Episode Book

EPISODE #5963

Running One Model Across Mismatched GPUs

PUBLISHED	14 October 2026 at 14:47 UTC
DURATION	31:32
LISTEN	myweirdprompts.com/episode/multi-gpu-model-parallelism-83583/
HOSTS	Corn, Herman, Hilbert
SCRIPT MODEL	deepseek-flash
VOICES	chatterbox-regular (chatterbox-local)
PIPELINE	V5.4
SOURCES	11 cited by the research (listed at the end)
BOOK GENERATED	19 October 2026 at 17:42 UTC

SYNOPSIS

You own two GPUs, so you bolt them together and expect the model to spread itself across both. It almost never works that way. This episode breaks down the three real techniques for running one open-weight model across multiple cards on a single Linux box: tensor parallelism, pipeline parallelism, and offloading. Each one is a different trade between capacity, speed, and the cost of cards talking to each other — and the interconnect, not the compute, sets that price. We walk through what vLLM and llama.cpp actually support, why tensor mode is still experimental, why attention head counts can veto your card count, and what the documentation quietly admits about mismatched hardware. If you're speccing a new multi-GPU server or just adding a second card to an existing machine, this is the honest version of whether that second card is worth it.

TRANSCRIPT



Corn

There's a thing people do with graphics cards that almost never works the way they expect, and it's worth an episode, because the moment you own two GPUs the temptation to bolt them together is enormous.



Herman

And the hardware makes it look so easy. Two slots, two cards, one machine.



Corn

Daniel wants to know what it actually takes to run one open-weight model, say a DeepSeek, across several GPUs on a single Linux box. Both versions of it. The scrappy path where you've got a card in there already and you add a second one that doesn't match it, maybe different memory, different generation, even a different vendor. And the deliberate path where you're speccing a new server and you buy several identical cards on purpose.



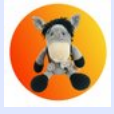
Herman

And he's asking about the underlying techniques, the tooling, and the honest version of whether that second card is ever worth it. Which is a better question than it sounds, because the answer is sometimes no.



Corn

That's the tension. Capacity and speed look like the same thing when you're shopping, and they aren't. So let's start with the vocabulary, because everything else hangs off it.



Herman

Right. The first distinction, and it's the one people skip, is between running one model across several GPUs and running several copies of a model on several GPUs. Data parallelism is the second one. Each card holds a complete replica, there's no talking between them, and they just serve different requests. That raises throughput if you have concurrent traffic. It does not pool memory, and it does nothing for latency.



Corn

A queue of requests, one card each.



Herman

Model parallelism is the other thing entirely. You split one model, the cards cooperate on every token, and now you can fit something that doesn't fit anywhere. That's the mode Daniel is asking about, and it's the mode with all the sharp edges.



Corn

So the whole episode is: pooling memory is possible, and it is not free.



Herman

Every technique is a trade. You're buying capacity or you're buying speed, and the price is paid in communication between the cards. What governs that price is the interconnect, not the compute. That's the theme, and I'll be honest, it took me a while to internalize it properly.



Corn

What does that mean for the person who just wants to add a card.



Herman

It means there's no magic. Nothing in the stack looks at your two mismatched cards and transparently spreads a model across them by free memory. Pooling is always something you configure explicitly, or something a specific backend gives you as a side effect. That's the honest starting point.



Corn

Alright. Three techniques, then. Which is the one people reach for first?



Herman

Tensor parallelism, usually, because it sounds like the good one. And it is the good one for latency, when your hardware supports it. The mechanism is this. You take each layer's weight matrices and you shard them across the GPUs. Every card works on the same computation for the same token, but on a slice of it, and then after each layer you have to synchronize, because every card needs the partial results from the others to feed the next layer. That synchronization is an AllReduce.



Corn

After every layer.



Herman

After every single layer. So the cards are chatting constantly. It minimizes latency, because you're using all the hardware on every token, but it is completely at the mercy of how fast those cards can talk to each other. If the link is slow, the AllReduce is slow, and every layer waits.



Corn

So it's a fast highway or it's nothing.



Herman

That's what vLLM's own guidance says, essentially. Use tensor parallelism when the model is too large for a single GPU but fits on a single node with multiple GPUs. Fits on a node is load-bearing there.



Corn

And there's a constraint beyond the link, isn't there. Something about heads.



Herman

Attention heads. Tensor parallelism has to divide the attention heads evenly across the cards. If your model has sixty-four heads and you try to run tensor parallel size of three, vLLM doesn't degrade gracefully, it errors and tells you the sixty-four heads must be divisible by three. Which they aren't.



Corn

So how many cards you can use isn't up to you, it's up to the model's head count.



Herman

Up to the model's head count. That's a annoying thing to discover after you've bought the third card.



Corn

What's the fallback, then.



Herman

Pipeline parallelism. Same idea, much gentler on the wiring. Instead of slicing each layer, you split the model by depth. Card one holds layers one through twenty, card two holds twenty-one through forty, the rest on card three. A token goes in at the front, comes out at the back, and the cards pass activations along the chain. It's an assembly line.



Corn

And the traffic is tiny by comparison.



Herman

Tiny. You're passing one batch of activations per stage boundary, not doing a collective operation per layer. So it tolerates a slow interconnect, and it handles uneven splits, which matters enormously if your cards aren't identical. vLLM's advice is to use pipeline parallelism instead of tensor parallelism for higher throughput and lower communication overhead when you don't have NVLink. They specifically call out the L40S as a card to



Herman (continued)

pipeline rather than tensor.



Corn

But latency.



Herman

Latency is worse, yes, because a single request has to walk through every stage in sequence. Card two is idle while card one works on that one request, and so on down the line. The only way you get throughput out of a pipeline is if it's full, which means lots of concurrent requests, each one at a different stage. One user, one request, and you're watching three cards take turns.



Corn

So it's like a car factory. One car on the line at a time and you've built a very expensive way to be slow.



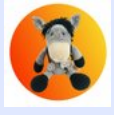
Herman

Exactly that. The line is only worth it when there are cars on every station.



Corn

Then there's the third one, offloading. Which is what most people actually end up doing, I suspect, whether they call it that or not.



Herman

Most people, yes. Offloading is not parallelism at all, in the sense that the cards aren't cooperating on the same computation. You keep some of the weights in system RAM, or on a slower slot, and you only route the rest to the fast card. In llama.cpp you do it with a couple of flags. There's one that keeps the expert weights of the first N layers in RAM, and there's a general one that lets you route specific tensors to a named buffer by pattern matching. You're saying: this tensor lives on the CPU, this one lives on the first card, this



Herman (continued)

one on the second.



Corn

And the point of it is purely capacity.



Herman

Purely capacity. Offloading lets a model run that otherwise would not fit at all. It doesn't make anything faster. What you're doing is trading speed for the ability to run the thing, and the speed you're trading is the bandwidth of wherever you parked those weights. If they're in system RAM, every token has to pull them across the bus.



Corn

Now, llama.cpp. It's the tool most people on this show's audience would actually reach for, and its options map almost one to one onto what we just described.



Herman

They do, and it's cleaner than you'd expect. There's a split mode flag. Set it to none and you use a single card. Leave it on layer, which is the default, and it does pipeline parallelism. That's the most compatible mode, it tolerates slow interconnects, and it's the best option for prefill, which is that first pass where the model chews through your whole prompt.



Corn

And row, before you ask. Row is deprecated. Nobody should be using it.



Herman

Then there's tensor, which is the experimental one. It splits weights and the KV cache across cards and it's the fastest mode for token generation when it works, and the docs are blunt about it. They say treat it as experimental, the code is less mature than pipeline parallelism, performance should be good for multiple NVIDIA GPUs on the CUDA



Herman (continued)

backend, and no guarantees otherwise.



Herman

It's carrying the whole tent. And it comes with requirements. Tensor mode needs flash attention on, and it needs a non-quantized KV cache, so sixteen or thirty-two bit. Auto-fit isn't supported with it at all. And there's a list of architectures it simply isn't implemented for.



Corn

How long a list.



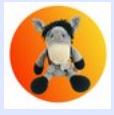
Herman

Long. DeepSeek2 is on it, which is directly relevant to Daniel's example. Most of the hybrid and mixture-of-experts architectures are on it, the state space models like Mamba. If you try it on one of those you get an error saying tensor split mode is not implemented for the architecture, and that's that.



Corn

So the model Daniel picked as his example may not even be able to use the fast mode.



Herman

On llama.cpp, quite possibly not. It would fall back to layer split, which does work, it's just not the fast path. That's a real thing to know before you start tuning.



Corn

And the tensor split flag itself, the one for proportions.



Herman

That's the one you use for mismatched cards, and it's the documented way to handle them. You give it a ratio. A ratio of three to one means the first card gets seventy-five percent and the second gets twenty-five. It's flexible about how you write it, three comma one or seventy-five comma twenty-five are the same thing, it normalizes.



Corn

So a big card and a small card is a supported configuration, not a hack.



Herman

Supported for layer mode and for tensor mode. Though read the docs on that carefully, because the behavior of the automatic split differs. If you omit the ratio, layer and row split proportionally to memory, which sounds good. Tensor splits evenly, which does not sound good if your second card has a third of the memory of your first.



Corn

And there's fine print about it not accounting for the KV cache, I think.



Herman

There is. The automatic proportional split looks at memory, but it doesn't fully account for the compute buffers and the KV cache each card needs, so with very heterogeneous cards, like a twenty-four gigabyte card and a six gigabyte card, the automatic split can hand the small card more than it can actually hold. There's work in the tracker aimed at making that distribution KV-cache-aware.



Corn

Before we leave llama.cpp, the interconnect layer underneath. That's where some of the sharpest gotchas live.



Herman

It is. There are three separate things there and they behave differently. NCCL is the cross-GPU communication library, and it's selected at build time. On CUDA it's on by default, and llama.cpp uses it automatically for those reductions in tensor mode. If you build without it you get a warning saying NCCL is unavailable, multi GPU performance will be suboptimal.



Corn

And the AMD equivalent.



Herman

RCCL, and it's the opposite story. It's opt-in, and the docs say why: it's disabled by default because unlike NCCL it wasn't universally beneficial during testing. That's a wonderfully honest sentence to find in a build file.



Corn

It is. Then peer-to-peer.



Herman

CUDA peer-to-peer is also opt-in. It needs driver support, which generally means workstation or data center cards rather than consumer ones, and there's a warning attached that it can cause crashes or corrupted outputs on some motherboards or BIOS configurations, particularly when IOMMU is enabled.



Corn

Corrupted outputs. Not a crash, a wrong answer.



Herman

A wrong answer that looks like a right answer. Which is a theme we're going to come back to.



Corn

Let's come back to it now, actually, because that's the point where the mechanisms stop being theory and start being your problem. Take the mismatched rig. Different memory, different generation, different vendor. What actually happens.



Herman

So there's a documented rig that did exactly this, and I think it's the most useful single data point on the subject. An NVIDIA V100 with thirty-two gigabytes, plus two AMD Radeon AI Pro R9700 cards, which are RDNA4, also thirty-two gigabytes each. Three cards, two vendors, running one model in one container, using the Vulkan backend and split mode layer.



Corn

Vulkan is the interesting part. That's the one that surprised me.



Herman

It surprised me too. The finding from that writeup is a single sentence: CUDA is NVIDIA only, ROCm is AMD only, and the one backend that talks to both is Vulkan. That's it. That's the whole answer to mixed-vendor in one process. If you want an NVIDIA card and an AMD card feeding the same model, Vulkan is the only path there without something more exotic.



Corn

And it worked. What did it produce.



Herman

It produced a working model. On a Qwen model at eight-bit, roughly thirty-five billion parameters, they got somewhere between twenty-nine hundred and thirty-seven hundred tokens per second on prefill, and about sixty-two tokens per second on generation. Over PCIe three, which is not fast.



Corn

Sixty-two tokens a second is usable. That's not a toy.



Herman

It's usable for one person. And it's the proof that the capacity game works. But now the cost, and this is where the writeup is admirably unsentimental about its own rig. There is no cross-vendor peer-to-peer. None. It doesn't exist. So every single time the model crosses from an NVIDIA card to an AMD card, the data goes from the GPU out over PCIe, into system RAM, and back down PCIe into the other card.



Corn

Every layer boundary.



Herman

Every layer boundary that sits between vendors. On PCIe three, at roughly sixteen gigabytes a second per slot each way, the inter-GPU path is the slowest part of the whole system. And their conclusion is the line I'd put on a poster: layer split across vendors buys you capacity, it does not buy you speed. And then the follow-through: if your goal is maximum throughput rather than maximum model size, you will often do better running separate single-vendor instances and routing between them.



Corn

Which is data parallelism. Two separate piles of cards serving the same model, no cooperation at all.



Herman

Two separate servers in one box, effectively. It's less exciting, and it's usually faster.



Corn

Before we go further, I want to flag something, because it's the part of this that would actually burn me. The silent corruption.



Herman

That's the right thing to flag. The multi-GPU toolkit that documents these backend combinations reports that splitting a model across two AMD GPUs corrupts the output on ROCm, and it is silent about it. Silent. No error, no warning, the model just produces wrong text. Their advice is to use Vulkan to do the split instead.



Corn

And elsewhere? Windows, I think.



Herman

There's a llama.cpp issue reporting tensor-split instability on Windows as well. So the pattern across all of it is that a multi-GPU split is not automatically correct. It can be wrong in ways that don't announce themselves. Which means if you're going to do this, you validate outputs. You don't just look at a benchmark number and conclude that because it was fast, it was right.



Corn

Sixty-two tokens a second of confident nonsense.



Herman

Confident nonsense, at sixty-two tokens a second.



Corn

Alright. Now flip the scenario. Purpose-built server, several identical cards, same vendor, chosen deliberately. What actually gets easier.



Herman

Uniform memory, uniform compute, and the option of a fast interconnect. The first two are convenience. The third one changes what's possible. Identical cards mean the automatic splits behave, the ratios are trivially one to one, and you can use tensor mode without the uneven-split problems, which is exactly the mode that needs a fast link.



Corn

Which leads us to the thing people spec wrong, I think. The motherboard.



Herman

The motherboard, the lanes, and the topology. This is where more of these builds go wrong than at the card selection stage. Because here's the number that reframes the whole thing: the same GPU, the same model, the same flags, run at PCIe five point oh by sixteen lanes, gave fifteen hundred and thirty-two tokens a second on prefill. At eight lanes, eleven seventy-five. At PCIe four point oh by four lanes, two hundred and ninety-nine.



Corn

Two ninety-nine.



Herman

From fifteen thirty-two. That's three point nine times the prefill for eight times the link width.



Corn

So the card you bought is not the thing you bought.



Herman

The card you bought is half the thing. The guidance that came out of that measurement is blunt: lanes to the GPU matter more than the GPU, as long as any of the weights live off it. And if you're splitting a model across cards, weights are living off it constantly.



Corn

This is a build-planning point. You put two cards in the two slots that are wired for eight lanes each and one of them is running at a quarter speed, and you'd never know from the spec sheet.



Herman

You'd never know, and then you'd blame the software. Which brings me to the counterintuitive result, and I think this is the direct answer to Daniel's last question, whether a second GPU is necessarily worthwhile.



Corn

Go on.



Herman

When the model already fits on one card, adding a second card makes things worse. Not marginally worse. Up to three times worse. There's a measured case with a large mixture-of-experts model, gpt-oss one twenty b, that ran at two hundred and fifty-eight tokens a second on a single card, and about eighty-five tokens a second in a dual-card configuration.



Corn

You added a card and lost two thirds of your speed.



Herman

Because now every token involves coordination between two cards, and the coordination costs more than the parallelism buys. The rule the toolkit distills from it is four words: only split what does not fit.



Corn

And if it doesn't fit?



Herman

Then a second card is worth a lot, even a bad one. This is the flip side and it's surprising. On a unified memory machine, they moved eighteen overflow expert layers out of system RAM and onto the integrated GPU, which shares the same memory as the CPU, so nothing about the memory changed. Same bandwidth, same sticks.



Corn

What changed.



Herman

Who computes on them. That's the entire change. And it gave sixty-six percent more prefill and fifty-eight percent more generation. Just by letting a weak little integrated GPU do the arithmetic instead of the CPU doing it across the bus.



Corn

Because the CPU reading from system memory over the bus is slower than a slow GPU reading from the same memory.



Herman

That's the insight, and it generalizes further than you'd think. Generation speed is set by the memory bandwidth where the weights live, not by how fast the chip is. A weak GPU reading from fast memory beats a strong GPU reading from slow memory. Every time.



Corn

Which is the opposite of how people shop for this. They shop on teraflops.



Herman

They shop on teraflops, and for generation the teraflops are largely irrelevant. Prefill is compute-bound when the weights are resident where they should be, but the moment they spill over a link, prefill becomes transfer-bound, and then the link dominates. Two different bottlenecks for two different phases of the same request.



Corn

There was something about memory carve-outs on those unified machines that went the other way, though. A case where giving the GPU more memory made it worse.



Herman

Yes, and it's a lovely trap. Same unified machine, they gave the integrated GPU a sixty-four gigabyte dedicated carve-out. On paper, more memory for the GPU. In practice, Vulkan prefill dropped forty-five percent, from three hundred and fifty-two tokens a second down to one eighty-eight. Because that carve-out memory isn't visible to the CPU, so host writes to it had to go through a small base address register window and a staging buffer.



Corn

More memory, less speed. Because the path to it got narrower.



Herman

It's the same lesson as the PCIe lanes. Where the memory is matters more than how much of it there is.



Corn

Alright, the tooling split. NVIDIA, AMD, and the things that only exist on one side.



Herman

vLLM handles both, which is worth saying clearly. Tensor and pipeline parallelism work on CUDA and on ROCm. On one node it uses Python multiprocessing to fan out, and across nodes it uses Ray. AMD publishes its own tuning guidance for the MI300X and MI350X class cards, the communication libraries, the attention backend selection, all of it.



Corn

And AMD's own advice on scale?



Herman

Their guidance is to stay inside a single node, which is up to eight GPUs on those systems, and use tensor or data parallelism there, adding pipeline parallelism only once you go beyond eight. Beyond a node you're crossing between machines and the fabric changes character.



Corn

What does the eight-card picture actually look like.



Herman

There's a good benchmark on their side with DeepSeek R1, the six hundred and seventy-one billion parameter model, on eight MI300X cards. With tensor parallelism across all eight plus expert parallelism, you get about fifty-two percent higher throughput and eighty percent lower time to first token at a concurrency of sixty-four compared to the alternative arrangement. And at very high concurrency, the data-parallel arrangement reached seven thousand one hundred tokens a second, which was about forty-seven percent higher than the tensor-parallel one.



Corn

So at high load, replicas beat splitting.



Herman

At high load, replicas beat splitting, for the same reason as always. Splitting is a capacity tool. Replication is a throughput tool. And there's a specific mechanism for that which is worth knowing if you're ever doing KV cache math. Tensor parallelism across eight cards duplicates the full KV cache eight times over, so your effective concurrency is one times, not eight. Data parallel plus expert parallel gives you eight times the batch capacity. It's the same memory going to work in completely different ways.



Corn

And TensorRT-LLM, which is where the vendor lock-in is least subtle.



Herman

NVIDIA only, Blackwell and Hopper. They have a whole parallelism feature page and the entire thrust of it is NVLink, up to the rack-scale systems. There's material on optimizing mixture-of-experts communication over NVLink with one-sided collective operations. It's excellent, and it assumes you bought NVIDIA's interconnect.



Corn

Which is the point. At the top end it's not a software choice, it's a purchase.



Herman

VLLM and llama.cpp are the portable options. llama.cpp is the most portable of all, it speaks CUDA, ROCm, Vulkan, SYCL, Metal, and it has an RPC backend. But here's the catch with that portability: Vulkan is the only one of those that spans vendors inside one process. Everything else is one vendor per process.



Corn

So the vendor-neutral question, the one Daniel asked directly. Is there a real answer.



Herman

Vulkan is the accidental escape hatch, and I'd call it accidental because it wasn't built for this. It works, you get one model across two vendors, and you pay for it in two currencies. You lose peer-to-peer, since Vulkan isn't giving you that, and you lose the vendor-tuned kernels, the hand-optimized matrix multiplication paths that CUDA and ROCm both have.



Corn

There was also a comment in the writeup about the tooling for this being conspicuously absent.



Herman

There's no general-purpose tool that pools mismatched VRAM transparently. Nothing scans your cards, sees a twelve gigabyte card and a twenty-four gigabyte card, and balances the model for you at load time. Pooling is always explicit configuration, the split ratios and the tensor placement flags, or it's a specific backend choice like Vulkan. The llama.cpp automatic split is proportional to memory, which is the closest thing to what people imagine, and as we said, it doesn't account for the KV cache and the compute buffers, which is why the heterogeneous-card case is still being worked on.



Corn

And the RPC backend, which is the other way to pool memory, if you're willing to leave the machine.



Herman

The RPC backend is interesting and you should know what it costs. You run a small server on a second machine, it exposes that machine's devices, and llama.cpp on the first machine treats them as additional devices. So your second machine becomes another pile of VRAM.



Corn

And the catch.



Herman

Several. The documentation calls it a proof of concept, fragile and insecure, and there's no authentication at all. That's not a small thing, it's a listening socket with no auth that lets a remote machine run computations on your hardware.



Corn

Or the reverse, depending on how you've got it pointed.



Herman

Either way, all the traffic goes over your network, which is slower than PCIe by a lot. So the honest framing of what it buys you: it turns I can't run this model at all into I can run this model slowly. It does not turn slow into fast. And the most common failure isn't bandwidth. It's version skew, the two machines running different builds and hanging at the handshake.



Corn

So that's the map. Now the decision, which is where I'd want to end up if I were Daniel's reader. Someone's got one card, they're thinking about a second one. What's the actual framework.



Herman

First question, and it's binary: does the model fit on what you already own. If it fits, do not split it. Run independent instances instead, because adding a card to a model that fits loses you up to three times the speed, as we said. That's the rule, only split what does not fit.



Corn

And if it doesn't fit.



Herman

Then the second card buys you capacity, and the question becomes which card and how it's attached. A weak card in fast memory beats a strong card behind a slow link. If you have a choice between one bigger card and two smaller ones, one bigger card is simpler, always, because there's no communication and no split logic and nothing to validate. Splitting is what you do when the single big card doesn't exist at a price you'll pay.



Corn

And the deliberate server buyer.



Herman

Buy identical cards, buy a board with enough lanes for all of them at full width, and buy the interconnect if the interconnect is on offer, because tensor mode lives and dies by it. Then validate your outputs, not just your throughput numbers. Run a known prompt and check the answer actually makes sense, because we've now got documented cases of silent corruption.



Corn

Capacity or speed. Those are the two things, and the shopping list is different for each.



Herman

That's the whole episode in one line, and it took us an hour of research to get comfortable saying it that way, but that's it. Capacity or speed, and the interconnect decides which one you're actually buying.



Hilbert

The number's thirty-one. Not twenty-four.



Corn

Which number.



Hilbert

The carve-out. You said sixty-four gigabytes, that's what they set it to. But the ceiling on that window that's actually visible from the host is thirty-one gigabytes before the driver hands you a stub. Twenty-four, people keep saying twenty-four, that's the last generation. It's thirty-one.



Corn

Noted. Thirty-one.



Hilbert

My sister's husband does this for a living. Sells the server boards, not the cards. He says the cards are the easy part, the motherboard is where he makes his money.



Corn

That's conveniently aligned with his commission.



Hilbert

He says the second card is never the expense. The second card is the cheap part, because you've already paid for the case and the power supply and the board. What you pay for is the second slot actually running at width. He had one customer who bought three cards and ran them at four lanes each because the board only had one full-width slot and two that were wired for a chipset. The third card was decoration.



Corn

Decoration in a three-thousand-dollar case.



Hilbert

He charged him a design fee after the fact. First one he ever charged a design fee to, he said, because the man argued with him twice about whether it mattered.



Corn

Did it matter.



Hilbert

It mattered. He's not to be trusted on pricing, my brother-in-law, he marks everything up and he knows it. But he's right about the lanes.



Corn

How much did he mark it up.



Hilbert

I don't know what he charges. I know what he paid for the last one he sold me, and I'm not going to repeat it, because it makes the other one look reasonable.



Corn

The other one.



Hilbert

There was a machine. It was in the basement, and it was bolted to a board that had been bolted to the joists, because it wouldn't sit flat with the cards in it. Four cards. It lived down there for a while.



Corn

Bolted to the ceiling.



Hilbert

To the joists. The board went up first. It's still there, actually, the board is, the machine came down when I moved the desk.



Corn

And the cards.



Hilbert

That's what I want to say about the whole subject. Everybody talks about the split and the backend and none of it is the hard part. The hard part is that a second card is a physical object that has to go somewhere it fits and get air it can't get and sit on a board that will actually run it. Everything after that is software, and software you can fix.



Corn

That's a fair point.



Hilbert

My brother-in-law's the one who sold me the board.



Corn

So the board's still on the ceiling.



Hilbert

The board's still on the ceiling. He asked me about it at the wedding.



Corn

Alright. So where does that leave someone thinking about a second GPU.



Herman

I think it leaves them with a question to ask before they spend anything, which is what they're actually buying. If they want the model that doesn't fit, a second card is the only path and almost any card helps, provided the weights land in fast memory. If they want more throughput on a model that already fits, the second card is a slowdown and the money is better spent on a better single card or on a second machine.



Corn

And the honest summary from all of it is that this is a capacity tool. The speed shows up as a side effect of how well the cards are wired together, and if they're not wired well, it doesn't show up at all.



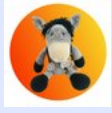
Herman

That's the part that doesn't reach the product listings. Nowhere on the box does it say what the card will do at eight lanes, and that's the number that decides it for a lot of these builds.



Corn

There's still no tool that looks at two mismatched cards and quietly does the right thing. That gap is real and it's the thing I'd most want someone to take from the hour.



Herman

And validate the output. That's the other one. Fast and wrong is the worst combination in this whole space.



Corn

That was Daniel's question, more or less, and it turned out to have a more interesting answer than the yes or no he was probably expecting. If any of you have run a model across mismatched cards, we want to hear about it. Hilbert Flumingtop produces the show, and he'd want you to check your lanes.



Herman

Check your lanes.



Corn

For more along these lines, there's episode two, Local STT For AMD GPU Owners; episode seventeen, Cloud Render Superpowers; and episode eighteen, Beyond the GPU. This has been My Weird Prompts.



Herman

Send us your own prompt on Telegram at t dot me slash MWP listener bot. We'll be back soon.

SOURCES

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

1. **llama.cpp `docs/multi-gpu.md` [primary]**
   (fetched 2026-10-14)
   https://raw.githubusercontent.com/ggml-org/llama.cpp/refs/heads/master/docs/multi-gpu.md

2. **vLLM Parallelism and Scaling [primary]**
   (fetched 2026-10-14)
   https://docs.vllm.ai/en/stable/serving/parallelism_scaling/

3. **vLLM repo (docs/serving/parallelism_scaling.md, csrc/rocm, Dockerfile.rocm) [primary]**
   https://github.com/vllm-project/vllm

4. **llama.cpp repo (ggml-cuda, ggml-hip, tools/rpc) [primary]**
   https://github.com/ggml-org/llama.cpp

5. **TensorRT-LLM docs (home + Parallelism in TensorRT LLM feature page) [primary]**
   (updated 2026-10-13)
   https://nvidia.github.io/TensorRT-LLM/

6. **AMD ROCm blog, The vLLM MoE Playbook: A Practical Guide to TP, DP, PP and Expert Parallelism, 2025-11-24**
https://rocm.blogs.amd.com/software-tools-optimization/vllm-moe-guide/README.html

7. **grosan.co.uk, Layer Split Model Parallelism on Hybrid AMD NVIDIA AI Servers using Vulkan and Llama CPP, 2026-06-02**
https://grosan.co.uk/layer-split-model-parallelism-on-hybrid-amd-nvidia-ai-servers-using-vulkan-and-llama-cpp/

8. **daimonionnn/multi-gpu-llm-toolkit (README, doc/performance-model.md, doc/rocm-bugs.md), created 2026-04-09**
https://github.com/daimonionnn/multi-gpu-llm-toolkit

9. **SharedLLM, llama.cpp RPC backend: distributed inference across multiple machines, 2026-07-18**
https://sharedllm.org/blog/llama-cpp-rpc-distributed-inference.html

10. **llama.cpp Discussion #19544, Split model between AMD and NVIDIA (answered 2026-02-12)**
https://github.com/ggml-org/llama.cpp/discussions/19544

11. **llama.cpp Issue #12654 (VRAM-aware layer splitting) and PR #14484 (compute-buffer/KV-cache-aware layer distribution)**
https://github.com/ggml-org/llama.cpp/issues/12654