Episode Book

EPISODE #5960

There's No Such Thing as Tokens Per Second

PUBLISHED	14 October 2026 at 14:32 UTC
DURATION	30:16
LISTEN	myweirdprompts.com/episode/llm-inference-performance-metrics-72413/
HOSTS	Corn, Herman, Hilbert
SCRIPT MODEL	deepseek-flash
VOICES	chatterbox-regular (chatterbox-local)
PIPELINE	V5.4
SOURCES	14 cited by the research (listed at the end)
BOOK GENERATED	19 October 2026 at 17:41 UTC

SYNOPSIS

Everyone quotes tokens per second and time to first token as if they're the whole picture. They're not — an LLM serving system doesn't have a performance number, it has a surface. This episode breaks down prefill versus decode, why one is compute-bound and the other memory-bandwidth-bound, the latency and throughput metric families that quietly disagree with each other, and how to design a benchmark that doesn't lie to you. Plus: goodput, the queue cliff, and the worked example where 2,000 tokens per second turns out to mean four out of ten requests were acceptable.

TRANSCRIPT



Corn

Last time we did this, the benchmark was about whether the transcript came back right. Word error rate, accuracy, the whole thing treated as an exercise in transcription quality...



Herman

And a completely separate exercise in whether the thing could keep up.



Corn

Right. This time Daniel wants the second half of that, and he wants it about language models generally. Which means we're doing inference performance properly. His framing, near enough, is that everybody quotes tokens per second and time to first token and treats those as the picture, and he wants the rest of it. The full metric families. How prefill and decode differ and why that changes what you're even measuring. What's worth collecting on the hardware side versus what's just more telemetry, his phrase. The actual tooling, GenAI-Perf and MLPerf and vLLM's bench utilities and Prometheus and Grafana, plus where OpenTelemetry fits. Whether you can compare engines at all or whether each one needs its own instruments. And then the part I think he actually cares about...



Herman

Designing the test.



Corn

Designing a test that doesn't lie to you. Impressive numbers that mean nothing, that's the thing he's poking at.



Herman

The line I'd put at the top is that there's no such thing as tokens per second.



Corn

There's one number everyone quotes.



Herman

There's a family of numbers, and they disagree with each other. The disagreements are the entire information content of the exercise.



Corn

So the first thing to get straight is that an LLM serving system doesn't have a performance number. It has a surface.



Herman

A surface, yes. Latency, throughput and resource saturation all trade against each other. Pick any two and you're sacrificing something on the third. So when someone hands you a single figure, they've collapsed a surface into a point, and they've chosen which dimension to hide.



Corn

Which dimension do people usually hide?



Herman

The latency tail. Almost always. Because the average looks fine and the ninety-ninth percentile looks like a car crash, and one of those makes a better slide.



Corn

Herman.



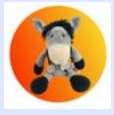
Herman

I'm just describing the industry.



Corn

You're describing yourself. You have a slide deck.



Herman

I have several. And the reason this is a surface rather than a number is that inference happens in two phases that stress completely different parts of the hardware. That's the organising idea for the whole episode and it explains almost every confusing number you'll ever read. Prefill and decode.



Corn

Prefill first.



Herman

Prefill takes your entire prompt and runs it through the forward pass in one go to build the KV cache. Every token in the prompt gets processed, in parallel, all at once. That makes it compute-bound. High arithmetic intensity, lots of matrix multiplication against lots of data, the GPU is busy doing maths. And the output of prefill is the thing you actually experience as time to first token. Long prompt, longer prefill, later first token.



Corn

And decode is the slow drip.



Herman

Decode generates one token at a time. Each new token reads the whole weight matrix and the whole KV cache out of high-bandwidth memory, does a comparatively tiny amount of arithmetic, writes one token, and then does it all again for the next one. That's memory-bandwidth-bound. The GPU is mostly waiting on memory. And that's what sets the gap between tokens.



Corn

So the second phase is the one where the expensive hardware sits idle waiting on a bus.



Herman

Not idle exactly, but yes, it's starved of bandwidth rather than starved of compute. And that's why a chip with more memory bandwidth wins on decode even with identical compute. The AMD MI355X moves eight terabytes a second, the MI325X is six. Same model, same software, the faster memory wins on tokens per second per user.



Corn

You've got that number ready.



Herman

I've had it ready all week.



Corn

So the practical consequence is that a benchmark with long prompts and short outputs is measuring prefill, and a benchmark with short prompts and long outputs is measuring decode.



Herman

And those two runs will produce tokens-per-second figures that are not comparable to each other, for the same model, on the same GPU, on the same afternoon. Which is where we should start on the metrics themselves, because people quote tokens per second as though it's one number, and it isn't even one kind of number.



Corn

Go.



Herman

The latency family first. Time to first token is exactly what it sounds like, request goes in, first streamed token comes out. It's driven by how long the prompt is, how long the request sat in a queue, and how fast prefill runs. It's the primary perceived-responsiveness metric, because it's the wait before anything happens at all. And the targets are tight. A chatbot wants to be under five hundred milliseconds. Code completion wants under a hundred.



Corn

A hundred milliseconds for code.



Herman

Because you're typing. If the ghost text arrives after you've already typed the next word it's worthless, it's not slow, it's wrong.



Corn

Fair.



Herman

Then time per output token, which is where the definitions start to bite. TPOT is the average gap between tokens after the first, and the formula is end-to-end latency minus time to first token, divided by output tokens minus one. That minus one is because the first token isn't a gap, it's the arrival.



Corn

And inter-token latency is the same thing.



Herman

Almost. For a single request, mean inter-token latency and TPOT are the same number. Across a whole benchmark run with many requests, they stop being the same number, and the reason is weighting. Average TPOT is request-weighted. Average inter-token latency is



Herman (continued)

token-weighted. And those are different averages of the same underlying gaps.



Corn

Say that again, because that's the kind of thing that ruins an afternoon.



Herman

Two requests. One generates a hundred tokens, one generates ten. The short one has a couple of slow gaps in it. If you average the per-request TPOT values, the short request counts as much as the long one, so those slow gaps pull the average up hard. If you average every inter-token gap across both requests, the long request contributes a hundred gaps and the short one contributes ten, so those same slow gaps barely move it. Same run. Two different averages. Two tools that both describe themselves as reporting average inter-token latency will disagree, and neither is wrong.



Corn

So when you compare two engines on a dashboard, you might be comparing a request-weighted number to a token-weighted number and calling it a speed difference.



Herman

You might. And there's no way to tell from the outside, because both tools call it latency. You have to go read what each one means. It's the single most common way a cross-tool comparison goes quietly wrong.



Corn

And end to end?



Herman

End to end is network overhead plus time to first token plus tokens times TPOT. It's what the user actually waited, wall clock, request to last token.



Corn

The user waited.



Herman

That's the one that matters to a human and the one that appears least often on a marketing chart.



Corn

So now the throughput family, because tokens per second is three different measurements wearing one coat.



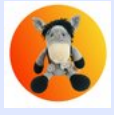
Herman

At least three. System throughput is total output tokens per second across every request in flight. That's the number on the big chart, and it's a property of the server, not of any user's experience. User throughput is roughly the inverse of inter-token latency at long outputs, and it falls as you add concurrent requests, because you're sharing the same hardware.



Corn

The server gets faster, the user gets slower.



Herman

That's exactly what happens, and it's the most important sentence in this episode. Then completed requests per second, which is a count rather than a rate of token generation, and useful when requests have wildly different sizes.



Corn

Alright. So the server number goes up, the user number goes down, and somewhere between them is the truth. Where does the queue come in?



Herman

The queue is the thing that turns a smooth chart into a cliff. As concurrent requests climb, you'd expect the per-token gaps to widen, and eventually they do. But the first thing to break is time to first token, because a request that hasn't started prefill yet is just sitting in a queue. Nothing about the model is slow. It's waiting for a slot.



Corn

And that's visible if you know where to look.



Herman

It's visible in two counters. How many requests are running, and how many are waiting. If anything is waiting, your queue is building, and you are over-concurrency rather than under-provisioned. That distinction is the difference between adding hardware and lowering your batch size.



Corn

Hold on. Say that once more, plainly. Waiting requests means what?



Herman

It means the engine has more requests than it's willing to run at once. So you're not short of capacity, you're past the point where extra concurrency helps. Adding a GPU will raise the ceiling, but the immediate cause of your bad time to first token is that a request is parked in a queue behind other work.



Corn

So somebody looks at a bad p95, buys another card, and gets the same p95 because the queue just got longer and wider.



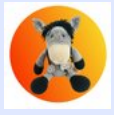
Herman

They get a slightly better p95 and a much larger bill.



Corn

There we are.



Herman

And this is where percentiles stop being a reporting nicety and become the actual analysis. You report p50, p95 and p99 for time to first token, inter-token latency, and end to end. Never just the mean.



Corn

Give me the case for the tail. I want the version you'd use on someone who says the average is fine.



Herman

A two hundred millisecond mean time to first token with an eight second p99 is a broken chat interface. The average is fine. The average is wonderful. One request in a hundred waits eight seconds to see anything, and those are the requests people remember, because the experience of waiting eight seconds for the first character is the experience of the product being broken.



Corn

And which percentile is the one to promise?



Herman

p95 is the common service-level line. p50 is what the typical user gets. p99 is near worst case, and it's where the architecture's bad days live.



Corn

Now. Goodput.



Herman

Goodput is the fraction of requests that meet all your service-level objectives at once. Not one of them. All of them. So you'd set something like time to first token under five hundred milliseconds, time per output token under fifteen, end to end under two seconds, and goodput is the share of requests that satisfy every one of those conditions simultaneously.



Corn

And that number goes the other way.



Herman

It goes the other way. As you push load up, raw system throughput keeps climbing, because the server is doing more work per second. Goodput climbs too, to a point, and then it starts falling, because the extra requests are being served badly. You end up with a chart where tokens per second is at its maximum and most of your users are not getting a usable response.



Corn

So the peak of one chart is past the edge of the other.



Herman

That crossover is the real capacity limit of the system. Not the maximum throughput. The point where throughput and service quality part company.



Corn

Here's the worked one, then. Same model, same card. Two thousand tokens a second system throughput, and p95 time to first token of three seconds, because the queue is deep. What's the goodput?



Herman

If your service-level objective is five hundred milliseconds, nearly everyone above the median is failing the first-token test. You'd be looking at goodput around forty percent. So the headline number is two thousand, and the honest number is that four out of ten requests were acceptable.



Corn

Two thousand.



Herman

That's the gap the whole episode is about. A single tokens-per-second figure isn't wrong, it's just answering a question nobody asked. The question people mean to ask is how many requests can I serve acceptably, and that's goodput.



Corn

And then the other worked example, the prefill versus decode one. Two workloads, same model. Four thousand token prompts with fifty token outputs, versus two hundred token prompts with eight hundred token outputs.



Herman

The first stresses prefill completely. Enormous prompt, built once, tiny generation on the end. Time to first token is everything and the per-token gap barely shows up. The second is the mirror image. Small prompt, long generation, so time to first token is trivial and the whole cost is decode, one token at a time off the memory bus.



Corn

And if someone quotes a single tokens-per-second figure for both of those...



Herman

They've averaged two different machines. A system tuned for prefill throughput and a system tuned for decode throughput are different builds, sometimes different hardware choices, and always different batching decisions. Quoting one number across both is like quoting a car's top speed and its towing capacity as a single figure.



Corn

It'd be a good car.



Herman

It would be the best car.



Corn

So we know what to measure on the wire. Next question is what to measure on the box, and which of those signals actually tell you something.



Herman

The short list, and I mean short, because most of what a GPU will tell you is noise. Number one, KV-cache utilisation. That's the real saturation signal. The KV cache is the memory the engine holds for every token of every request currently in flight. When it pins at a hundred percent, the engine can't admit new work, so requests queue, and time to first token inflates.



Corn

So the queue symptom from earlier has a cause, and this is it.



Herman

That's the chain. Cache full, admissions block, queue grows, first token gets late. If you watch one thing on a serving box, watch that. Number two, running versus waiting requests, which is just the queue made visible. Number three, the prefill-versus-decode time split, because that tells you which of the two phases is actually costing you. And



Herman (continued)

number four, GPU utilisation and memory, not as a performance number but as a way to tell compute-bound from starved.



Corn

And power?



Herman

Power is a facility question more than a performance one, though MLPerf now measures it, wall-measured system power and energy per stream, because at scale the electricity is a line item.



Corn

The card that watches its own usage.



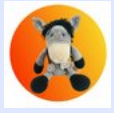
Herman

You'd approve of that.



Corn

I would. It's the only part of this I've enjoyed so far.



Herman

The useful discipline is a speed-of-light model. You calculate the best a single request could possibly do on this hardware, and then you compare your measured number to that ceiling. For decode, single-request tokens per second at the limit is HBM bandwidth divided by the model weights plus the context length times the KV bytes per token.



Corn

Plain version.



Herman

Every token you generate has to move the entire model's weights plus everything the request has cached, out of memory and into the compute units. Bandwidth divided by bytes moved is your ceiling. That's it. That's the whole model.



Corn

And that ceiling is why the memory-bandwidth number on the spec sheet matters more than the flops number for chat workloads.



Herman

For decode, yes. And the reason that's better than staring at a GPU utilisation percentage is that utilisation has no reference point. Ninety percent utilisation could be excellent or terrible and the number itself won't tell you which. Your measured tokens per second against the theoretical ceiling tells you whether there's room.



Corn

Do the kernel thing. The one about benchmarks not travelling.



Herman

A kernel at ninety percent of peak floating point throughput on an A100 can drop to thirty percent on an H100, on the same code, with no changes. Because the ratio of compute to memory bandwidth changes between generations. The old kernel was shaped to the old balance, and it lands on the new card misaligned.



Corn

So you rerun it and the software got worse.



Herman

The software is identical. The hardware moved underneath it. Which is the argument for never trusting an optimisation benchmark older than the generation you're deploying on.



Corn

Any optimisation benchmark in particular?



Herman

All of them. It's the reason "portable performance" is a fragile phrase. You can port the code. You cannot port the assumption that the code is shaped correctly for the machine it's now sitting on.



Corn

Tools next, because Daniel listed a few and I want to know which ones are real.



Herman

Two families, and conflating them is a named mistake. Load testers generate controlled traffic against an endpoint and report token metrics. Observability platforms scrape a running server and report what it's doing. Different questions, different instruments.



Corn

Load testers first.



Herman

vLLM ships the most complete set of the engines, because it's the same project as the server. There's an online benchmark against an OpenAI-compatible endpoint, and it reports time to first token, time per output token, inter-token latency and throughput natively. It takes ShareGPT conversations or synthetic data with a controlled input and output length. It has a ramp-up mode so you can watch the system under increasing load. And it can compute goodput out of percentiles rather than making you do it by hand.



Corn

And the rest of the subcommands.



Herman

There's a latency benchmark for a single request, a throughput benchmark for offline maximum, a sweep that walks across parameter combinations, and then a plot command that draws you the Pareto frontier from the sweep results. That last one is the one people forget exists.



Corn

NVIDIA's client-side one.



Herman

GenAI-Perf is a client-side tool, so it sits outside the server and measures what a caller would see. Output token throughput, time to first token, inter-token latency, request throughput. It has a goodput mode with the same definition, completed requests per second that meet specified metric constraints. And its analyze subcommand emits p99 performance and p99 GPU telemetry columns side by side, which is a small thing that saves a lot of arithmetic.



Corn

And it hits the same endpoints.



Herman

OpenAI-compatible chat, completions, embeddings. It pairs with their Perf Analyzer if you want to go deeper.



Corn

The standardised suite.



Herman

MLPerf Inference is the only thing in this space that's a common yardstick across vendors, and it's now at version six point one for datacenter. Four scenarios so you can't just run the one that flatters your hardware. Closed division, where you're not allowed to change the model or the maths, versus Open division, where you are. Available, Preview and RDI categories for how ready the submission is. And the power measurements, wall-measured system power and energy per stream.



Corn

And it's the only cross-engine comparison that exists.



Herman

Largely. Beyond MLPerf, each ecosystem's numbers come from its own tooling, so you're comparing instruments as much as engines.



Corn

Which is exactly what Daniel is worried about. The rest of the load testers.



Herman

TensorRT-LLM ships its own bench with throughput, latency and build subcommands and a dataset preparation helper. For production capacity work, the vLLM project points people at GuideLLM. And there are lighter options, llmperf and an LLM-aware Locust setup, if you want something smaller.



Corn

Now observability, because that's a different animal.



Herman

Prometheus and Grafana, scraping the live server. vLLM exposes a metrics endpoint with counters that are worth knowing by name. Generation tokens and prompt tokens, obviously. Prefix cache hits and prefix cache queries, which tell you whether your prompt



Herman (continued)

caching is doing anything. Number of preemptions, which is the engine throwing out a request's work to make room for another, and that's a sign you're over capacity. And admission rejections, which is the engine refusing work at the door.



Corn

Preemptions and rejections. Those are the two I'd want on an alert.



Herman

Those are the two. Ray Serve ships an LLM-specific Grafana dashboard, token throughput, time to first token, time per output token, end to end, the prefill and decode split, and peak throughput per model, and it's on by default from recent Ray versions.



Corn

And OpenTelemetry, where does that sit?



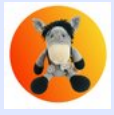
Herman

That's the per-call layer, and it's different from the server metrics. OpenTelemetry's GenAI conventions define a span for an inference call, and metrics like client token usage and client operation duration, plus server-side first-token time and time per output token. So you can trace one request through a stack rather than watching an aggregate.



Corn

And the sensible default.



Herman

By default it captures no prompt content and no tool arguments at all. Only metadata, model name, token counts, durations. Because prompts contain things that shouldn't end up in a tracing backend. That was a deliberate choice, and it's the right one.



Corn

You sound like you approve.



Herman

I do approve. I've seen what ends up in logs.



Corn

Anything worth noting about the conventions themselves?



Herman

They got big enough to split out. The GenAI semantic conventions now live in their own repository, spun out in May, and it's an active piece of work with a lot of open issues. Which tells you the shape of the thing is still moving.



Corn

Engines. Daniel asked whether benchmarking differs per engine, and the answer looks like yes by construction.



Herman

Each one exposes its own tooling because each one has its own internals. vLLM has the bench suite. TensorRT-LLM has its own bench with a compiled engine step. Text Generation Inference is a Rust server over gRPC, and its instrumentation follows that design. llama.cpp runs GGUF quantised models and its whole selling point is compile once, run anywhere.



Corn

And the comparison people actually quote.



Herman

There was a benchmark from Jan comparing TensorRT-LLM and llama.cpp on the same hardware. TensorRT-LLM came out thirty to seventy percent faster on token generation, with compiled models more than twenty percent smaller. And then the caveats, which are the interesting half. It's less portable, and it doesn't run on older NVIDIA GPUs at all.



Corn

So it's faster exactly where it runs.



Herman

Which is the honest summary of every engine comparison ever published.



Corn

Alright. Designing the test, because that's the bit that decides whether any of this was worth doing.



Herman

Control the things that change the answer. Input length and output length, separately, because they stress different phases. Concurrency and request rate, and burstiness, because a steady rate and a spiky one are different systems. Batch size in the engine, quantisation and precision, GPU count, tensor parallelism, engine version. Write all of it down, because a number without its configuration is a rumour.



Corn

And the cache sizing.



Herman

Maximum concurrency is roughly your KV cache in tokens divided by the maximum sequence length. For capacity planning you test at eighty to ninety percent of that, not at a hundred, because a hundred is the number where it falls over.



Corn

And the way you actually pick a deployment point.



Herman

Sweep concurrency. One to a hundred and twenty-eight is a common baseline range. Plot the results. You get a Pareto frontier, latency on one axis, throughput on the other. And you pick the point at the edge of your latency service level, not the point of maximum throughput, because maximum throughput is on the far side of the cliff.



Corn

The most optimal point isn't the fastest point.



Herman

It's the fastest point you can still call acceptable. Those are almost never the same coordinate.



Corn

Mistakes, then. The ones that produce impressive numbers.



Herman

Measuring HTTP request duration and calling it inference latency is the big one. It captures the whole response, so time to first token is invisible inside it, and the streaming gaps are invisible inside it. You get a perfectly reasonable-looking number that contains none of the information you wanted.



Corn

Ignoring output-length variance.



Herman

If some requests generate twenty tokens and some generate two thousand, your average is a blend of two different workloads and describes neither. Unbounded concurrency is the next one. If you keep adding load until the system collapses, you haven't measured capacity, you've measured the collapse point, and they're not the same event.



Corn

Confusing system throughput with the user's experience.



Herman

Which is the goodput point again, and it's the one people keep relearning. Reporting averages instead of percentiles. Using a micro-benchmarker, one of the engine's own tools, as a production service-level evaluator, which it isn't built to be. And forgetting prefix caching, which is a quiet one, because if your benchmark prompts share a prefix and your production traffic doesn't, or the reverse, the numbers won't transfer.



Corn

So the summary of the entire business is that the metric is an argument about what matters.



Herman

And most published tokens-per-second figures are arguments nobody checked.



Hilbert

It's not latency. It's a wait, and you're all calling it the wrong thing.



Hilbert

If I stand in a queue at the yard, that's not the thing being slow. That's me waiting for a slot. You said it yourselves. You said it an hour ago and then you went back to calling the lot latency.



Herman

That's a fair distinction, actually.



Hilbert

I've been timing it. Forty-one days.



Corn

Timing what?



Hilbert

The gate. There's a man on the gate at the builders' merchants on the industrial road, and he lets you in when he feels like it, and I've been writing it down. Every visit since the eighth of September. I've got a notebook. Time I arrive, time the barrier lifts.



Herman

You're measuring his queue.



Hilbert

I'm measuring the system. Barrier up, barrier down, in the book, and at the end of the day I work out the average. Two minutes and forty seconds.



Corn

On average. What's the worst one?



Hilbert

I don't record the worst one. I record the average.



Herman

So you've built a system where the only number you have is the one you told us not to trust.



Hilbert

It's the honest one. Nine visits. One of them I was there fifty-five minutes because a lorry was blocking the entrance, but that doesn't change the average much, so I left it in with the others.



Corn

You left it in.



Hilbert

It happened. It goes in the book.



Herman

Hilbert, do you see the problem?



Hilbert

I see nine visits in a notebook and a mean of two minutes forty, and I'll hear no complaints about it from two people who spent an hour telling me that a single number is a lie. If the average is a lie then my average is a lie and I still know exactly how long I wait at that gate.



Corn

That's airtight.



Hilbert

Thank you.



Corn

Did you ever ask the man to open the barrier faster?



Hilbert

He's ninety-one. His uncle had the post before him. The barrier went in during the fuel crisis and it's never once been serviced, and it takes eleven seconds to rise, which I've measured, and that's the gate, not the man. I'm not going to stand there and complain about the eleven seconds.



Herman

That's a prefill and decode split.



Hilbert

It's a barrier.



Corn

What do you want us to do with this?



Hilbert

Nothing. But the ninth visit was Tuesday and it was one minute and ten seconds, and if it stays there the average comes down to two minutes twenty, and then I'll have to redo the whole page.



Corn

So you'd rather the number be right than be good.



Hilbert

I'd rather the number be the number.



Herman

Which is more or less where we landed, just with fewer lorries.



Corn

Leave it. The thing we keep circling is that the impressive chart and the honest chart point in different directions.



Herman

And the crossover is the only number that means anything operationally.



Corn

So if you take one thing from this, it's that tokens per second is not a performance measurement. It's a coordinate, and if you don't know what it's costing you on latency, you don't know where you are on the surface.



Herman

The corollary being that the system showing the highest throughput is often the one serving the fewest acceptable responses, and the only way to see that is to measure goodput rather than generation rate.



Corn

There's still no standard way to compare engines. The definitions themselves vary. TPOT and inter-token latency get used interchangeably in plenty of published material, and some sources do treat them as the same quantity. Check the definition before you trust a cross-tool number.



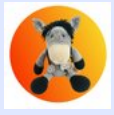
Herman

Observability is splitting down the middle while everyone watches. Live server metrics through Prometheus and Grafana on one side, per-call tracing through the GenAI conventions on the other. Different questions, constantly conflated. As those conventions mature, somebody has to decide which becomes the default and which becomes the fallback.



Corn

My guess is per-call wins eventually, because tracing a single request is what you actually need when something goes wrong for one user, not ten thousand. And the aggregate was always a summary of requests you never got to look at.



Herman

That's the thread to leave people with. The aggregate tells you a system is unwell. Only the individual call tells you why.



Corn

Which is more or less what Hilbert said about the gate. Thanks to our producer, Hilbert Flumingtop.



Herman

If you want to argue with any of this, or you think our numbers are wrong, we'd rather hear it than not.



Corn

Send us your own prompt on Telegram at t dot me slash MWP listener bot. Or find everything at my weird prompts dot com.



Herman

If this was your kind of episode, go back for episode three, Safetensors or something else; and episode eight, Building Your Own Whisper. This has been My Weird Prompts, the human-AI collaboration podcast.



Corn

We'll be back soon.

SOURCES

What the research for this episode read before the script was written. Primary sources first.

1. **vLLM Docs [primary]**
   Benchmark CLI
   https://docs.vllm.ai/en/latest/benchmarking/cli.html

2. **vLLM Docs [primary]**
   Production Metrics
   https://docs.vllm.ai/en/latest/usage/metrics.html

3. **vLLM Docs [primary]**
   Metrics design doc
   https://docs.vllm.ai/en/latest/design/metrics.html

4. **NVIDIA Triton Docs [primary]**
   GenAI-Perf README
   https://docs.nvidia.com/deeplearning/triton-inference-server/user-guide/docs/perf_benchmark/genai-perf-README.html

5. **NVIDIA Triton Docs [primary]**
   Benchmark Goodput with GenAI-Perf
   https://docs.nvidia.com/deeplearning/triton-inference-server/user-guide/docs/perf_analyzer/genai-perf/docs/goodput.html

6. **MLCommons [primary]**
MLPerf Inference: Datacenter v6.1
https://mlcommons.org/benchmarks/inference-datacenter/

7. **NVIDIA TensorRT-LLM Docs [primary]**
Benchmarking / trtllm-bench
https://nvidia.github.io/TensorRT-LLM/developer-guide/perf-benchmarking.html

8. **Modular LLM Inference Handbook**
Key metrics for LLM inference (undated)
https://handbook.modular.com/llm-inference-basics/llm-inference-metrics/

9. **Anyscale Docs**
Understand LLM latency and throughput metrics (undated)
https://docs.anyscale.com/llm/serving/benchmarking/metrics

10. **DigitalOcean**
LLM Inference Benchmarking: Measure What Matters (updated 2026-02-17)
https://www.digitalocean.com/blog/llm-inference-benchmarking

11. **qainsights perf-skills**
LLM Inference Performance reference (undated)
https://github.com/qainsights/perf-skills/blob/main/skills/perf/references/topics/llm-inference.md

12. **Jan.ai**
Benchmarking NVIDIA TensorRT-LLM (undated)
https://jan.ai/post/benchmarking-nvidia-tensorrt-llm

13. **OpenTelemetry Blog**
Inside the LLM Call: GenAI Observability (2026-05-14)
https://opentelemetry.io/blog/2026/genai-observability/

14. **OpenTelemetry**
semantic-conventions-genai repo (created 2026-05-05)
https://github.com/open-telemetry/semantic-conventions-genai