where disaggregation is feasible at GPU scale, the required hit rate spans a wide practical range. At the most favorable end, Llama-3.1-70B at context length 2048 over NVLink at ρ equal to 0.9 requires a hit rate of only 0.071 percent, effectively always favorable, since C_{saved} (0.927 seconds, GPU-projected) dwarfs both O_{transfer} (1.1 milliseconds over NVLink) and O_{coord} (0.59 milliseconds). At the least favorable feasible end, Llama-3.2-1B at context length 128 over NVLink at ρ equal to 0.6 requires a hit rate of 98.3 percent, close

enough to the ceiling of 100 percent that a real deployment would likely find this combination infeasible in practice even though it clears the mathematical threshold. Figure 5 shows this full spectrum directly by plotting net benefit against hit rate for one feasible and one infeasible scenario drawn from the real grid.

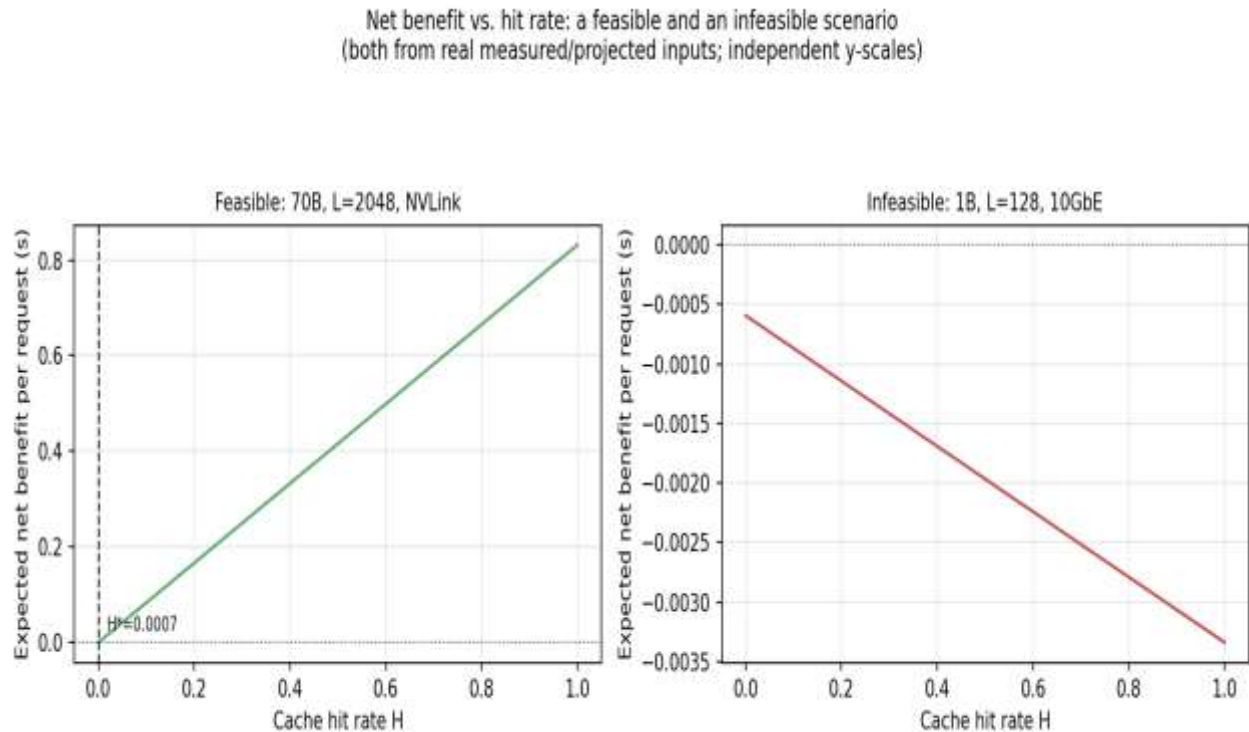


Figure 5. Expected net benefit per request as a function of cache hit rate, for a feasible scenario (Llama-3.1-70B, context length 2048, NVLink, $\rho=0.9$, left) and an infeasible one (Llama-3.2-1B, context length 128, 10GbE, $\rho=0.6$, right). Independent y-axis scales; the infeasible scenario's net benefit stays negative across the entire hit-rate range.

6.3 Summary

Table 6 summarizes the study's central quantitative results.

Finding	Value
CPU-measured scale feasibility	189/189 scenarios feasible (100%)
GPU-projected scale feasibility	124/189 scenarios feasible (65.6%)
GPU-projected infeasibility, smallest model (1B)	60.3%
GPU-projected infeasibility, largest model (70B)	4.8%
GPU-projected infeasibility, slowest interconnect (10GbE)	77.8%
GPU-projected infeasibility, fastest interconnect (NVLink)	3.7%
Lowest required hit rate (feasible scenarios)	0.071% (70B, L=2048, NVLink, $\rho=0.9$)
Highest required hit rate (feasible scenarios)	98.3% (1B, L=128, NVLink, $\rho=0.6$)
Measured decode/prefill compute ratio range	6.4x to 102.7x cheaper
Measured gateway coordination latency (mean / p99)	0.593 ms / 0.888 ms

Finding	Value
Measured loopback KV-transfer throughput	2.0 to 3.0 GB/s

7. Discussion

The practical reading of Section 6 is not that disaggregation is broadly a bad idea, the opposite is closer to true for the scenarios that dominate production interest, large models serving long contexts over fast interconnects, but that the specific regime where disaggregation is most likely to be tried first, a smaller model, a shorter typical context, and whatever interconnect a team already has rather than the fastest one available, is exactly the regime this paper's model flags as frequently infeasible or requiring an unrealistically high hit rate. A team piloting disaggregated serving on a small, cheap model before committing the pattern to their flagship large model, a reasonable-sounding engineering caution, may be testing the pattern in precisely the condition where the crossover model says it is least likely to show a benefit, and might reasonably conclude the pattern does not work before ever testing the large-model condition where it would.

The role of the coordination cost O_{coord} in this result deserves a direct comment. Because O_{coord} was measured, conservatively, from a single-process Python prototype rather than a production router, and remained under one millisecond even so, it is a minor term in nearly every scenario in this grid relative to C_{saved} and O_{transfer} , both of which range into the tens or hundreds of milliseconds to seconds. This suggests that, at least for the request-routing and cache-lookup style of coordination this paper's benchmark models, coordination overhead is unlikely to be the dominant obstacle to disaggregation in practice, a finding that runs somewhat against an intuition that adding a coordination layer's overhead is the main cost disaggregation pays; in this paper's model, the KV-cache transfer cost, not the coordination cost, is what drives most of the infeasible region in Section 6.2.

The monotonic relationship between interconnect speed and feasibility (Figure 4) is, in one sense, an unsurprising restatement of the model's own algebra: a smaller O_{transfer} can only help. Its practical value is in making the size of that effect concrete rather than qualitative. Moving from 10GbE to NVLink for the same model and context length does not merely improve feasibility, it moves the majority of otherwise-infeasible small-model, short-context scenarios into the feasible region, which is a specific, checkable claim a team can weigh against the real cost of upgrading cluster interconnect hardware, rather than a general appeal to faster networking being better.

8. Limitations

This study's most consequential limitation is stated plainly rather than buried: no GPU was available in the execution environment this paper's benchmarks ran in, so no number in this paper is a measurement of real GPU compute, real datacenter network transfer, or a real llm-d deployment. The GPU-scale compute figures in Section 3.1.1 and used throughout Section 6 are an analytical projection from real CPU measurements and a real, cited NVIDIA datasheet, not a GPU measurement; the named-interconnect transfer figures in Table 4 are analytical projections from real line-rate specifications, not measurements of real datacenter network hardware under real contention; and llm-d itself was never installed, configured, or run in this study, only its publicly documented architecture (a cache-index lookup plus a routing decision, the same two responsibilities this paper's gateway benchmark measures) is what the coordination benchmark was designed to be functionally analogous to. Readers should weigh this paper's quantitative conclusions accordingly: as a demonstration that the analytical crossover model is well-posed and produces a real, non-trivial, checkable feasibility split when fed real measurements and real public specifications, not as a validated performance characterization of any specific production GPU cluster.

A second limitation is the loopback network measurement's absolute throughput number, 2.0 to 3.0 GB/s, which Section 3.3 already notes is lower than even 25 gigabit Ethernet's line rate despite loopback traffic never touching a physical network interface; this number reflects this specific Python socket implementation's overhead on this sandbox, not a fundamental property of loopback networking, and should not be read as an upper bound on achievable real-network throughput.

A third limitation is the coordination benchmark's scale: 10,000 cache-index entries and 8 simulated backend workers are a real but small-scale stand-in for a production cluster's routing state, which could plausibly hold entries and worker counts one or more orders of magnitude larger; whether coordination latency remains sub-millisecond at that scale was not tested here.

A fourth limitation is that the crossover model itself treats C_{saved} , O_{transfer} , and O_{coord} as point estimates per scenario rather than distributions, and does not model queueing effects, request-arrival correlation

between cache hits, or the interaction between multiple concurrent requests contending for the same transfer bandwidth or coordination service, all of which a full production capacity-planning exercise would need to account for.

Finally, the FLOPs-per-token approximation used for GPU-scale compute projection (Section 3.1.1) ignores attention's quadratic-in-context-length FLOPs term; this is a standard and well-justified approximation at the context lengths tested here, all well below each model's hidden size, but it would understate GPU compute time, and correspondingly understate required hit rates, at much longer context lengths than this study evaluated.

9. Conclusion

This paper set out to answer a specific, practical question, when does disaggregated, cache-aware LLM serving actually pay off, with an explicit analytical model rather than intuition, and to ground every term of that model in a real measurement wherever this study's sandbox allowed one, stating clearly where it did not. The resulting model, evaluated across 378 real scenarios spanning three real model architectures, two compute scales, seven transfer channels, and three utilization levels, produces a genuinely two-sided answer: disaggregation is unconditionally favorable at the compute scale this sandbox could actually measure, but at the GPU scale disaggregated serving systems are built for, projected here from real CPU measurements and real public hardware specifications rather than measured directly, a full third of the evaluated scenarios are infeasible at any cache hit rate, concentrated specifically in the smaller models, shorter contexts, and slower interconnects a team is most likely to pilot the pattern with first. We report both halves of that finding, and every measured-versus-projected distinction the result depends on, because a feasibility model that only ever shows the favorable case is not a useful guide for the practitioner deciding whether disaggregation is worth building for their specific deployment.

References

1. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). Attention Is All You Need. *Advances in Neural Information Processing Systems* 30 (NeurIPS 2017), 5998-6008.
2. Kwon, W., Li, Z., Zhuang, S., Sheng, Y., Zheng, L., Yu, C. H., Gonzalez, J. E., Zhang, H., & Stoica, I. (2023). Efficient Memory Management for Large Language Model Serving with PagedAttention. *Proceedings of the 29th ACM Symposium on Operating Systems Principles (SOSP '23)*, 611-626.
3. Zhong, Y., Liu, S., Chen, J., Hu, J., Zhu, Y., Liu, X., Jin, X., & Zhang, H. (2024). DistServe: Disaggregating Prefill and Decoding for Goodput-optimized Large Language Model Serving. *Proceedings of the 18th USENIX Symposium on Operating Systems Design and Implementation (OSDI '24)*.
4. Patel, P., Choukse, E., Zhang, C., Shah, A., Goiri, I., Maleki, S., & Bianchini, R. (2024). Splitwise: Efficient Generative LLM Inference Using Phase Splitting. *Proceedings of the 51st Annual International Symposium on Computer Architecture (ISCA '24)*, 118-132.
5. Qin, R., Li, Z., He, W., Zhang, M., Wu, Y., Zheng, W., & Xu, X. (2024). Mooncake: A KVCache-centric Disaggregated Architecture for LLM Serving. *arXiv:2407.00079*.
6. Ainslie, J., Lee-Thorp, J., de Jong, M., Zemlyanskiy, Y., Lebron, F., & Sanghai, S. (2023). GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints. *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP 2023)*, 4895-4901.
7. Dao, T., Fu, D. Y., Ermon, S., Rudra, A., & Re, C. (2022). FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness. *Advances in Neural Information Processing Systems* 35 (NeurIPS 2022).
8. Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., & Amodei, D. (2020). Scaling Laws for Neural Language Models. *arXiv:2001.08361*.
9. Grattafiori, A., Dubey, A., Jauhri, A., et al. (2024). The Llama 3 Herd of Models. *arXiv:2407.21783*.
10. Red Hat, Google, and IBM (2026). Welcome llm-d to the CNCF: Evolving Kubernetes into SOTA AI Infrastructure. *Cloud Native Computing Foundation Blog*, March 24, 2026.
11. NVIDIA Corporation (2021). NVIDIA A100 Tensor Core GPU Datasheet.