

Trace the Request Before You Buy the Next GPU

Sean Fraser

17 October 2026

Abstract

A serving endpoint can have requests waiting while the GPU still has room for useful work. This paper separates that queue into admission, cache pressure, prefill interference, and host overhead, and it treats a GPU purchase as the last step rather than the first. The account uses published serving systems. It is not a new benchmark of my single-node machine. The June 2023 vLLM report found 60 to 80 percent memory waste from fragmentation and over-reservation in the systems it examined, and it reported residual waste under 4 percent after paging, in the last block of a sequence. The PagedAttention paper reports a 2 to 4 times throughput gain at the same level of latency against FasterTransformer and Orca. Those are different baselines and different claims. Neither one is a license to treat every idle gap as a GPU shortage.

Introduction

A long queue and a busy GPU graph are not the same diagnosis. Continuous batching exists because requests finish at different iterations, and a static batch cannot replace a finished request until the whole batch ends. Buying another accelerator before tracing that path can pay for the same scheduling problem twice.

A request is not a packet that disappears after transmission. Autoregressive generation keeps the key and value state for every token already seen, and it keeps that state across later steps [Kwon et al., 2023b]. Scheduling and memory allocation have to cooperate. A batch-size setting alone does not describe serving capacity.

This paper uses published implementation details and historical benchmarks. It does not report a new measurement of my single-node machine, one Threadripper PRO 7960X with one RTX PRO 6000. The recommendations are a test order, not promised speedups. Where a published gain came from a named engine, model, and arrival pattern, I leave it there. Stacking those gains into one planning factor is how a capacity spreadsheet overstates the case.

The contribution is the separation, and the rule that follows from it. Split admission, cache pressure, prefill interference, and host overhead. Change one of those mechanisms. Buy capacity only when a repeatable test shows that the accelerator, or the memory that holds the model, is the constraint the proposed replica or shard actually removes.

What Busy Fails to Say

Let a completed request i have arrival time a_i , first token time f_i , completion time c_i , and O_i output tokens. Time to first token is

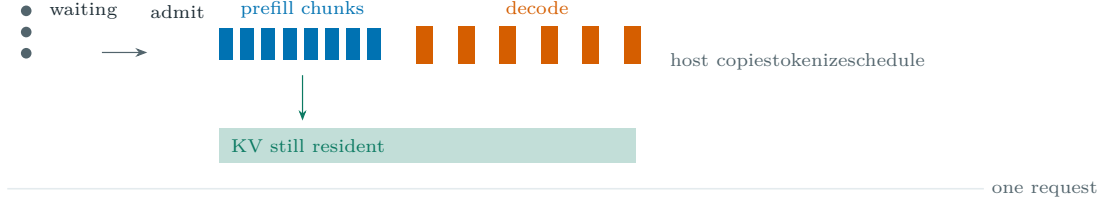


Figure 1: One request, left to right. Waiting requests are admitted, prompt work is cut into chunks, decode emits tokens, and the KV state stays resident under all of it. Host work sits beside the device, not inside the colored bars. Schematic. Not a trace.

$$\text{TTFT}_i = f_i - a_i.$$

Output events for that request occur at times $e_{i,1} = f_i, e_{i,2}, \dots, e_{i,K_i} = c_i$. The gap between successive events is

$$g_{i,k} = e_{i,k+1} - e_{i,k}.$$

Call $g_{i,k}$ an inter-token gap only when each event carries one token. If speculative decoding puts several tokens in one event, $g_{i,k}$ is an inter-event gap. A mean of those gaps hides a stall. If the service promise is a stream that does not hiccup, keep the distribution of $g_{i,k}$. Fail the request when the maximum gap exceeds the limit you set. Do not fail it on the mean alone.

Per-token time, for $O_i > 1$, is a different quantity:

$$\text{TPOT}_i = \frac{c_i - f_i}{O_i - 1}.$$

TPOT equals the mean event gap only when $K_i = O_i$, that is, when each event is one token. Say which timer you recorded.

Goodput, for this paper, is completed logical requests per second that meet the agreed latency limits, with each logical request counted once. For an observation window of length H ,

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

$\mathcal{C}$ is the set that completed in the window and met both limits. Rejected, cancelled, and still-running requests stay out of the numerator. They stay in the offered-request count when you report the qualifying fraction. If the client slows down when the server slows down, the run is no longer a server measurement. It is a measurement of a polite client.

This definition is an operating proposal. It is not a benchmark result. Choose τ_f and τ_p before you inspect the winning run.

Admission Is Not the Same as a Batch Size

Autoregressive generation proceeds one iteration at a time. In a static batch, a short completion leaves an unused place while a long completion is still running. Continuous batching makes the admission decision between iterations, so a new request can join without waiting for every current request to finish.

The useful networking analogy is resource admission, not a utilization target. Ask which work can enter, what state it must retain, and which requests wait behind it. Then optimize for completed requests inside a latency objective. A busier device is a fact to investigate. It is not proof of a better service.

Orca-style iteration-level scheduling is the idea Sarathi-Serve builds on when it argues that chunked prefill can keep a batch from stalling behind a long prompt [Agrawal et al., 2024]. I am not treating that paper’s application results as a universal multiplier. The mechanism is enough for the diagnosis. If short requests wait because the batch cannot refill, the missing object is an admission rule. It is not automatically another GPU.

Cache Pressure Is a Memory Fact, Then a Latency Fact

The June 2023 vLLM engineering report identified 60 to 80 percent memory waste in the systems it examined, from fragmentation and over-reservation [Kwon et al., 2023a]. That is a memory-management finding. It is not evidence that every GPU wastes that percentage of compute. Recovering cache space helps admission only when cache capacity is the limiting resource.

The same report says the KV cache is large, up to 1.7 GB for a single sequence on LLaMA-13B in their example, and dynamic, because sequence length is not known at arrival [Kwon et al., 2023a]. PagedAttention partitions each sequence into blocks. A block table maps logical positions to physical blocks that need not be adjacent. The engine allocates physical blocks as the sequence grows. It does not reserve one contiguous maximum-length region at arrival [Kwon et al., 2023b,a].

The report describes residual waste under 4 percent in its evaluated setting, concentrated in the last block of a sequence [Kwon et al., 2023a]. Treat that as an attributed historical result. Do not treat it as a guarantee for every byte a modern serving process consumes. Model weights, execution buffers, and other allocations sit outside that claim.

The SOSP paper’s throughput claim is a different sentence, and it should stay a different sentence. vLLM with PagedAttention achieved 2 to 4 times higher throughput than FasterTransformer and Orca at the same level of latency. The gain was more pronounced with longer sequences, larger models, and more complex decoding, in that paper’s setting [Kwon et al., 2023b]. The blog’s comparison against Hugging Face Transformers, up to 24 times on one of its one-completion plots, is a third baseline. The same post also reports a lower range against Hugging Face when each request asks for three completions, and up to 3.5 times against Text Generation Inference [Kwon et al., 2023a]. Calling any of these a pure scheduler improvement erases part of the mechanism. Do not multiply them.

When the KV cache cannot hold the active work, a serving engine can preempt a request and recompute its state later. The server can stay busy while it spends effort recovering work instead of finishing more requests. Watch cache occupancy and preemption next to queue delay. If preemptions rise, test a lower active-sequence limit before you test a larger allocation. A configuration that boots

is not a capacity test.

A logical floor, not a measurement: 16 requests, 8192 cached tokens, 128 KiB of KV per token, is 16 GiB. If the card has room for 8 of those requests after weights, a scheduler that admits 16 will preempt or refuse. Preemption looks like work. Refusal looks like a queue. The purchase proposal usually cites the queue. The first repair to test is “admit 8.”

Prefill Interference Is a Scheduling Choice

Prefill reads the prompt. Decode emits later tokens. Prefill is often compute-heavy. Decode is often limited by memory traffic. Those descriptions explain why a scheduler might mix the two. They do not prove the bottleneck on a workload you have not profiled.

Chunked prefill cuts a long prompt into pieces and interleaves those pieces with decode. vLLM’s stable documentation of disaggregated prefill says the same tail can be chased with chunked prefill, and that choosing the chunk size in practice is hard [vLLM Project, 2026]. Smaller budgets can leave more of the iteration for tokens already in flight. Larger budgets can move a new prompt faster. A recommendation that claims both tails always improve is hiding which workload paid.

Sarathi-Serve’s published case for chunked prefill is that a long prefill otherwise stalls decode, and that splitting the prefill removes that stall on the applications they measured [Agrawal et al., 2024]. DistServe assigns prefill and decode to different GPUs. On the applications and latency limits in that paper, it serves up to 7.4 times more requests or a 12.6 times tighter SLO than the state-of-the-art systems it compares, while staying inside the limits for more than 90 percent of requests [Zhong et al., 2024]. The abstract says those existing systems colocate the two phases. The gain is not a property of the word “disaggregation.”

vLLM’s documentation is blunter about throughput. Disaggregated prefill, on the stable page, does not improve throughput. It exists so time to first token and inter-token latency can be tuned apart, and so a long prefill does not inflate the gap between tokens [vLLM Project, 2026]. If the proposal in front of you quotes a DistServe multiplier and a vLLM feature in the same cell, split the cell.

For interactive traffic, define the acceptable gap between streamed outputs separately from the wait for the first output. For document work, keep the longest prompts you claim to support in the test. A budget selected on short prompts has not passed the workload that creates the interference.

Host Overhead Is the Part the GPU Graph Hides

Request state, scheduling, tokenization, detokenization, and response streaming consume CPU time around model execution. A GPU graph can look idle because the host has not handed it the next batch. Low utilization is then a host fact. It is not a batching fact.

A decode step of 15 ms, of which 9 ms is host copy and launch, is a sketch, not a measurement from this machine. That step will not become a 6 ms step because a second GPU was added, unless that GPU removes the copy. The estate becomes two servers with the same 9 ms tax. Profile the path before the device. Compare server-side timings with the client’s view, so a slow client does not become an imaginary GPU constraint.

Save the engine version, the full configuration, the model revision, the tokenizer, the driver, and the hardware identity with every run. Documentation defaults are a starting hypothesis. The running

process is the thing being measured.

Prefix Reuse Is Not Speculation

Two optimizations get bundled in capacity talks because both reduce a latency number. They reduce different numbers.

Prefix caching skips prefill for a repeated prefix. It does not make the new tokens cheaper to generate. A benchmark that warms the cache and then reports tokens per second has added a hit rate to a decode rate and called the sum a capacity. Split the runs: cold start, shared prefix, unshared prefix. If the hit depends on a system prompt you are about to edit, the capacity you just recorded will move when the prompt moves.

Speculative decoding spends extra compute to propose tokens the target might accept. Leviathan, Kalman, and Matias report a 2 to 3 times acceleration against the standard T5X implementation of T5-XXL, with identical outputs, when the correction is done so that the target distribution is preserved [Leviathan et al., 2022]. That result is about inter-token latency on that setup. It does not refill a KV cache that does not fit. Use it to avoid a memory purchase only after you have shown that the step is latency-bound at a batch the memory can hold.

A constant-acceptance toy, conditional probability α , γ proposals, expects $(1 - \alpha^{\gamma+1})/(1 - \alpha)$ tokens per verification cycle. For $\gamma = 4$ and $\alpha = 0.8$ that is 3.36 tokens. If each draft token costs a tenth of a target step and verification costs one target step, the speedup is about 2.4 times. The toy is not the T5-XXL result. It is a way to see that the acceptance rate is the whole game. A bad draft on your domain still costs the draft pass and still emits about one token.

Keep a precision change in its own experiment. If the candidate uses a different weight or cache representation, require a task-quality check beside the latency run. A serving change should not quietly change what counts as an acceptable answer.

What to Measure Before a Purchase

Use a fixed arrival window and a fixed drain period for every candidate. Replay arrivals independently of response completion. When the server falls behind, keep counting rejected, cancelled, and timed-out requests. Do not remove them from the result because the client gave up. The steps below are a protocol. I did not run them for this paper.

1. Record the baseline. Use the intended model and a representative request trace, including arrival timestamps. Keep sampling settings, stop conditions, and output-token caps fixed. Separate warm-up from measurement.
2. Record five counters on that run: queue delay, KV occupancy, preemption count, GPU active time, and host time in tokenize, schedule, copy, and launch. Use the engine’s own counters. If the engine does not expose one of them, stop. A utilization graph is not a substitute.
3. Stop if host time is the large share of the decode step. Change the host path. Repeat step 2. Do not buy a GPU for that signature.
4. Lower the active-sequence limit if preemptions rise and each sequence’s KV still fits in free device memory. Repeat step 2. If goodput recovers, the constraint was cache pressure. A replica is optional capacity, not the first repair.

5. Buy a replica only if all four of the following are true. Host time is not the large share of the step. Each request’s weights and KV fit on one GPU at the longest context you serve. The device is busy on useful tokens, not on preemption, at the concurrency you must serve. The same trace on a second copy meets τ_f and τ_p .
6. Buy or enable a shard only if one request does not fit on one GPU at that context. Confirm that the sharded placement fits. Record whether the new collective stays inside the server. Accept the shard only if the same trace meets τ_f and τ_p , and only if the added collective time is smaller than the miss you are removing.
7. Increase offered load until the agreed limits fail. Record failures, cancellation, and whether the tail recovers after the burst.
8. Repeat the candidate after a restart, and on the difficult prompt-length buckets. Keep the previous configuration available for rollback.

A replica copies the model and splits requests that already fit. It does not make one oversized sequence fit. A shard splits one model across GPUs so one sequence can fit. It adds a collective. It does not remove a host copy that still runs on every step. Parallelism can add communication. A CPU, transport, or single-request latency limit needs its own diagnosis. Additional replicas are not a universal remedy.

Report client-observed TTFT, the distribution of $g_{i,k}$, TPOT, and end-to-end latency. State whether each figure is a request-level distribution or an event-gap distribution. Report sample counts by prompt-length bucket. Do not average percentile summaries across buckets and then call the average a percentile.

A Cohort the Mean Would Pass

The goodput definition is easier to misread than to write down. This cohort is an example, not a trace. I did not collect it.

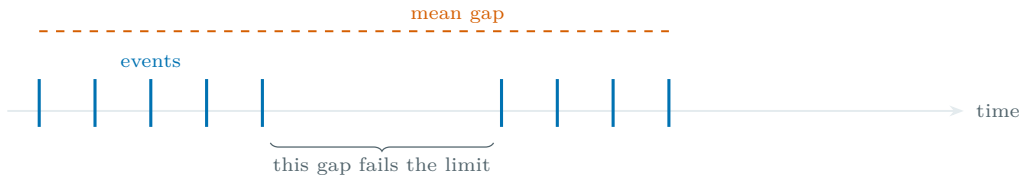


Figure 2: One stream of output events. The ticks are mostly even. One gap is long. The dashed line is the mean, which still looks acceptable. Schematic of the gate, not a trace.

Take an observation window of 60 seconds. Ten logical requests are offered. Eight complete. Two are still running when the window ends. Of the eight that complete, six meet both limits. Two meet the time-to-first-token limit and miss the maximum event-gap limit. One of those two has a single long gap in the middle and a mean gap that would still pass.

$$G = \frac{6}{60} = 0.1$$

completed requests per second. The qualifying fraction of offered requests is 6/10. The two unfinished requests stay in the offered count and stay out of the numerator. If the gate had used

the mean gap, and if that mean were under the limit, the count would be 8 and G would be 8/60. The example assumes that mean. It does not measure it. The point is the gate. A mean can pass a stream the maximum-gap rule rejects.

Choose τ_f and τ_p before you look at this table. If you choose them after you see which requests failed, the cohort is no longer a test.

Four Counter Patterns

The purchase steps name the counters. They do not show what a passing pattern looks like next to a failing one. The rows below are signatures to test, not results from a machine.

What the counters show	Constraint	First change
Host time is most of the decode step. GPU active time is low.	Host path	Change the host path. Do not add a GPU.
Preemptions rise. Each sequence still fits in free device memory.	Cache pressure	Lower the active-sequence limit. Repeat the trace.
Short requests wait. KV occupancy has room. Preemptions do not rise.	Admission	Admit between iterations. A static batch is the thing to change.
One sequence's weights and KV do not fit on one GPU.	Model memory	Shard, or shorten the context. A replica of the same GPU does not make that sequence fit.
Each sequence fits. The device is busy on tokens, not on preemption. Goodput still misses.	Accelerator capacity	Then test a replica on the same trace.

Read a row only if step 2 of the protocol recorded the counters in it. A missing counter means the row is unavailable. Do not fill it from a utilization graph.

Two rows can be true at once. A long prefill can sit in front of a short decode while the host is also late. Change one mechanism, then repeat the trace. If you change the token budget and the active-sequence limit in the same run, you will not know which change moved G .

How Often a Prefill Can Cut In

Chunked prefill does not have a universal chunk time in this note. It does have a count. If the scheduler splits a prompt of P tokens into pieces of at most C tokens, the prompt takes

$$n = \left\lceil \frac{P}{C} \right\rceil$$

pieces, provided it is not finished in one step. An 8192-token prompt at a budget of 512 tokens is 16 pieces. The same prompt at a budget of 2048 tokens is 4 pieces. A 32768-token prompt at 512 tokens is 64 pieces. These counts are arithmetic. They are not milliseconds.

Each piece is a chance to delay decode that is already in flight, if the scheduler cannot overlap that piece with the decode it is trying to protect. Sixteen chances are not sixteen times the damage of four chances until you measure the piece time. Record that time. Then multiply. Do not multiply a datasheet peak by n and call the product interference.

The DistServe split removes that interleaving by putting prefill on other GPUs. It also adds a transfer. The transfer is the subject of the fabric note. It is not free because the prefill no longer shares the decode GPU. vLLM’s stable page says the disaggregated path does not improve throughput, and that a chunk budget can chase the same tail [vLLM Project, 2026]. Pick one mechanism. Hold the trace still.

A budget selected on short prompts has not seen n for the longest prompt you claim to serve. Keep that prompt in the test. If the long prompt is absent, the interference case is absent.

A Static Batch Holds the Slot

Eight places are open. One request is still generating at output token 400. Seven finished at output token 20. In a static batch those seven places stay empty until the long request ends. The empty stretch is 380 iterations on each of those seven places. That is admission delay, not device work. Continuous batching can put a new request into a finished place at the next iteration. The count is the mechanism. It is not a throughput multiplier, and I am not attaching one. If the counters show free KV, no rise in preemptions, and short requests waiting, change admission before you buy a GPU.

Write the cohort down in one sheet. One row per offered request.

1. Write the arrival time, the first-token time, and the completion time. Leave completion blank if the request is still running.
2. Write the output-token count and the maximum event gap. Leave the gap blank only if no token was emitted.
3. Mark whether the request met τ_f and τ_p . A blank completion is a miss.
4. Count offered requests, including the misses. Divide the passes by the window length. That quotient is G .

A sheet that drops the unfinished rows will report a higher G than the definition allows. Keep the offered count on the same sheet as the completions. Do not compute the qualifying fraction from the completed rows alone.

The 400-token and 20-token lengths are an example, so the empty stretch is visible. They are not a trace from this machine. Replace them with the output lengths in the trace you will actually replay. If those lengths are uniform, the static batch has less idle admission to recover, and the purchase argument changes. Record the output-length distribution before you treat the queue as a GPU shortage.

The same sheet also separates a replica from a shard. Add two columns: whether each request’s weights and KV fit on one GPU, and whether a proposed shard keeps its collective inside the server. A row that fits, and whose device time is tokens rather than preemption, is a replica candidate. A row that does not fit is a shard candidate. A row whose host time dominates is neither. Do not let a single “GPU busy” cell collapse those three rows into one purchase.

Count the idle stretch before you price it. Seven finished places, each idle for 380 iterations, are

2660 iteration-slots that a static batch did not refill. If a new request would have occupied one of those places for 20 iterations, the batch could have started many of them and still have been waiting on the long request. The 2660 figure is 7×380 . It is not a measured goodput gain. It tells you where to look. If the trace has that shape, the first change is admission. If the trace does not have that shape, do not import this example into the purchase.

A uniform trace removes the idle stretch. Eight requests that all run to 400 tokens leave no finished place to refill. Continuous batching then has little admission work to do, and a long queue is more likely to be device work, cache pressure, or a host that has not launched the kernel. Record the output-length distribution first. The repair follows the distribution, not the name of the scheduler.

Limits

I did not operate a serving cluster for this paper. The vLLM blog, the PagedAttention paper, Sarathi-Serve, DistServe, and the speculative-decoding paper are the measurements. Mine are the boundaries and the arithmetic examples. The 16 GiB cache sketch uses 128 KiB per token. That figure is a GQA-style assumption from the companion note on local context, not a dump of a running allocator.

No engine release was benchmarked here. Feature defaults move. Check the installed documentation before you treat a sentence in this paper as a configuration.

AI assistance was used to retrieve pages, check arithmetic, and compile the PDF. I am the author of the argument. This note is not peer reviewed.

Conclusion

Trace the request before you change capacity. A queue can be admission, cache pressure, a long prefill in front of a short decode, or a host that has not launched the kernel. Those are different repairs. A GPU purchase belongs at the end of that list, after the path has shown what is actually missing.

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>.
- Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Yu, Joey Gonzalez, Hao Zhang, and Ion Stoica. vllm: Easy, fast, and cheap llm serving with pagedattention. vLLM Blog, 20 June 2023. Retrieved 17 October 2026, 2023a. URL <https://blog.vllm.ai/2023/06/20/vllm.html>.
- 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, 2023b. 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>.

vLLM Project. Disaggregated prefilling (experimental). vLLM documentation, stable. Retrieved 17 October 2026, 2026. URL https://docs.vllm.ai/en/stable/features/disagg_prefill/.

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>.