Episode Book

EPISODE #5899

Same GPU, Same Weights, Different Answers

PUBLISHED	12 October 2026 at 12:18 UTC
DURATION	21:32
LISTEN	myweirdprompts.com/episode/inference-engine-benchmarks-comparison-25521/
HOSTS	Corn, Herman
SCRIPT MODEL	deepseek-flash
VOICES	chatterbox-regular (chatterbox-local)
PIPELINE	V5.4
SOURCES	12 cited by the research (listed at the end)
BOOK GENERATED	19 October 2026 at 17:41 UTC

SYNOPSIS

Two servers, same GPU, same weights, same prompt — different output. This episode digs into Daniel's thought experiment: three providers running vLLM, SGLang, and TensorRT-LLM on identical hardware, where the customer can't choose the engine. Does the inference engine actually change speed, cost, and the answers themselves? We walk through the architecture of all three — PagedAttention, RadixAttention, and the TensorRT-LLM backend overhaul that removed TensorRT — then look at the benchmark data, where the fastest engine changes with the load, and a 2026 CISPA paper showing that swapping only the backend moved GSM8K scores by up to 16.6 percentage points on the same weights. If you rent endpoints and never see the middle layer, this one's for you.

TRANSCRIPT



Corn

Here's the question that stuck with me all morning. Two servers. Same GPU. Same weights. Same request. Different answer.



Herman

And nobody changed the model.



Corn

Nobody changed the model. That's the part that bothers me.



Herman

Well, Daniel's thought experiment. He wants three providers, identical hardware, identical VRAM, identical weights, and the only difference between them is the inference engine underneath. One's running vLLM, one's running SGLang, one's running TensorRT-LLM. And the customer can't pick which one they get.



Corn

He wants to know if that's a footnote or a first-order decision. Whether the engine is just an execution layer you can swap out, or whether it changes the speed, the cost, and the answers the model gives.



Herman

Three dimensions. Performance, output quality, and operational efficiency.



Corn

And then the practical one underneath all of it. If you're an ordinary developer picking a provider, how much should you actually care what engine they're running, versus just worrying about hardware, price, and whether the thing stays up.



Herman

Right, so before we argue about whether engines differ, we have to say what an engine is.



Corn

Because it isn't the model and it isn't the GPU.



Herman

It's everything in between. The software layer that loads a trained checkpoint, takes your request, and turns it into generated tokens. The model is the weights. The engine is the machinery that runs them.



Corn

And it's a foggy layer for most people. You rent an endpoint, you send a prompt, tokens come back. Nobody shows you what's in the middle.



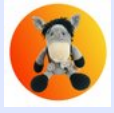
Herman

Which is exactly why the experiment is worth doing. All three engines now share the same baseline feature set. Continuous batching, paged KV cache, prefix caching, quantization down to FP8, INT8, INT4, speculative decoding, tensor and pipeline parallelism. That feature gap closed over 2024 and 2025.



Corn

So the brochure features stopped being the difference.



Herman

The features stopped being the difference. What separates them now is architecture. The question isn't whether an engine supports batching. It's how it schedules, how it caches, and how it executes when the batching gets heavy.



Corn

Which is a much harder thing to put on a pricing page.



Herman

Much harder. And one note on the model name before we go further. DeepSeek Flash 4 isn't a real artifact. The closest thing is DeepSeek V4 Flash, which shows up as a supported model in SGLang's repo.



Corn

Doesn't matter for the experiment, though.



Herman

Doesn't matter at all. These findings are model-agnostic. You could run them on anything.



Corn

So the engine is the layer between the weights and the tokens. Let's get under the hood, because the three of them solve the same problem three different ways.



Herman

Start with vLLM, because it set the template everybody else builds on. PagedAttention, Kwon and colleagues, SOSP 2023.



Corn

That's the one that treats the KV cache like operating system memory.



Herman

Exactly like it. The cache gets chopped into sixteen-token pages instead of one contiguous block per sequence. Before that, you were wasting something like forty percent of your VRAM on fragmentation. Fragmentation drops to under four percent.



Corn

Let me make sure I've got the analogy right, because it's a good one. It's like the difference between allocating a fixed-size parking spot for every car that might ever show up, versus a valet system that parks them wherever there's room and keeps a map.



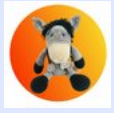
Herman

That's almost exactly it. The old way reserved a contiguous block sized for the worst case, so a request that only needed four hundred tokens still held a slot big enough for four thousand. PagedAttention hands out pages on demand and the map is the page table. Same physical VRAM, way more concurrent sequences fitting inside it.



Corn

And what's running the show on top of those pages?



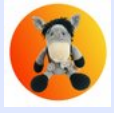
Herman

A Python scheduler with pluggable attention backends. FlashAttention-2, FlashInfer, xFormers, take your pick. The coverage is the other half of the story. Four hundred plus model architectures. NVIDIA, AMD ROCm, Google TPUs, Gaudi, plain CPU.



Corn

So vLLM is the generalist.



Herman

The generalist, and the one that made the whole category serious. Now SGLang takes a different cut. RadixAttention, Zheng and colleagues, NeurIPS 2024. Instead of caching prefixes per sequence, it builds a radix tree.



Corn

A trie.



Herman

A trie over every cached KV page in the system. So instead of one system prompt getting reused, it finds the longest common prefix across all prior sequences at once. On few-shot work, or anything branchy where requests share bodies of text, SGLang claims to eliminate seventy to ninety percent of the prefill compute.



Corn

That's a lot of compute to be throwing away for free.



Herman

It's a different data structure, and it shows. The other thing SGLang does is constrained decoding. JSON, regex, grammars. That lives in the scheduler, not bolted on top as a wrapper.



Corn

So a provider running SGLang gives you structured output as a first-class citizen.



Herman

As a first-class citizen, yes. And the expert parallelism for very large mixture-of-experts models is where SGLang has been strongest. That's the DeepSeek-R1 and V3 scale.



Corn

Which is why they list the V4 Flash models in their repo.



Herman

That's exactly why. Then there's TensorRT-LLM, and this is where the story has changed.



Corn

The compiled engine.



Herman

Historically, yes. It compiled the model into a hardware-specific engine file. Aggressive operator fusion, kernels auto-tuned to your exact GPU, CUDA graphs capturing the whole forward pass. NVIDIA-only territory.



Corn

Play to the metal.



Herman

And here's the update that changes the argument. TensorRT-LLM 1.2 removed the TensorRT backend entirely. PyTorch is now the only execution backend, and the engine-build command line is gone.



Corn

So the compiled-kernels-beat-Python story is, in part, history.



Herman

Which matters because a provider selling you on TensorRT-LLM today may not be running the thing people mean when they say TensorRT-LLM.



Corn

You're buying the reputation.



Herman

You may be buying the reputation, yes. So let's get to why one of these can be faster on the exact same silicon. Three mechanisms. First, kernel fusion and auto-tuning calibrated to the specific card. Second, CUDA graph capture, which kills CPU-side launch overhead. Five to ten microseconds per launch, which sounds like nothing until you're running at low batch and it's a real fraction of your step time.



Corn

And third?



Herman

Scheduler overhead. vLLM's Python scheduler costs you two hundred to five hundred microseconds per step at high concurrency. SGLang pushes more of that down into C++ and CUDA. At one request, nobody notices. At a few hundred concurrent, that's your throughput.



Corn

Okay. So they differ architecturally, and they differ for reasons you can name. Now the part I want, which is what happens when you load them.



Herman

The single most important finding in all of this. There is no universally fastest engine. The winner changes with the load.



Corn

That's the sentence, Herman.



Herman

One model, one H100, Llama 3.1 8B across all three. At batch one, the three of them are within four percent of each other. A hundred and fifty-eight tokens per second, a hundred and fifty-three, a hundred and fifty-two. Basically a tie.



Corn

Which is what you'd predict if the engine were plumbing.



Herman

And the moment you add concurrency, the tie breaks. At concurrency thirty-two, TensorRT-LLM leads on time to first token. Two hundred thirty-five milliseconds against vLLM's five hundred fourteen.



Corn

Same hardware.



Herman

Same hardware, same weights. Then you push to saturation and vLLM peaks highest. Five thousand three hundred thirty-three tokens per second against SGLang's five thousand two hundred thirty-five and TensorRT-LLM's four thousand eight hundred thirteen.



Corn

So the ranking flipped twice on one benchmark.



Herman

Flipped twice. And the L40S benchmark on Qwen3 32B from late August makes it sharper. At concurrency sixty-four, TensorRT-LLM ran thirty-seven percent higher throughput than vLLM. Three hundred forty-two tokens per second versus two hundred forty-five.



Corn

And then long context?



Herman

Long context is where it gets ugly. Eight thousand token inputs. TensorRT-LLM did double the throughput and finished time to first token in eight point six seconds. vLLM took sixty-three point eight seconds. SGLang sixty-eight point six.



Corn

Seven and a half times.



Herman

Seven and a half times, on identical weights, for the same prompt.



Corn

So if you're the customer and your provider happens to be the vLLM one, your long-document summarizer is unusable and you have no idea why.



Herman

And you can't change the engine. That's the scenario.



Corn

Now flip it. Who won the low-concurrency latency in that same suite?



Herman

vLLM. Eight hundred thirty-four milliseconds to first token at concurrency eight. SGLang two thousand twenty-nine, TensorRT-LLM two thousand five hundred forty-one.



Corn

So TensorRT-LLM is seven and a half times faster when you're full and eleven percent slower when you're quiet.



Herman

That's the actual shape of it. And SGLang quietly had the most consistent inter-token latency in the whole suite. Sixty point six to seventy-seven point eight milliseconds across every concurrency level.



Corn

The one that never wins the headline.



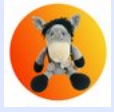
Herman

Never wins the headline, and it's the one that never falls over either. Which, for a product, is worth something.



Corn

Cost. Because that's the dimension a provider actually prices on.



Herman

RunInfra derived it from Modal's rates. H100 at three ninety-five an hour, L40S at one ninety-five. vLLM came out at fifteen point eight cents per million output tokens on fp8. SGLang seventeen cents. TensorRT-LLM twenty-two point eight cents on bf16.



Corn

And the L40S?



Herman

Roughly double the cost per token versus the H100. Hardware beats engine on cost.



Corn

Every time?



Herman

At that level, yes. But here's the one that made me sit up. Same test, quantization format mattered more than engine did. AWQ or GPTQ against FP8 on vLLM gave a forty-three percent throughput gain. Three hundred forty-nine against two hundred forty-four.



Corn

So you're benchmarking engines and the format underneath is the bigger lever.



Herman

Which is a humbling result if you build your whole purchasing decision on the engine logo. And then there's TensorRT-LLM's operational bill, which doesn't show up in tokens per second at all. Change the model, the quantization, the sequence length, or the GPU, and you rebuild the engine. Build times run from five minutes to over two hours.



Corn

So you need a CI pipeline and versioned artifacts.



Herman

You need a CI pipeline and versioned artifacts, and that's people, not GPUs. That's the bit that shows up in someone's week.



Corn

The one number I keep thinking about is that forty-three percent. You could do everything right on engine selection and lose more than that by picking the wrong weight format.



Herman

Right. The engine is the variable in the experiment, but it isn't always the biggest lever in the system.



Corn

Now the piece that surprised me and I suspect will surprise most people listening. Identical weights do not guarantee identical answers.



Herman

This is the thread I'd never have thought to pull. There's a 2026 paper out of CISPA. Holding weights, decoding settings, and hardware constant, and changing only the backend, benchmark scores moved by up to sixteen point six percentage points.



Corn

Sixteen point six points.



Herman

DeepSeek R1 Distill Qwen 7B on GSM8K. Seventy-eight point four seven percent on the transformers reference implementation. Sixty-one point eight seven percent on Ollama. Same weights.



Corn

What's driving the gap? If the weights are the same, something's changing them.



Herman

Two categories, and they're very different in character. The first is correctable engine defaults. Ollama force-prepends a BOS token, and applies a hidden repetition penalty of one point one. That penalty alone was worth eleven point six seven points on that benchmark. Turn it off, and it comes back.



Corn

So most of the headline is a default nobody documented.



Herman

Most of the headline is a default nobody documented, and that's the honest read. The paper surveyed two hundred distinct inference engines, and thirty-five thousand machine learning papers, and found the inference stack is almost never reported.



Corn

The layer that changes the answers is the layer nobody writes down.



Herman

That's the finding. The second category is optimization-induced numerical drift. Prefix caching, CUDA graphs, kernel tie-breaking, FP32 versus FP16 accumulation. No single layer is broken. The drift compounds as you go deeper into the network.



Corn

And that half is much smaller?



Herman

Much smaller. Sub-one-point in most ablations. On L40 cards the total spread was actually slightly wider, up to seventeen point two percent. And jailbreak vulnerability of DeepSeek R1 moved eight point nine points by engine alone.



Corn

The safety number moved too.



Herman

There's a second study out of Linz that crossed three models, five frameworks, six benchmarks, four generation modes. Even under greedy decoding, backend accounts for roughly thirty-nine percent of the variance a practitioner sees out of the box.



Corn

Out of the box being the important qualifier.



Herman

Out of the box, yes. Because a lot of that is the defaults again. Per-item disagreement hit forty-eight percent on StereoSet and forty percent on TruthfulQA for Llama 3.2 1B. And the effects were more pronounced on factual benchmarks than on bias ones.



Corn

Which is backwards from what I'd have guessed. You'd expect the vibes-y stuff to be fragile and the factual stuff to be stable.



Herman

The opposite. Accuracy questions have a narrow band of right answers, so a small nudge in the logits flips the answer. Bias questions have a lot of equally plausible answers, so the nudge doesn't change much.



Corn

That actually makes sense. Now let's do determinism, because that's the part I've been waiting for.



Herman

Thinking Machines Lab, Horace He, September 2025. The popular explanation is that concurrency plus floating point makes inference nondeterministic. That explanation is incomplete. The forward pass has no atomic adds in it, and it's run-to-run deterministic. What it isn't is batch-invariant.



Corn

Say that in English.



Herman

The same prompt produces different logits depending on how big the batch around it is. And batch size changes with how many other people are using the server, which is weather. So you send the identical request twice and get different numbers out.



Corn

How different?



Herman

They sampled Qwen3 235B a thousand times at temperature zero on "Tell me about Richard Feynman." Eighty unique completions.



Corn

Out of a thousand, at temperature zero.



Herman

Eighty. And they were identical for the first hundred and two tokens, then diverged. One run said Queens, New York. Another said New York City.



Corn

Temperature zero is not deterministic. That's worth putting on a poster.



Herman

It's the single most quoted line about inference that people get wrong.



Corn

So what does the fix cost? Because there's always a fix and there's always a cost.



Herman

Sixty-one and a half percent slower on Thinking Machines' original batch-invariant kernels. SGLang shipped it in September of 2025 and got the average slowdown down to thirty-four point three five percent. Range of twenty-four point four to forty-six percent depending on the workload. CUDA graphs claw back about two point eight times on top of that.



Corn

A third of your throughput for a guarantee.



Herman

A third of your throughput. vLLM shipped the flag for it, and there's a footnote: async scheduling is on by default and can still break invariance. And fused MoE kernels are still non-invariant in some open issues.



Corn

So the switch exists and doesn't fully deliver yet.



Herman

Exists and doesn't fully deliver. Which is honestly where a lot of this sits.



Corn

Deployment-level reproducibility. This is the study that made me want to do the episode.



Herman

Priyansh Bhatter, vLLM 0.25.0, Qwen3.5 9B, GSM8K. Fresh vLLM deployments matched only fifty-nine to sixty-four percent of outputs. Identical weights, identical GPU, identical config.



Corn

Not close to identical.



Herman

And the obvious suspect was batching, so they capped batch at one, eight, and thirty-two, and the three behaved identically. Batching was not the cause. Then they took two containers and had them share the same compiled autotune cache.



Corn

And?



Herman

Thirteen hundred nineteen outputs. Thirteen hundred nineteen identical matches, across different H100s.



Corn

So the variable is the autotune state.



Herman

The compile and autotune state, not the sampler. And the FP8 W8A8 path had zero divergences across thirteen thousand one hundred ninety comparisons, which is its own quiet finding about quantization.



Corn

If I'm a developer reading that, the interesting bit is the accuracy held.



Herman

Accuracy held within zero point three five points. Verdict flips about one percent of the time. But the text churn is around forty percent. So the answer is usually the same and almost never the same words.



Corn

Which is fine for chat and fatal for an audit trail.



Herman

Fatal for an audit trail. And that's the split I'd draw for Daniel. Two effects, two completely different remedies. The performance differences are large but visible. You benchmark your own workload and you know. The quality differences are occasionally double digits in the lab, usually sub-one-point in practice, and completely invisible unless you go looking.



Corn

And they compound on long chains of reasoning. Which is where people are actually deploying these now.



Herman

Where they're actually deploying them. That's the bit that should worry a team that runs evals.



Corn

Here's what I keep coming back to. Daniel asks how much attention an ordinary developer should pay to the engine, and the honest answer is less than you'd think, unless you're doing one of a few specific things.



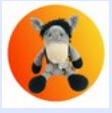
Herman

The work where the answers have to be the same every time.



Corn

Evals, reinforcement learning rollouts, anything regulated or auditable. If your job is to produce a number that someone else can reproduce, the engine is not a footnote. It's the thing.



Herman

And if you're shipping a chat product, then hardware, price, and whether the provider stays up will swamp it.



Corn

But there's a catch, which is the piece that makes the whole thought experiment slightly unreal.



Herman

Which is that the customer can't see it.



Corn

Two hundred distinct engines in the survey. The stack is rarely reported in papers. So a provider dashboard says "H100, eight billion parameters, four dollars a million tokens," and it does not say vLLM.



Herman

Most don't say anything.



Corn

So the advice "benchmark your own workload" runs into the fact that you often can't tell which engine you're benchmarking.



Herman

You can measure the outcome. You can't measure the cause.



Corn

The deployment gap. Research validates on one stack and ships on another, and the stack isn't recorded on either end.



Herman

Which is exactly why Daniel's framing is good. Hold the weights constant, hold the hardware constant, and the thing you can't see is the thing that moved the number.



Corn

Herman. One thing I want to put to you, since you're the one who reads the papers. Is the engine becoming less of a differentiator over time?



Herman

I think so, on the performance axis. The features converged. TensorRT-LLM gave up the compiled backend. And every difference we cited is operating-point-dependent, which means it's the kind of difference a provider can engineer around once they know it matters.



Corn

So the axis moves.



Herman

It moves to determinism guarantees and operational simplicity. Which are the two things you cannot fake with a benchmark chart.



Corn

The benchmark chart is exactly what can't be faked.

SOURCES

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

1. **Thinking Machines Lab (Horace He), 2025-09-10 [primary]**
https://thinkingmachines.ai/blog/defeating-nondeterminism-in-llm-inference/

2. **SGLang Team, 2025-09-22 (updated 2025-09-24) [primary]**
https://www.lmsys.org/blog/2025-09-22-sglang-deterministic/

3. **vLLM repo [primary]**
https://github.com/vllm-project/vllm

4. **SGLang repo (confirms DeepSeek V4 Flash support) [primary]**
https://github.com/sgl-project/sglang

5. **Inference Engineering, updated 2026-06-01 (architecture deep-dive)**
https://inferenceengineering.tech/learn/vllm-vs-sglang-vs-tensorrt-llm/

6. **RunInfra, 2026-06-20, updated 2026-09-21 (vLLM 0.23.0, SGLang 0.5.13, TRT-LLM 1.2.1)**
https://runinfra.ai/news/vllm-vs-sglang-vs-tensorrt-llm-benchmark

7. **Michał Wojdylak, 2026-08-28 (L40S/Qwen3-32B, AIPerf v0.11.0)**
https://michalwojdylak.com/blog/llm-inference-benchmarks-vllm-sglang-tensorrt-llm

8. **and https://arxiv.org/html/2605.19537v2**
The Silent Hyperparameter, Pape/Evertz/Schönherr (CISPA), v2 2026-05-20
https://arxiv.org/abs/2605.19537

9. **What We Observe as LLM Behavior Can Be a Side-effect of Inference Backend, Masoudian et al. (JKU Linz), 2026**
https://arxiv.org/html/2608.04714

10. **Spheron, 2026 (VLLM_BATCH_INVARIANT, pricing as of 2026-10-12)**
https://www.spheron.network/blog/vllm-batch-invariance-why-temperature-0-isnt-deterministic/

11. **Priyansh Bhatter, 2026 (vLLM 0.25.0, Qwen3.5-9B, GSM8K)**
https://www.priyanshbhatter.com/research/deployment-reproducibility

12. **SGLang/RadixAttention paper (Zheng et al., NeurIPS 2024)**
https://arxiv.org/abs/2312.07104