# Episode Book

EPISODE #5900

# Serving DeepSeek-V4.1-Flash on Your Own GPU

| | |
|---|---|
| PUBLISHED | 12 October 2026 at 12:21 UTC |
| DURATION | 22:34 |
| LISTEN | myweirdprompts.com/episode/deepseek-v4-flash-local-inference-87980/ |
| 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

You've got the weights, you've got the rack, you've got a GPU bolted in that you're not replacing. So which engines can actually serve DeepSeek-V4.1-Flash — and how much of your "engine choice" is really a hardware decision wearing a software costume? This episode walks the five-name shortlist (vLLM, SGLang, TensorRT-LLM, llama.cpp, ONNX Runtime) against the model card's supported paths, then runs the two-server thought experiment: one NVIDIA box, one AMD box, same budget line, wildly different menus. Along the way: why the most portable engine is the one that can't load the model, why "supports AMD" and "supports your AMD card" are two different sentences, and why the quantization method printed on the checkpoint can decide whether your inference runs at all.

# TRANSCRIPT



**Corn**

Last time we ran the thought experiment from the outside. Three commercial providers, identical DeepSeek Flash 4 weights, identical GPUs, different engines underneath, and we asked whether that choice changes anything you can feel.



**Herman**

And this time the server is ours.



**Corn**

Daniel wants to take it local. Our own Linux box, our own rack, our own engine selection. Which sounds like freedom until you remember the GPU is already sitting there, bolted in, and cards are not free. So the question he's really asking is how much of that freedom is real. Which engines can even run this model, and is choosing an engine a software preference or a hardware decision wearing a software costume.



**Herman**

Two threads. Who supports the model, and who supports your card.



**Corn**

And before we compare anything, we should be honest about what we're actually deploying, because we've been calling it DeepSeek Flash 4 and that name doesn't exist.



**Herman**

It doesn't. There's no product called Flash 4. The family is DeepSeek-V4-Flash — two hundred eighty-four billion parameters total, thirteen billion active, mixture of experts, previewed back in April, official release at the end of July.



**Corn**

And that one's already been superseded.



**Herman**

By V4.1-Flash, September tenth. Five hundred fifty-two billion in the backbone, but only eight billion active per token during prefill and sixteen during decode. Multimodal. A million-token context. FP4 main KV caching, which gets it to eight hundred ninety bytes per token of cache — roughly a quarter of what V4-Flash needed, and about four hundred thirty-seven times smaller than the original DeepSeek-V1 cache.



**Corn**

So it got bigger and cheaper at the same time.



**Herman**

That's the whole trick of these MoE designs. The parameter count is the number on the poster, but what you actually pay for at inference is the active path and the cache. And there's a speculative decoding module bundled with it, DSpark. This is not a small dense model you casually serve on a spare box. It's a frontier mixture of experts that happens to have been open-weighted.



**Corn**

Which brings the shortlist into focus.



**Herman**

vLLM. SGLang. TensorRT-LLM. llama.cpp. ONNX Runtime. Five names people throw around as if they're five equivalent options, and they are not remotely equivalent.



**Corn**

Here's the tension I want to keep in view for the whole episode. The engines that support a brand-new frontier MoE are exactly the ones that ship vendor-specific builds. And the most portable engine of the five is the one least likely to run this model at all.



**Herman**

That's the shape of it. So let's start with who can actually load the weights today.



**Corn**

The model card lists the ready local paths, and it's a short list. vLLM, SGLang, and Docker Model Runner. Copy-paste serve commands right there on the page.



**Herman**

vLLM serve, sglang launch server. And TensorRT-LLM merged support on October eleventh, pull request nineteen seven two seven. So as of yesterday, four engines can run it.



**Corn**

Yesterday. That's the level of freshness we're dealing with.



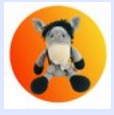
**Herman**

And the two that aren't on the list are llama.cpp and ONNX Runtime. Neither is listed as a supported path for V4.1-Flash. Both would need real implementation work.



**Corn**

Not "flip a flag" work.



**Herman**

No. llama.cpp's whole ecosystem is GGUF and its quantization formats, and its architecture coverage lags on brand-new mixture-of-experts designs. ONNX Runtime consumes ONNX graphs, and the conversion path for a model this new, with this many custom kernels, isn't there yet. It's not that nobody could do it. It's that nobody has, and until they do, you're on your own.



**Corn**

There's an escape hatch though.



**Herman**

There is, and it's interesting. DeepSeek ships a self-contained Python reference implementation in the inference folder of the repo. Model dot py, kernel dot py, generate dot py, convert dot py. Plus a set of Rust libraries they call the deepseek recipe. So you could, in principle, build your own serving path from the vendor's own code.



**Corn**

In principle.



**Herman**

Exists is not the same as supported. A reference implementation is written to be readable, not to serve traffic. It doesn't batch for you. It doesn't have a paged attention scheme. It's a teaching artifact that happens to run.



**Corn**

It's the difference between a recipe and a restaurant.



**Herman**

That's exactly what it is. And I want to come back to that escape hatch at the end, because I think it matters more than it looks.



**Corn**

Now the architecture question, because Daniel asked it precisely. ONNX Runtime is the clean case of one engine with interchangeable backends. Explain how that actually works.



**Herman**

Execution Providers. One runtime, one API, and underneath it a set of hardware-specific backends you can swap. CUDA, TensorRT, DirectML, MIGraphX from AMD, OpenVINO from Intel, QNN for Qualcomm, WebGPU. You list them in priority order and the runtime picks the first one available. CUDA first, CPU as the fallback, something like that.



**Corn**

So the model doesn't care which one it landed on.



**Herman**

The graph is the same graph. The execution provider handles the hardware-specific libraries underneath it. That's the abstraction everyone says they want — write once, run on whatever's in the box.



**Corn**

And it works.



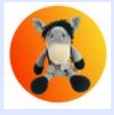
**Herman**

It works, within the ONNX format. That's the honest version. It's the closest thing the ecosystem has to the interchangeable execution layer idea.



**Corn**

Which makes what follows funnier. Compare that to vLLM.



**Herman**

vLLM is one codebase with vendor-specific build paths inside it. They compile separate CUDA kernels and separate ROCm kernels, and they ship distinct container images. vllm slash vllm-openai for NVIDIA, vllm slash vllm-openai-rocm for AMD. Same project, two builds.



**Corn**

Different picture from ONNX Runtime though. There, one artifact runs anywhere. Here, the artifact is chosen for you by your card.



**Herman**

Right, and SGLang does the same thing. There's an amd-sglang wheel, there's a separate ROCm Dockerfile. It's one project maintaining parallel hardware paths rather than one runtime dispatching at load time.



**Corn**

And then TensorRT-LLM isn't even in that category.



**Herman**

TensorRT-LLM isn't a portable engine at all. It compiles the model into a TensorRT engine at build time, specific to the GPU architecture you're targeting. You're not loading weights and dispatching kernels. You're producing a compiled artifact for one class of hardware.



**Corn**

So you don't pick TensorRT-LLM and then pick your GPU. You pick your GPU and then TensorRT-LLM builds for it.



**Herman**

And that's the trade. You give up flexibility you might want later and you get performance you can't get any other way. That's the deal on the tin, and it's a real deal.



**Corn**

Here's the irony I can't get past. The engine that best delivers the interchangeable-backend ideal is the one least likely to run DeepSeek-V4.1-Flash.



**Herman**

ONNX Runtime is the cleanest implementation of the portable design, and it's the one that can't load the model today. Meanwhile vLLM and SGLang, which do load it, are the ones maintaining separate kernel builds per vendor. Portability and cutting-edge model support pulled in opposite directions.



**Corn**

So the abstraction isn't free. It's paid for in latency to new architectures.



**Herman**

Every layer of indirection costs you something, and what it costs is the time between a model shipping and the runtime being able to execute it well.



**Corn**

That's who supports the model. Now the harder question, and the one Daniel actually cares about. Who supports your GPU.



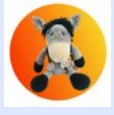
**Herman**

Two otherwise comparable Linux servers. One NVIDIA, one AMD. Same rack, same power, same budget line. And the shortlist of engines you can run collapses differently on each.



**Corn**

Start with the one that isn't a comparison at all.



**Herman**

TensorRT-LLM is NVIDIA-only, full stop. The support matrix is Blackwell, Hopper, Ada Lovelace, Ampere, Grace Hopper, and GB200 NVL72. Anything not on that list gets community support only. There is no AMD path, no TPU path, no Trainium path. Not a weak path — none exists, and none is on a public roadmap.



**Corn**

So on the AMD box, that entire engine is gone before we've discussed anything else.



**Herman**

Gone. It's a hardware decision that presents itself as a software decision. You think you're choosing an engine and you're actually choosing a vendor.



**Corn**

vLLM is the broadest of the ones that matter here.



**Herman**

vLLM supports NVIDIA CUDA at compute capability seven point five and up, AMD ROCm, and Intel XPU. On the AMD side the list is MI200s, MI300, MI350, the Radeon RX 7900 series, the RX 9000 series, and the Ryzen AI MAX parts. ROCm six point three or newer, with MI350 needing ROCm seven. Prebuilt ROCm wheels for seven point zero and seven point two point one. No native Windows support, which matters if your "Linux server" is actually a Windows box with a Linux VM bolted on.



**Corn**

That's a real list. Not a gesture at AMD.



**Herman**

And it's not just datacenter silicon either. vLLM supports consumer Radeon cards. Hold that thought, because SGLang doesn't.



**Corn**

SGLang's list is longer on paper. NVIDIA, AMD, Google TPU, Intel, Apple Silicon, Huawei Ascend, Moore Threads. Sounds like the most portable engine going.



**Herman**

Until you look at what "AMD" covers. It's datacenter Instinct only. MI300X, MI325X, MI350X, MI355X. There's a tracking issue, number three zero five nine nine, that states it plainly. SGLang's ROCm path is datacenter-Instinct-only, while every direct competitor already supports consumer cards. vLLM, llama.cpp, Ollama, LM Studio, KoboldCpp.



**Corn**

So "supports AMD" and "supports your AMD card" are two different sentences.



**Herman**

They're two different sentences, and the gap between them is where most of the disappointment lives. If you bought a Radeon because it was on the shelf and reasonably priced, SGLang is not for you, and no amount of reading the hardware table will tell you that unless you read the tracking issue.



**Corn**

And llama.cpp, the portability champion, supports basically everything.



**Herman**

CUDA, HIP for AMD, MUSA for Moore Threads, Metal for Apple, SYCL for Intel, Vulkan, WebGPU, CANN for Ascend, plus plain CPU. It is the most hardware-agnostic engine in the field by a wide margin. One hundred thirty thousand stars, MIT licensed.



**Corn**

And it can't run the model.



**Herman**

That's the trade in one sentence. The most portable engine is the one that can't do the thing we're here to do.



**Corn**

Which leaves the two-server thought experiment looking grim on the AMD side.



**Herman**

On the NVIDIA box you get vLLM, SGLang, TensorRT-LLM, and llama.cpp. Four engines, all viable, real choice. On the AMD box TensorRT-LLM is gone, SGLang is gone unless you're on datacenter Instinct, and llama.cpp can't load the model anyway. You're down to vLLM, and that's it.



**Corn**

One engine. That's not a shortlist.



**Herman**

That's a decision already made for you by a purchase you made two years ago.



**Corn**

There's something under this that nobody puts on the comparison chart. Quantization kernels.



**Herman**

This is the part that surprises people who think of quantization as a neutral knob you turn for speed. SGLang's own AMD documentation lists what works on AMD: FP8, AWQ, MXFP4, W8A8, GPTQ, compressed-tensors, Quark, petit nvfp4. And then it lists what doesn't. Anything that depends on Marlin kernels or NVIDIA-specific implementations. awq_marlin, gptq_marlin, gguf, modelopt fp8, modelopt fp4.



**Corn**

So the method name printed on the checkpoint determines whether it runs at all.



**Herman**

It determines whether it runs, and if it runs, how fast. AWQ on AMD falls back to slower Triton dequantization kernels. Same nominal quantization scheme, materially slower execution, because the fast kernel was written for the other vendor.



**Corn**

And you find that out after you've downloaded the checkpoint.



**Herman**

After you've downloaded it, converted it, and hit the wall. That's a vendor lock that doesn't announce itself. The engine says AMD supported. The checkpoint says AWQ. The fast path says NVIDIA only. Three green ticks that add up to a slow run.



**Corn**

So the constraint operates at three levels, and I want them said clearly because Daniel asked whether this is preference or constraint. Level one, engine availability. Which engines exist on your card at all.



**Herman**

Level two, quantization kernel availability. Which checkpoints run at full speed once the engine loads.



**Corn**

Level three, build-time compilation. Whether your engine has to be compiled for your specific architecture before it runs anything.



**Herman**

And TensorRT-LLM is level three in its purest form. You don't ship it a model. You ship it a model and an architecture and it produces an engine file that works on that combination and nowhere else.



**Corn**

Which is why the performance numbers have to be read carefully.



**Herman**

RunInfra's benchmark is the honest illustration. TensorRT-LLM has the lowest first-token latency at moderate load. Two hundred thirty-five milliseconds versus vLLM's five hundred fourteen at concurrency thirty-two. That's a genuine, noticeable win for interactive use.



**Corn**

And at saturation?



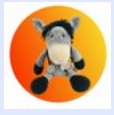
**Herman**

vLLM wins throughput. Five thousand three hundred thirty-three tokens per second on a single H100, and the lowest derived cost, fifteen point eight cents per million output tokens on FP8. Which tells you there's no single fastest engine. The winner changes with the operating point.



**Corn**

Fast at low load, cheapest at high load, and those are different engines.



**Herman**

And neither number transfers to the AMD box, because the fast paths don't exist there. The benchmark is real and it's also conditional.



**Corn**

So Daniel's framing question. Is choosing an engine primarily a software preference?



**Herman**

No. Not primarily. It's constrained by the GPU you own, and it's constrained at all three of those levels at once.



**Corn**

I'd go further. The order of operations is backwards. You don't pick an engine and then find hardware. The hardware is already in the rack when the question arrives. You're not choosing, you're filtering.



**Herman**

And the filter runs in the order Daniel guessed. Model architecture first, because that eliminates engines outright. Hardware second, because that eliminates more. Workload third, because workload is what picks between the survivors.



**Corn**

That's the answer to his last question too. How much difference does the choice make on the same machine?



**Herman**

On the same machine, with the same weights, the difference between the fastest and slowest viable engine is a multiple, not a percentage. But the machine you're on decides which engines are even in that comparison.



**Corn**

Here's what I keep coming back to. He asked whether it's software preference or hardware constraint, and the honest answer is that those two things got fused years before he asked. The preference is real, but it's expressed within a box that somebody else's purchase order drew.



**Herman**

You know, we did an episode a while back about local speech-to-text for people who'd bought AMD cards.



**Corn**

And the shape of that conversation was identical. Same constraint, different workload. The hardware you own narrows the menu, and the narrowing is invisible until you try to order.



**Herman**

Which is why the consumer-Radeon gap is the case study I'd keep. It's not that SGLang is bad on AMD. It's that SGLang is excellent on MI300X and absent on the card a normal person bought. One engine, two truths, and the documentation's top line says AMD.



**Corn**

Let me guess what my brother reaches for here.



**Herman**

Go on.



**Corn**

The escape hatch. The reference implementation. You've been circling it since we mentioned it.



**Herman**

I have, because I think it's the only interesting future in this whole landscape. Every layer of portability we've discussed is a vendor's abstraction, and vendors have no incentive to make their abstraction run on a competitor's silicon. The reference implementation is the one path that belongs to nobody.



**Corn**

And runs at nobody's speed.



**Herman**

Today it does. But a reference implementation is the seed of a portable runtime, and someone building for the AMD box has a reason to water it that NVIDIA never will. If the MoE era keeps going the way it's going, that's the path with the most room in it.



**Corn**

I'll take the pessimistic side. A from-scratch implementation has to keep pace with a model family that ships every few weeks. It's one release behind on the day it's written.



**Herman**

That's the race. I don't know who wins it, and I'd rather say so than pretend. But I don't think it's nothing, either.



**Corn**

Agreed, and I'll leave it at that. The hardware story is where this gets uncomfortable, and I think Daniel already knows why.



**Hilbert**

They're right, mostly.



**Herman**

Mostly.



**Hilbert**

The card decides. I don't argue with that. But the card isn't what stopped me. The README stopped me. Weekend before last, I had a card sitting in a machine doing nothing, and I decided it was going to run a model. Something smaller. It should have taken an evening.



**Corn**

And it didn't.



**Hilbert**

It took two days and eleven hours. I kept a note of it, because I wanted the number. Eleven hours of it was one error message, and the message was wrong. The README said the version in it was the version that worked. It was not the version that worked. It was the version before the one that worked, and there was a one-line note in a changelog, which I found at hour nine, that said so. I had been installing and uninstalling since Friday.



**Herman**

The changelog note.



**Hilbert**

It said the kernel had been renamed. That's all. One word had changed, and the documentation hadn't. I fixed it in about four minutes once I knew, and then I sat there for another two hours because I was annoyed.



**Corn**

The vendor was fine and the paperwork lied.



**Hilbert**

The hardware was fine the whole time. I want that on the record. The card did everything I asked. The instructions didn't.



**Herman**

What did you do with it once it ran?



**Hilbert**

I ran it four times and turned it off. What I actually want it for is something else. I've got a man who does my accounts, and I want to know whether the thing in this episode would let me point the whole setup at a shoebox of receipts and have it sort them. Sort them. Into piles. By hand it's a week, and I've been avoiding it since March.



**Corn**

You're not going to say why the receipts need sorting.



**Hilbert**

No.



**Herman**

Would it do it? The model.



**Hilbert**

Multimodal, million-token context, a graph you can hand a picture. On paper it'll do it. On paper it would have taken me an evening, too. So before I spend three thousand on a card to find out, I want to know whether the receipt pile is going to run, or whether it's going to be hour nine all over again, except this time the changelog is in a language I don't read.



**Corn**

That's the whole episode in one sentence, and I hate that it's him saying it.



**Hilbert**

I still have the card. It's in the machine. It's not doing anything, but it's in the machine.



**Corn**

The straight answer to Daniel, then. On our own Linux server, the freedom he's describing is real, and it's smaller than it looks. You get to pick the engine, roughly, within the set your card permits, and the set is not the same on the two servers he imagined.



**Herman**

Which leaves me with the question I don't have an answer to. If mixture-of-experts and FP4 cache are going to be standard for the next two years, does the portability gap widen or narrow? Every new architecture adds more custom kernels, and custom kernels are the least portable thing in software.



**Corn**

Which means the vendors with the most incentive to write them pull further ahead, and the portable runtimes fall further behind on day-one support.



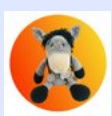
**Herman**

Unless the reference implementation quietly becomes a real path, which would be the interesting surprise.



**Corn**

Here's the uncomfortable part that I think Daniel's question surfaces without meaning to. The interchangeable execution layer sounded clean when we were watching three providers argue about it. It looks a lot different when you're the one buying the GPU.



**Herman**

When it's your purchase order drawing the box.



**Corn**

Because then portability isn't a design property. It's a line item you didn't know you were paying for.



**Herman**

That's the episode. Thanks as ever to our producer Hilbert Flumingtop, who is now apparently pricing graphics cards in his head.



**Corn**

If you enjoyed this one, try episode two, Local STT For AMD GPU Owners; episode three, Safetensors or something else; and episode eighteen, Beyond the GPU. This has been My Weird Prompts.



**Herman**

If you want to send us a prompt of your own, the bot is on Telegram, t dot me slash MWP listener bot. And a review wherever you listen helps people find the show.



**Corn**

We'll be back soon.

## SOURCES

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

1. **DeepSeek API Change Log (2026-09-10 V4.1-Flash release; full history) [primary]**
   https://api-docs.deepseek.com/updates

2. **DeepSeek-V4.1-Flash model card (architecture, benchmarks, local-app paths) [primary]**
   https://huggingface.co/deepseek-ai/DeepSeek-V4.1-Flash

3. **DeepSeek-V4-Flash model card [primary]**
   https://huggingface.co/deepseek-ai/DeepSeek-V4-Flash

4. **DeepSeek reference inference implementation [primary]**
   https://huggingface.co/deepseek-ai/DeepSeek-V4.1-Flash/tree/main/inference

5. **vLLM GPU install docs (CUDA/ROCm/XPU support matrix) [primary]**
https://docs.vllm.ai/en/v0.21.0/getting_started/installation/gpu/

6. **SGLang AMD GPU docs (quantization matrix) [primary]**
https://docs.sglang.io/docs/hardware-platforms/amd_gpu

7. **SGLang repo README (supported hardware table) [primary]**
https://github.com/sgl-project/sglang

8. **TensorRT-LLM repo (DeepSeek-V4.1-Flash support commit, 2026-10-11) [primary]**
https://github.com/NVIDIA/TensorRT-LLM

9. **llama.cpp repo README (backend table) [primary]**
https://github.com/ggml-org/llama.cpp

10. **ONNX Runtime Execution Providers docs [primary]**
https://onnxruntime.ai/docs/execution-providers/

11. **SGLang consumer Radeon support tracking issue**
https://github.com/sgl-project/sglang/issues/30599

12. **vLLM/SGLang/TensorRT-LLM benchmark**
https://runinfra.ai/news/vllm-vs-sglang-vs-tensorrt-llm-benchmark

13. **engine comparison (TensorRT-LLM NVIDIA-only)**
https://ai-solutions.wiki/comparisons/vllm-vs-tensorrt-llm-vs-sglang/

14. **HN comment on ROCm vs CUDA maturity (2024)**
https://news.ycombinator.com/item?id=39767893