Episode Book

EPISODE #5961

What Your GPU Is Actually Waiting On

PUBLISHED	14 October 2026 at 14:35 UTC
DURATION	27:27
LISTEN	myweirdprompts.com/episode/gpu-memory-bandwidth-decode-21799/
HOSTS	Corn, Herman, Hilbert
SCRIPT MODEL	deepseek-flash
VOICES	chatterbox-regular (chatterbox-local)
PIPELINE	V5.4
SOURCES	13 cited by the research (listed at the end)
BOOK GENERATED	19 October 2026 at 17:42 UTC

SYNOPSIS

You've got the model loaded, VRAM to spare, and the GPU reports near-total utilization — so why is generation crawling? This episode pulls apart the single number everyone screenshots and looks at the physical mechanisms hiding underneath it. VRAM capacity is a feasibility check, not a performance prediction; decode is a streaming memory workload that runs sixty to two hundred times to the memory-bound side of the roofline. Along the way: why a warp stalled on a memory read still counts as "active," how tensor core fragments sixty-four rows tall turn eight real requests into a high-speed multiplication of zeros, and why the CPU can eat thirty percent of decode time at large batch sizes. Then the practical half — consumption versus saturation, what to measure on the rest of the box, and how to walk up to a slow Linux server and actually figure out what's wrong.

TRANSCRIPT



Corn

We spent a whole episode on what the engine tells you. Tokens per second, time to first token, batching efficiency. All of that is the view from inside the software.



Herman

And today we flip the board over and look at the metal underneath it.



Corn

Daniel's question picks up exactly there. He says we did inference benchmarking from the engine's perspective, and now he wants the other side of the equation. Hardware level monitoring, on a Linux box, running open weight models. And his framing is the part that stuck with me. When people talk about hardware for local AI, the conversation is one resource. How much VRAM does the model need. Will it fit. How much is left over.



Herman

Like the model is a suitcase and the GPU is the overhead bin.



Corn

That's the whole conversation. And Daniel's point is that capacity is one number out of a whole board of them. So he's asking what else is worth watching. Compute utilization, memory bandwidth, memory controller activity, clocks, power, temperature, throttling. Then the rest of the box. CPU, system RAM, disk, PCIe, CPU to GPU traffic. And then the distinction I think is the real question underneath all of it. Consumption versus saturation. A GPU with nearly all its VRAM allocated is not the same thing as a GPU that is saturated, and a model that fits comfortably in memory can still be badly limited. So, what do you actually measure, what does it mean, and what would you check first on a slow server.



Herman

And the tooling question follows from that. NVIDIA versus AMD, which utilities for quick troubleshooting, which for continuous monitoring and alerting.



Corn

Right. So the through line here is that one GPU utilization number is not a measurement. It's an average of several different physical mechanisms wearing the same name, and the work is pulling them apart.



Herman

Here's the scene I want in everyone's head. A Linux box, one GPU with a lot of VRAM, an open weight model loaded. Something like DeepSeek. And the community conversation about whether this machine can run it stops at does it fit.



Corn

Fit is a yes or no question. It's not a performance question.



Herman

Correct, and that's the first thing to unlearn. VRAM capacity is a feasibility check. It tells you whether the model loads. It predicts almost nothing about how fast it will run. Fitting is the beginning of the question, not the end of it.



Corn

So if I've got a hundred gigabytes of model sitting in a hundred and forty gigabytes of VRAM, what have I actually established?



Herman

That you can start. That's it. You have established that the weights are resident and you won't be paging them in from system memory on every token. Which matters, but it's a floor, not a ceiling.



Corn

It's the difference between a car that has fuel in it and a car that's fast.



Herman

Well, and even that undersells it. Because the thing that actually determines your tokens per second is usually not the capacity at all. It's the bandwidth of the memory the weights live in. You can have a model that fits in a card with enormous headroom and still be badly bandwidth limited, because the model is small enough to fit and the card's memory is slow.



Corn

So a card with more memory can be slower than a card with less memory.



Herman

Constantly. That's the whole trap. Capacity is the spec on the box that people can compare easily. Bandwidth is the spec that actually governs decode.



Corn

Which gives us the two ideas for today. One, the counter isn't broken, it's over aggregated. GPU utilization collapses a bunch of mechanisms into one number. And two, consumption is not saturation. A resource can be almost fully allocated and still not be the thing that's holding you back.



Herman

And so the arc is this. What the GPU counters actually measure, then why decode is structurally memory bound, then the rest of the system, then consumption versus saturation properly, then the tooling, NVIDIA versus AMD, and finally how you'd actually walk up to a slow server and figure out what's wrong.



Corn

Let's start where the numbers come from. What is the GPU even telling you when you ask it how it's doing?



Herman

So the canonical answer on the NVIDIA side is DCGM, the Data Center GPU Manager, and specifically its profiling module. Module eight, if you're reading the docs. It exposes a big family of low overhead hardware counter metrics.



Corn

And what's in the family?



Herman

Graphics Engine Activity is 1001. SM Activity is 1002. SM Occupancy is 1003. Tensor Activity is 1004. Memory bandwidth utilization, which is really DRAM activity, is 1005. Then the pipe activities, FP64, FP32, FP16, at 1006, 1007, 1008. PCIe transmit and receive bytes at 1009 and 1010. NVLink transmit and receive bytes at 1011 and 1012. Default sample rate is one hertz, and you can push it down to a hundred milliseconds minimum.



Corn

A hundred milliseconds is fast enough to catch a stall between decode steps.



Herman

It is, and that turns out to matter later. But here's the caveat that I think is the single most important sentence in the documentation, and it's NVIDIA's own. SM Activity counts warps that are waiting on memory requests as active.



Corn

Say that again.



Herman

A warp that is stalled on a memory read is counted as active. The docs say it plainly. Active does not necessarily mean a warp is actively computing.



Corn

So the number that people screenshot and post as proof their GPU is working hard is partly counting warps that are doing nothing but waiting.



Herman

And NVIDIA even gives you a threshold, which is refreshingly blunt. A value of zero point eight or greater is necessary but not sufficient for effective use of the GPU. Below zero point five likely indicates ineffective usage.



Corn

Necessary but not sufficient. That's a very careful sentence from a company that would love to sell you more GPUs.



Herman

There are also two practical gotchas. Profiling counters require admin privileges, and that goes back to Linux driver 418.43. And some metrics cannot be collected at the same time. There's multiplexing inside the counter hardware, so if you crank the sample rate up and ask for everything, you can get zeros back.



Corn

And then people see zero percent and panic.



Herman

They open an issue. The card is fine. You asked it for incompatible things too fast.



Corn

So now the mechanism behind the thing Daniel specifically asked about. High memory usage, low computation. Why.



Herman

Because decode is structurally memory bound, and it's not a tuning problem, it's arithmetic. Take the H100. It has 989 teraflops of BF16 compute and 3.35 terabytes per second of HBM bandwidth.



Corn

Those two numbers are wildly different in scale.



Herman

They are, and the roofline model tells you how they interact. You divide compute by bandwidth and you get a ridge point. For the H100 that's roughly 295 floating point operations per byte.



Corn

Meaning what, in plain terms?



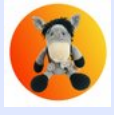
Herman

Meaning if your workload does at least 295 operations for every byte it pulls from memory, you're compute bound and the card's math units are your limit. If it does fewer, you're memory bound and the card's memory bandwidth is your limit. And decode steps during generation have arithmetic intensity of about one to five flops per byte.



Corn

One to five, against a ridge at two hundred ninety five.



Herman

Sixty to two hundred times to the left of the ridge. Not close. There is no amount of tuning that moves a workload two orders of magnitude across a roofline. Decode lives on the memory bound side, permanently.



Corn

So the card is pulling weights out of HBM as fast as it can and doing almost nothing with each byte.



Herman

And there's a field note that makes this concrete. A production H100 decode deployment, observed at four percent compute utilization.



Corn

Four.



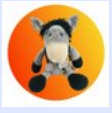
Herman

Four. DRAM read activity running flat, meaning the memory subsystem was working hard the whole time. And over fifty percent of the cycles in the attention kernel stalled waiting on HBM.



Corn

So the answer to Daniel's question is that the GPU isn't lying. It's busy. It's busy waiting.



Herman

The line from that writeup is the one I'd put on a wall. Decode is a streaming memory workload wearing a deep learning costume. And then, the number that matters for decode is not teraflops, it is terabytes per second.



Corn

That reframes the entire purchasing conversation.



Herman

It does, and there's a second mechanism that I think is even more satisfying once it clicks, because it explains why the utilization counter reads high while nothing useful happens. It's about how the tensor cores actually execute.



Corn

Go on.



Herman

On Hopper, the bfloat16 GMMA path. The tensor core does not multiply an arbitrary sized matrix. It executes fixed sized fragments. The fragments are sixty four rows tall.



Corn

Sixty four rows, fixed, regardless of what you asked it to do.



Herman

Regardless. So picture a decode step with eight concurrent requests. Each request contributes one new token, so one row of real data. Eight real rows.



Corn

And the fragment is sixty four rows.



Herman

So you have eight real rows and fifty six rows of padding, and the tensor core executes all sixty four because that's the shape it's built to execute.



Corn

The GPU is multiplying zeros.



Herman

At high speed, drawing full power, reporting high tensor activity, and delivering the throughput of eight tokens.



Corn

That's the aha. That's the whole thing. The counter is telling the truth about the wrong noun.



Herman

And it gets worse with the padding, because padding rows still consume memory bandwidth. So you're not just wasting compute cycles, you're wasting the resource you were actually short on.



Corn

So the fix, if there is one, is decomposition. Stop looking for a better single number and start asking what specifically the GPU is waiting on.



Herman

And the good news is the counters for that exist. They're just not the number on the front of the dashboard.



Corn

Alright, so the GPU is busy multiplying zeros. Which raises the obvious next question. What is it waiting on, and is the answer even on the GPU?



Herman

Very often it isn't. And this is where the framing shifts. Inference is not one GPU operation. It's a pipeline, and the GPU is one stage of it. The GPU can be perfectly available while the rest of the system fails to feed it work fast enough.



Corn

So the better first question is what is the GPU waiting for.



Herman

Which is exactly the framing one of the writeups used. Stop asking whether the GPU is busy and start asking what it's waiting on. So let's walk the non GPU resources. CPU preprocessing and tokenization first.



Corn

Because before a token exists, somebody has to turn text into integers.



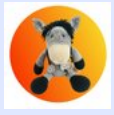
Herman

Somebody has to tokenize, build the batch, prepare the inputs. And the measurement here is striking. CPU time reaching up to thirty percent of decode execution at batch size five hundred twelve, on a model that is not large, OPT one point three billion.



Corn

Thirty percent of decode time is the CPU.



Herman

And the same work found a noticeable GPU idle gap between decoding steps, attributed to CPU processing time. So the GPU finishes a step, and then sits there while the CPU assembles the next one.



Corn

And that gap grows with batch size.



Herman

It grows with batch size, because there's more input to prepare. So the exact configuration where you'd expect the GPU to be most efficient is the configuration where the CPU eats the most of your time.



Corn

Which is a beautiful inversion of the usual story. People scale batch size up to use the GPU better, and hand more work to the part they weren't watching.



Herman

The same paper measured cache behavior too. L1 hit rates of twelve percent or less. L2 hit rates of two percent or less. And both get worse as batch size climbs.



Corn

Two percent on L2. Basically nothing is staying in cache.



Herman

Which means almost every access goes out to memory. And memory bandwidth is the thing we already established is the bottleneck. So it compounds.



Corn

Now give me the consumption versus saturation distinction properly, because that's the heart of this.



Herman

This is where it gets good. At maximum batch size in that same work, average DRAM read utilization was between forty eight and seventy seven percent.



Corn

So the memory was working, but not flat out.



Herman

Not flat out. And meanwhile average compute, warps actually in flight, never exceeded thirty five percent. And the metric called Unallocated Warps in Active SMs sat between forty and sixty six percent.



Corn

Unallocated warps. Meaning there were warps ready to run and the scheduler had nowhere to put them.



Herman

The paper's phrasing is that something is preventing warps from being scheduled. DRAM bandwidth saturation as the primary bottleneck, with most GPU compute capability sitting idle.



Corn

So you have a resource that is saturated and a resource that is merely consumed, in the same machine, at the same moment.



Herman

And that's the distinction Daniel asked for, in one measurement. Memory bandwidth saturated. Compute consumed but not saturated. Two different diagnoses with two completely different fixes.



Corn

And here's the pairing that resolves it, I think.



Herman

Model FLOPs Utilization and Memory Bandwidth Utilization. MFU and MBU. MFU tells you what fraction of the card's math throughput you're actually using. MBU tells you what fraction of its memory bandwidth you're using.



Corn

And the diagnosis falls out of the two of them together.



Herman

When MBU approaches one hundred percent while MFU stays low, you have confirmed that memory bandwidth is your ceiling and your system is operating correctly.



Herman

It's the sentence that turns low utilization from a problem into a diagnosis. Your card is fine. Your model is memory bound. That is what running a large language model on a GPU normally looks like.



Corn

So all the dashboards with the red ninety percent threshold baked in are alarming on healthy systems.



Herman

Constantly. Which is why the autoscaling story gets so ugly. Picture a system that scales out on a utilization trigger. Utilization is high, so it spins up another replica.



Corn

Adding replicas means splitting the traffic, so each replica gets a smaller batch.



Herman

Smaller batch, less fragment fill on those sixty four row GMMA fragments, more padding. Which keeps utilization high. Which triggers more scaling.



Corn

They're paying for more GPUs to sustain the misleading signal.



Herman

A feedback loop that spends money to keep the number that alarms you alive.



Corn

There's also the other direction. Bigger GPUs making utilization look worse.



Herman

Yes. Upgrade the accelerator, keep the pipeline the same, and you get a faster GPU waiting on the same old CPU. Utilization drops, because now the card is even less busy while it waits.



Corn

You spent the money and the number got worse, and the number was right.



Herman

Right. Which is why clock speeds, power, temperature and throttling belong in this conversation. Those are saturation signals that never show up in the utilization percentage at all.



Corn

Explain.



Herman

A thermally throttled GPU can report perfectly healthy utilization while running at a reduced clock. Utilization is a ratio. If the denominator shrinks because the card is running slow, the ratio can look great while absolute throughput is awful.



Corn

So the one number everyone watches is structurally blind to the failure that hurts most in a hot rack.



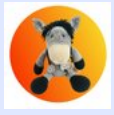
Herman

And AMD gets this shape right, incidentally. RDC exposes power throttle time and thermal throttle time as explicit health fields. That's the correct shape for this. Not a utilization percentage, a duration spent throttled.



Corn

What about offload and multi GPU?



Herman

When weights or the KV cache spill into system memory, PCIe and system RAM bandwidth stop being footnotes and become first class constraints. Suddenly your host memory bandwidth matters as much as the card's.



Corn

And you'd never know it from the card.



Herman

From the card, everything looks ordinary. With multiple GPUs, interconnect traffic becomes its own observable. That's why DCGM exposes NVLink transmit and receive bytes at 1011 and 1012. It's not decoration. On a multi GPU deployment the interconnect can be your ceiling.



Corn

Okay. So we know what to look for. Now, what do you actually type?



Herman

Split it by job. If you're troubleshooting interactively, right now, on one machine, the answer is nvidia-smi and its neighbors. nvidia-smi takes query options. You can ask for the whole report, you can ask for specific device categories, memory, utilization, power, clock. It loops with the right flag. It outputs CSV or XML, which is how it ends up in scripts. And it's backed by NVML, so there are Python bindings if you'd rather write the query.



Corn

What about the eye candy.



Herman

nvtop is the htop shaped one, and I'll come back to it, because it punches far above its weight. nvitop is the NVIDIA process viewer. btop and htop for the CPU side, iostat for disk.



Corn

That's the interactive tier.



Herman

That's the interactive tier. Then there's the tier for questions like what happened at three in the morning on Tuesday. That's DCGM plus dcgm-exporter into Prometheus into Grafana. dcgm-exporter feeds the time series, so you get history, and you get alerting.



Corn

DCGM's own health side?



Herman

Health watches, diagnostics with four run levels, and policies that alert on ECC and thermal events. DCGM 4.6.1 came out in August, under Apache 2.0, which matters for anyone who needed to care about licensing. And dcgmi dmon streams fields to your terminal if you want the live version without building a stack.



Corn

Then there's the third tier, which is different in kind.



Herman

Profiling. Nsight Systems for timelines, Nsight Compute for per kernel counters. That's where you go when you need to know which kernel is the problem, not which resource. And there's a conflict worth knowing about. DCGM profiling counters and Nsight can't run at the same time. You have to pause the one to run the other.



Corn

You have to stop your monitoring to profile.



Herman

There's a command for it, pause and resume, but yes. You cannot have both live on the same card simultaneously.



Corn

Here is where the story stops being symmetric, because the two vendors do not ship the same shape of telemetry.



Herman

They don't. And the interesting part is that it isn't really about which company has better tools. It's about architecture. Start with what AMD ships. amd-smi, which replaced rocm-smi. MIT licensed, with C plus plus, Python and Go libraries, JSON and CSV output.



Corn

Which sounds competitive on paper.



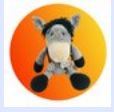
Herman

On the single machine level it is. Then RDC, the ROCm Data Center tool, which is a daemon, rdcd, plus a CLI, rdci. And RocProfiler for the profiling tier.



Corn

AMD has published the mapping between their fields and DCGM's.



Herman

They have, which is helpful. In July they put out a blog mapping RDC onto DCGM equivalents. Power and thermal throttle time against DCGM's two violation fields. XGMI read and write against NVLink. ECC via RAS. And their busy percentage fields against SM Activity, SM Occupancy and Tensor Activity.



Corn

On the single machine, the correspondence is there.



Herman

On the single machine it's there. Here's the asymmetry. AMD SMI has no daemon and no metrics endpoint.



Corn

No daemon at all?



Herman

No daemon, no endpoint. It reads and controls one machine. Fleet collection is left to whatever you build on top of its libraries. Whereas NVIDIA ships dcgm-exporter and the Kubernetes GPU Operator. So the fleet story is a product for one vendor and a project for the other.



Corn

That's not a small distinction for anyone running this at scale.



Herman

It isn't. A comparison scored DCGM seventy six out of a hundred against AMD SMI at seventy, and the widest gap was exactly there, in fleet integration.



Corn

The single machine gap is fourteen points of nuance and the fleet gap is the difference between configuring something and building something.



Herman

That's a fair way to put it.



Corn

What about the honest cross vendor answer. Is there a tool that just works on both?



Herman

nvtop is the standout, and I want to give it credit properly. Version three point three two supports AMD, Apple, Huawei Ascend, Intel, NVIDIA, Qualcomm Adreno, Broadcom VideoCore, Rockchip, MetaX, Enflame, Tenstorrent, Iluvatar CoreX. That's a remarkable list. And it plots GPU and memory utilization, temperature, power, clocks, and PCIe receive and transmit load.



Corn

PCIe load is a nice touch, given everything we just said about offload.



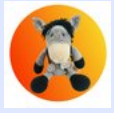
Herman

It's exactly the metric the offload conversation needs, and it's right there in the free tool. But here's the limit, and I want to be straight about it. No vendor neutral tool provides consistent hardware counter or profiling telemetry across NVIDIA and AMD.



Corn

Why not? It seems like a solved problem on its face.



Herman

Because the signals don't always measure the same physical quantity. There's an open project that explicitly refuses to alias AMD signals onto NVML shaped names for exactly this reason. Their argument is that AMD's graphics activity counter and NVIDIA's SM utilization counter sample at different scopes. They're not the same measurement with two names. They're two different measurements that happen to sound similar.



Corn

A vendor neutral dashboard would be presenting two different quantities under one label.



Herman

Which is worse than no dashboard, because it looks comparable. nvtop gives you a consistent interactive view, and that's real value. The deep counters stay vendor specific, and there's no abstraction over that as of today.



Corn

What's the practical takeaway for someone running a mixed fleet?



Herman

Use nvtop as your eyes for the quick look, everywhere. Then accept that the deep telemetry is two stacks, and staff for that. It's not going away because someone writes a wrapper.



Corn

So back to the slow server. Where would you actually start?



Hilbert

You're all describing my setup like it's a problem.



Hilbert

I've been doing this for years and I don't use any of that. I know when a card's unhappy.



Corn

By looking at it.



Hilbert

By listening to it. A memory bandwidth bound workload has a sound. It's the fan ramping and holding, steady, no oscillation. Compute bound is choppier. I called one in front of a room of people once and they asked me to leave.



Herman

You were asked to leave a room for identifying a bottleneck.



Hilbert

They were deploying at the time. It made them nervous.



Corn

How do you monitor, then. Because you clearly monitor.



Hilbert

I've got a gauge in the case. I built the case, so I mounted it myself. It's a mechanical gauge, the kind that goes on a pressure cooker. I read it by tapping it.



Herman

A pressure gauge.



Hilbert

The needle moves when the card's working. If it jumps when I tap the case, the workload's stalling. If it holds, it's streaming. I've kept a notebook of the readings for years. The software said everything was fine on a Wednesday and the gauge had been low for two days. Card died Thursday.



Corn

The gauge called it.



Hilbert

I've never trusted a dashboard since, and I don't intend to start.



Herman

The whole practice of watching counters.



Hilbert

It's a substitute for knowing the machine. You've got people watching nine numbers who've never once had the case open. Buying a dashboard instead of learning your hardware.



Corn

There is something to that. The counter tells you about a class of hardware. The ear tells you about yours.



Hilbert

I've had one of these cards and I don't have it now. Sold it when I moved. Point stands.



Corn

Alright. Herman, the order. Somebody walks up to a server that's slow.



Herman

Check throttling and clocks first. It's cheap, it's one command, and it rules out an entire class of problems before you touch anything interesting. A throttled card will make every downstream measurement look strange.



Corn

Then.



Herman

Then the pairing. Memory bandwidth utilization against compute utilization. That single comparison tells you which ceiling you're pressed against. High MBU with low MFU is a healthy memory bound system. Both high is a compute bound workload. Both low means you haven't found the real bottleneck yet and you should stop looking at the GPU.



Corn

If both are low.



Herman

CPU time and the inter step idle gap. That's where the pipeline shows up. If the GPU is waiting between steps, the answer isn't on the GPU.



Corn

Then PCIe and offload traffic, if weights or KV cache have spilled.



Herman

Disk last, and only if you're actually swapping. Disk is the one people check first and it's almost never the answer.



Corn

The framing to leave people with. You're not hunting for a bad number. You're looking for which resource is saturated and which is merely consumed.



Herman

Different diagnoses, different fixes. And only one of those two fixes is buying a bigger GPU.



Corn

The one thing I'd flag as an open question. As models get bigger and partial offload becomes routine rather than an edge case, does the GPU centric monitoring model survive? The interesting telemetry may be migrating off the card entirely, onto the interconnect and the host.



Herman

Nobody's shipping a clean cross vendor answer for that layer yet.



Corn

That's where we'll leave it. Thanks as ever to Hilbert Flumingtop for producing.



Herman

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



Corn

For more along these lines, there's episode two, Local STT For AMD GPU Owners; episode eighteen, Beyond the GPU; and episode three, Safetensors or something else. This has been My Weird Prompts.

SOURCES

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

1. **NVIDIA DCGM Profiling module docs [primary]**
   https://docs.nvidia.com/datacenter/dcgm/latest/learn/modules/profiling.html

2. **DCGM Exporter metrics [primary]**
   https://docs.nvidia.com/datacenter/dcgm/latest/reference/dcgm-exporter-metrics.html

3. **nvidia-smi man page [primary]**
   https://man.archlinux.org/man/nvidia-smi.1.en

4. **AMD ROCm blog, 2026-07-07 [primary]**
   https://rocm.blogs.amd.com/software-tools-optimization/rdc_rocprof_metrics/README.html

5. **nvtop repo, v3.3.2 [primary]**
   https://github.com/Syllo/nvtop

6. **aiperf AMD telemetry collector [primary]**
   https://github.com/ai-dynamo/aiperf/blob/5ad08166/src/aiperf/gpu_telemetry/amdsmi_collector.py

7. **AMD SMI docs [primary]**
   https://rocm.docs.amd.com/projects/amdsmi/en/latest/

8. **Dissecting GPU Utilization for LLM Inference on Nvidia Hopper, submitted 2026-09-11**
   https://arxiv.org/abs/2609.12923

9. **Groundy, 2026-09-14**
   https://groundy.com/articles/why-gpu-utilization-misleads-llm-inference-capacity-planning/

10. **Mind the Memory Gap, v2, 2025-03**
    https://arxiv.org/html/2503.08311v2

11. **Vansh Verma, 2025-11-18**
    https://vanshverma.com/notes/gpu-utilization-lie

12. **Yotta Labs, 2026-05-14**
    https://www.yottalabs.ai/post/why-llm-inference-has-low-gpu-utilization-cpu-pcie-memory-bandwidth-and-kv-cache-bottlenecks

13. **AMD SMI vs NVIDIA DCGM comparison, 2026**
    https://faun.dev/toolbox/amd-smi-vs-nvidia-dcgm/