Never idle: one stream is a memory load, 256 streams an arithmetic load

The baseline again, with three instruments on. One stream runs the card's memory at about half of its stated ceiling and its arithmetic at about 1% of the stated peak; 256 streams turn that around. The CPU stays at two cores of twenty and almost nothing crosses the network.

Qwen3.8-27B · one AMD Instinct MI300X · vLLM 0.27.1 **Run** 20261125T192827Z-probe

Changed instruments only: vLLM's estimates and counters, the profiler made available **Load** 1,024 tokens in, 256 out; 1, 32 and 256 streams

I ran the baseline's load again at 1, 32 and 256 streams with three instruments switched on: vLLM's own estimates of the work, AMD's device exporter on the card, and a profiler on the GPU kernels. With one stream, memory traffic runs at 48.9% of the card's stated ceiling and arithmetic at 0.9% of its stated peak. With 256 streams it is the other way round: 14.5% for memory, 23.2% for arithmetic. Those are vLLM's estimates; the card's own counters and the profiler tell the same story. The card is busy all the time at every level, and the CPU never is.

48.9%
of the stated memory ceiling: one stream's traffic
vLLM's estimate, of AMD's stated 5.3 TB a second

23.2%
of the stated arithmetic peak: 256 streams' work
vLLM's estimate, of AMD's stated 1.3 PFLOPS

100%
of the time the card is busy, at 1, 32 and 256 streams
AMD's exporter; the profiler agrees

2.0 of 20
CPU cores used by the engine, at every level
the machine is 10.9% busy

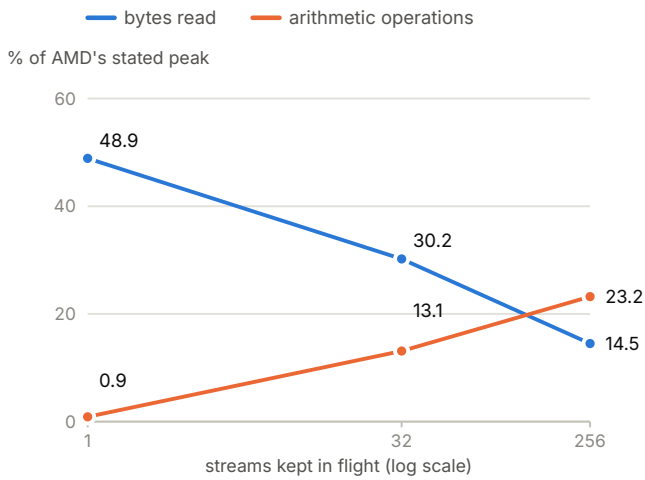

Figure 1 vLLM's estimates, as shares of AMD's stated peaks: 5.3 TB a second of memory traffic, 1.3 PFLOPS of arithmetic. Bytes read fall as streams are added; operations rise.
Source: summary.json sweeps.random-1k-256.levels[].est_read_share_of_peak, est_flops_share_of_peak

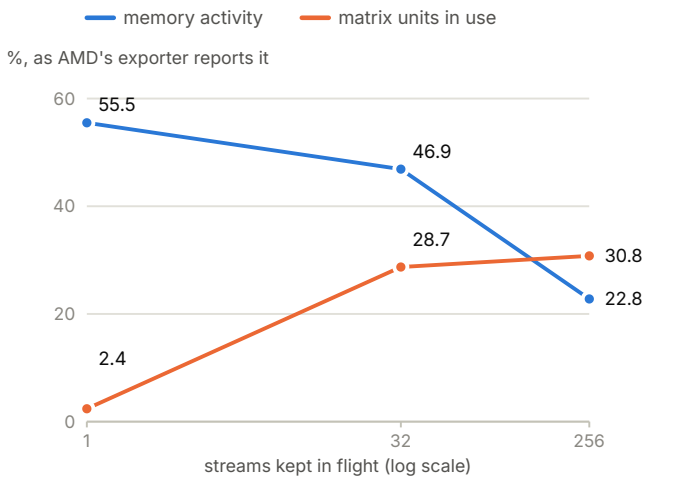

Figure 2 The card's own counters tell the same story: memory activity falls, the matrix-multiply units get busy. Means of the exporter's readings while each level ran; single readings swing widely.
Source: summary.json levels[].underneath.memory_activity_percent, matrix_unit_use_percent (AMD's device exporter)

MEASURED

The instruments cost little. Answer tokens a second with them, against the baseline run without: 55.3 against 56.2 for one stream (1.6% less), 745.9 against 782.7 for 32 (4.7%), 1,324.6 against 1,327.3 for 256 (0.2%). Same settings, same load, same requests per level; each level played once in each run.

MEASURED

Outside the card little happens. The machine's CPU is 10.5% to 10.9% busy, and the outside network carried at most 3,817 bytes one way during a level.

READING, NOT MEASUREMENT

My reading: one stream is bound by memory traffic, and many streams are bound by the kernels themselves. For one token the server reads the whole model and does very little arithmetic with it; with hundreds of streams the same reading serves them all, and what is left to wait for is the arithmetic. The pages that follow give the grounds, and a test of the reading.

The machine, and what this test looks at

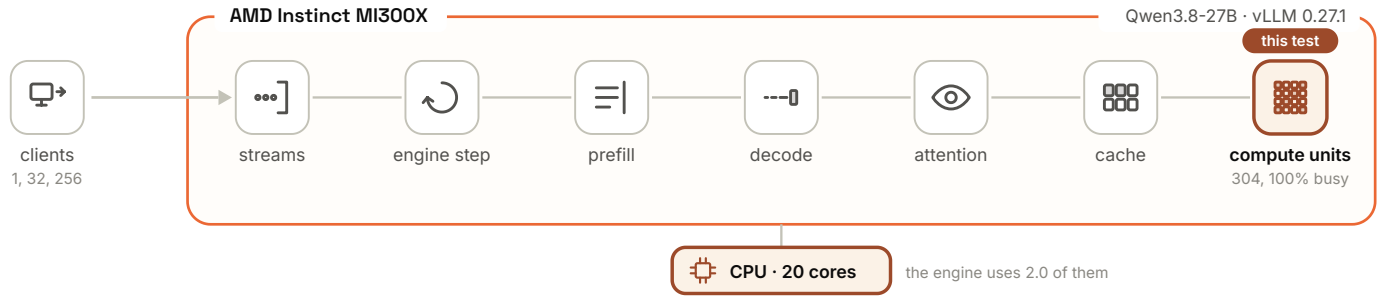

Figure 3 The card as every document in this series draws it. This test changes nothing in the server: it looks underneath, at the compute units, the memory beside them, and the CPU.
Source: FACTS.json; summary.json sweeps.random-1k-256.levels[streams=256].underneath

Three instruments and a sampler

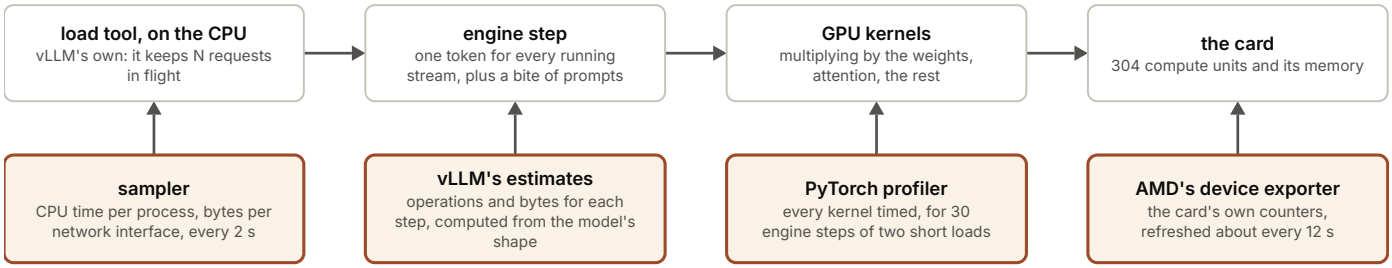

Figure 4 Top row: where the work happens, left to right. Bottom row: what watches it. An arrow from an instrument points at the part it reads.
Source: config.conf in summary.json (the server's switches), collect.py (the sampler), random-1k-256/samples.jsonl

1. The server starts with the baseline's settings plus switches that only add counters. The profiler is made available, not recording.
2. vLLM's own load tool keeps 1, then 32, then 256 requests in flight: 1,024 random tokens in, exactly 256 out, 4 requests per stream (at least 8), after 2 warm-up requests.
3. Beside it a sampler reads AMD's exporter, each vLLM process's CPU time, every network interface and the server's gauges every 2 s.
4. Then two short loads with the profiler recording 30 engine steps each: one stream, and 256.

Model	Qwen3.8-27B: 27.8 billion parameters, BF16 weights, 262,144 tokens of context
Card	one AMD Instinct MI300X: 304 compute units, 191.7 GiB of memory
Server	vLLM 0.27.1+rocm723, ROCm 7.2.3, PyTorch 2.11.0+gitd0c8b1f
Machine	a cloud GPU virtual machine: 20 virtual cores, Intel Xeon Platinum 8568Y+
This run	20261125T192827Z-probe, started 2026-11-25T19:28:40Z, server ready after 142 s

vLLM's estimates
Computed for every engine step from the model's shape: so many operations, so many bytes. Never read from the hardware.

AMD's device exporter
Read from the card. Its values are replaced every 11 to 16 s and cover one "reading" of unknown length: I compare levels, never read rates.

PyTorch profiler
Every kernel call timed on the GPU's own clock, for thirty engine steps: a sample, not the level.

LOAD	STREAMS	REQUESTS	FAILED	IT TOOK	PROMPT TOKENS	ANSWER TOKENS	WHAT IT IS FOR
level	1	8	0	37.0 s	8,192	2,048	the record
level	32	128	0	43.9 s	131,072	32,768	the record
level	256	1,024	0	197.9 s	1,048,576	262,144	the record
profiled	1	8	0	39.2 s	8,192	2,048	kernel times only
profiled	256	256	0	61.0 s	262,144	65,536	kernel times only

The load is made on the same machine as the server and shares its CPU cores; no request leaves it. Prompts are random tokens: nothing repeats, and the answers are not readable text.

The whole sweep, sample by sample

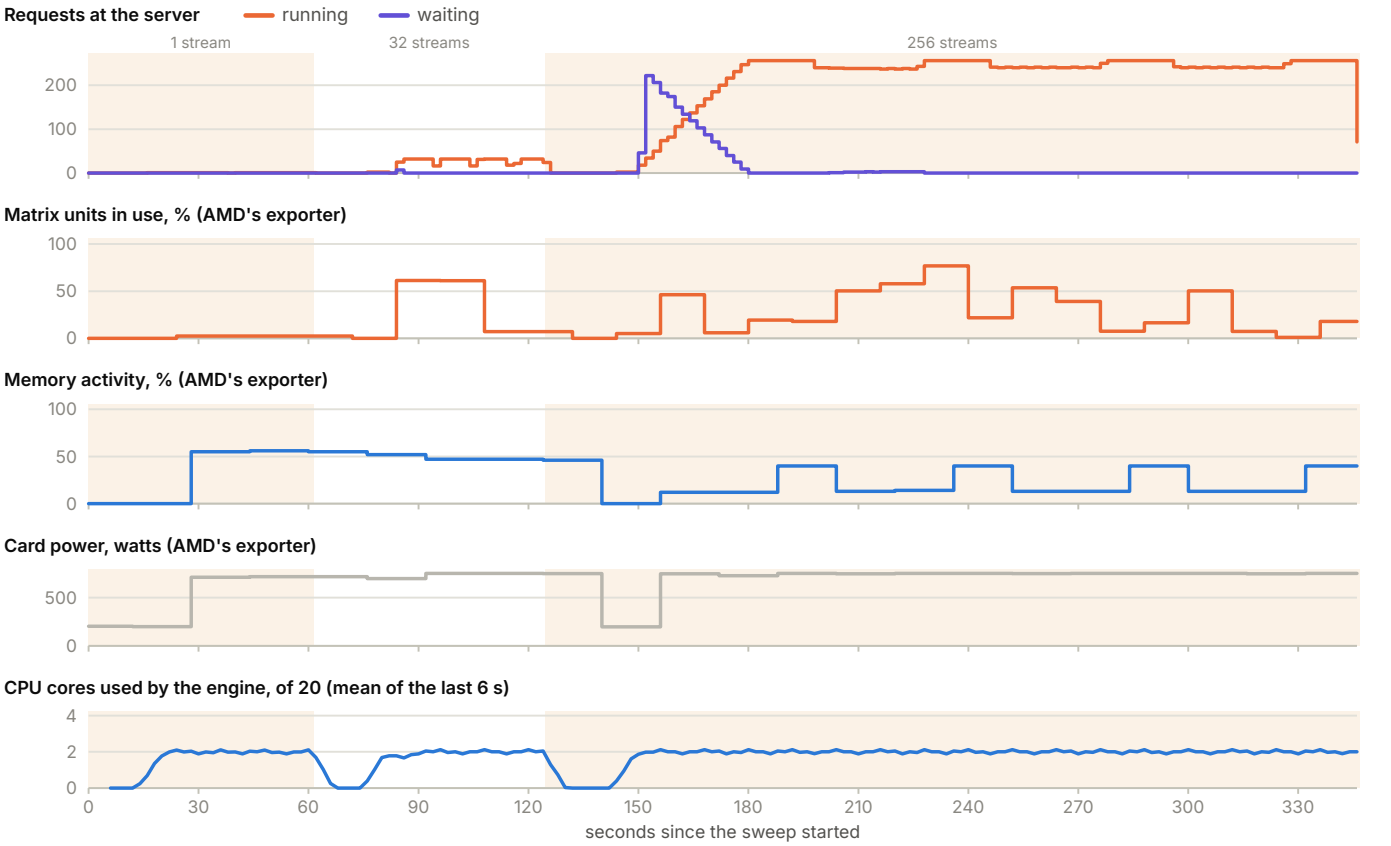

Figure 5 The sweep on the sampler's clock, one sample every 2 s. Each band is one run of the load tool: its start-up, two warm-up requests, then the level. The exporter's values are drawn as it holds them, 11 to 16 s at a time: at 32 and 256 streams the matrix units swing from reading to reading while the card stays at full power. The engine's CPU cores do not move with the load.
Source: random-1k-256/samples.jsonl; level marks from c1.t0 to c256.t1

The exact record

STREAMS	1	32	256
THE LOAD TOOL AND THE SERVER'S TIMERS (MEASURED)			
answer tokens a second	55.3	745.9	1,324.6
the same, per stream	55.3	23.3	5.2
with the prompt tokens	276.7	3,729.3	6,622.9
first token after, mean, s	0.14	2.54	6.23
ms per answer token	17.59	33.04	168.97
in line, s per request	0.00	0.41	3.28
reading the prompt, s	0.37	1.63	2.00
writing the answer, s	4.48	8.37	43.02
VLLM'S ESTIMATES, FROM THE MODEL'S SHAPE			
arithmetic, TFLOPS	12.7	170.6	302.8
of AMD's stated peak	0.9%	13.1%	23.2%
bytes read, TB a second	2.59	1.60	0.77
of AMD's stated peak	48.9%	30.2%	14.5%
bytes written, GB a second	2.6	35.0	62.2

STREAMS	1	32	256
AMD'S DEVICE EXPORTER, MEANS OVER THE LEVEL			
card busy	100.0%	100.0%	100.0%
matrix units in use	2.4%	28.7%	30.8%
vector units active	7.3%	21.8%	27.4%
memory activity	55.5%	46.9%	22.8%
occupancy	19.3%	17.3%	24.4%
power, watts	715	751	749
16-bit operations, billions *	13.0	160.0	271.7
GB fetched from memory *	3.24	1.93	1.54
GB written to memory *	0.011	0.185	0.357
THE SAMPLER: CPU AND NETWORK ACROSS THE LEVEL			
machine CPU busy	10.5%	10.7%	10.9%
CPU cores: the engine	2.01	1.99	2.01
CPU cores: the front end	0.02	0.04	0.07
CPU cores: the load tool	0.00	0.02	0.03
outside network, bytes in	374	1,166	3,300
outside network, bytes out	396	3,817	2,686

One column per level, from `summary.json`. * The exporter's operation and byte counts cover one reading of unknown length: compare the columns, do not read them as rates. Its "occupancy per active compute unit" read 1,323, 7.6 and 449, with no pattern, and is left out. The loopback and bridge interfaces of the machine counted 2.9 to 15.5 MB per level: those are the sampler's own reads, not the load.

vLLM's estimate, taken apart: the model is read once per engine step

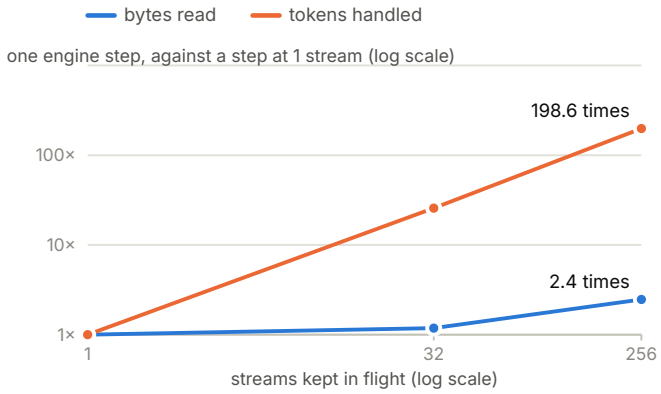

Figure 6 One engine step at 32 and at 256 streams, against a step at one stream. It carries hundreds of times the tokens and reads only a little more.

Source: the server's counters before and after each level (random-1k-256/c1, c32, c256 .before.prom, .after.prom)

SERVER'S COUNTERS, BY STREAMS	1	32	256
engine steps counted	2,560	1,292	1,322
tokens handled	12,800	166,400	1,313,280
tokens per engine step	5.0	128.8	993.4
GB read per engine step	46.9	55.3	115.8
GB read per token	9.37	0.43	0.12
billion operations per token	45.7	45.7	45.7

COMPUTED

Two numbers reproduce vLLM's bytes read at all three levels: **46.5 GB for every engine step**, whatever it carries, and 0.07 GB for every token in it (fitted on the 1- and 256-stream levels; the 32-stream level comes out 0.3% away). 46.5 GB is 84.7% of the weights the server loaded (51.1 GiB, which is 54.9 GB): the estimate reads the model's weights once per step. Its operations are simply so many per token.

Inside the 256-stream level: reading, then writing

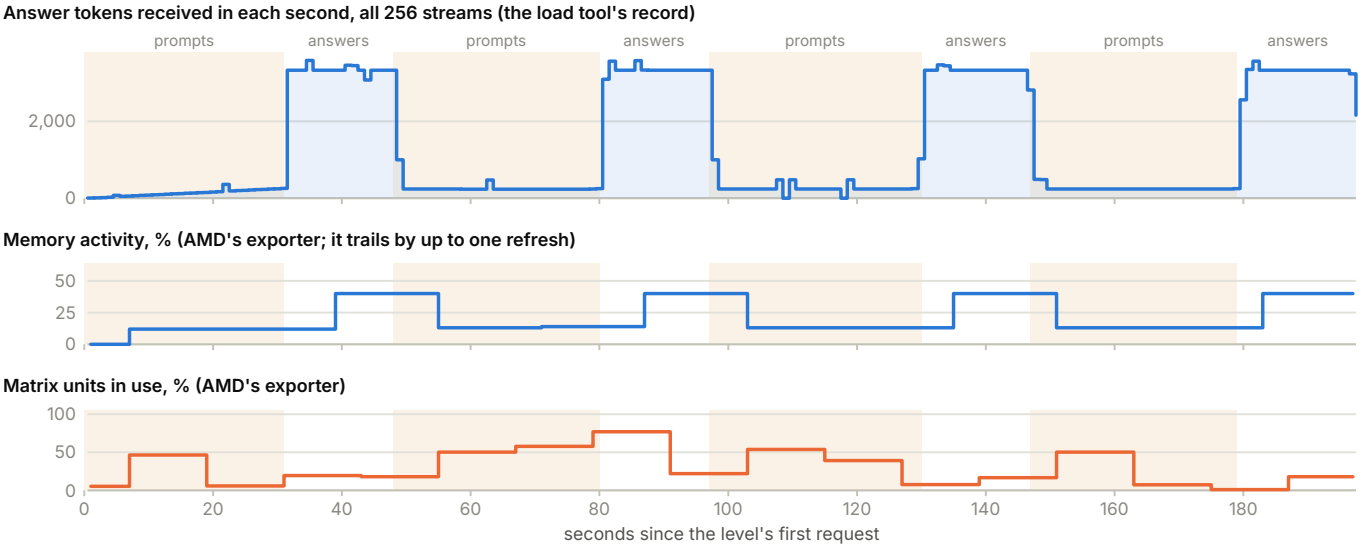

Figure 7 The 256-stream level second by second. Shaded: the engine is taking in prompts. Clear: it writes answers only. Memory activity reads high while answers are written and low while prompts are read; the matrix units mostly read higher in the prompt stretches, and swing. The two clocks are aligned to about a second; the exporter trails by up to one refresh.

Source: random-1k-256/c256.json (each request's first token and every gap after it); random-1k-256/samples.jsonl

THE 256-STREAM LEVEL	STRETCHES	SECONDS	OF THE LEVEL	ANSWER TOKENS IN A TYPICAL SECOND	FIRST TOKENS IN A TYPICAL SECOND	OF ALL ANSWER TOKENS
reading prompts	4	128	64.6%	239	8	11.7%
writing answers	4	70	35.3%	3,328	0	88.2%

MEASURED

The level is two loads taking turns. For 64.6% of its 198 seconds there is one engine step a second with 8 new prompts in it: 8,192 tokens, the step's limit. Every stream already answering gets one token from that step, about 239 answer tokens a second in all. In the other stretches the steps carry answers only: **3,328 answer tokens a second**, 13 for each of the 256 streams. The level's mean of 1,324.6 is the blend.

KEEP IN MIND

The summary's bytes read at 32 and 256 streams are a little high. Its counters also hold each level's two warm-up requests, which run before it in about 256 small steps, and it scales them by tokens: right for operations, not for bytes, which go mostly by the step. Without the warm-up steps I get 1.35 TB a second (25.5% of the stated peak) at 32 streams, not 1.60, and 0.71 TB a second (13.4% of the stated peak) at 256 streams, not 0.77. One stream is not affected; the direction is the same.

Where the GPU's time goes

One stream

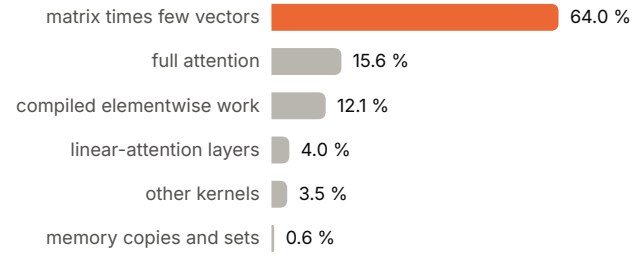

Figure 8 Share of GPU kernel time, 30 engine steps with one stream: nearly two thirds goes to multiplying the weights by the few vectors of one token.
Source: torch-profile/breakdown.txt, first trace (from the profiler's trace file)

256 streams

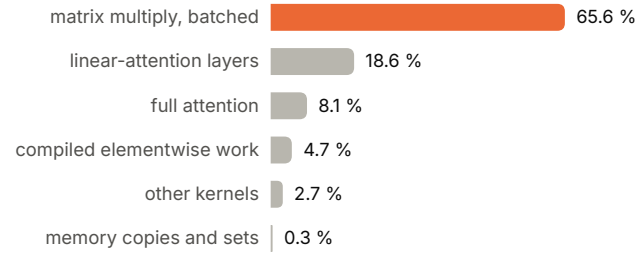

Figure 9 The same with 256 streams: two thirds again goes to the weights, now as batched matrix multiplies, and the linear-attention layers take almost a fifth.
Source: torch-profile/breakdown.txt, second trace

MEASURED

In both traces GPU kernels fill **100% of the traced span**: 0.548 s of kernels in 0.549 s with one stream (28,920 kernel calls), 6.085 s in 6.086 s with 256 (33,005 calls). The card does not wait between kernels, for the CPU or for anything else.

ESTIMATE

PyTorch's profiler counts the operations of the matrix multiplies from the shapes of the matrices. At 256 streams: 1,721 trillion operations in 3.106 s of GPU time over 1,550 calls, which is **554 TFLOPS while they run**, 42.6% of AMD's stated peak. They take 51.0% of the traced GPU time.



Figure 10 Arithmetic against AMD's stated peak for the card. Top two: vLLM's estimate for a whole level. Bottom: the profiler's count for the matrix multiplies alone, over the GPU time they ran. Busy all the time is not the same as at the peak.
Source: summary.json levels[].est_tflop_s; torch-profile/profiled-c256.profiler_out_0.txt (the row aten::mm); FACTS.json stated_peaks_used_as_divisors

The exact shares

1 STREAM: KERNEL FAMILY	SHARE	SECONDS	256 STREAMS: KERNEL FAMILY	SHARE	SECONDS
matrix times few vectors (wvSplitK_* kernels)	64.0%	0.351	matrix multiply, batched (rocBLAS Tensile kernels, Cijk_*)	65.6%	3.991
full attention	15.6%	0.086	linear-attention layers (gated delta rule, convolution)	18.6%	1.133
compiled elementwise work (norms, activations, gating)	12.1%	0.067	full attention	8.1%	0.494
linear-attention layers (gated delta rule, convolution)	4.0%	0.022	compiled elementwise work (norms, activations, gating)	4.7%	0.288
other kernels	3.5%	0.019	other kernels	2.7%	0.162
memory copies and sets	0.6%	0.003	memory copies and sets	0.3%	0.017

- **Matrix times few vectors**: a layer's weights times the handful of vectors one token brings. The whole matrix is read for very little arithmetic.
- **Matrix multiply, batched**: the same weights times hundreds or thousands of vectors at once.
- **Full attention**: the 16 layers that look back over every token of the context.
- **Linear-attention layers**: the other 48 of the 64, which keep a fixed-size state per stream.

NOT A RESULT

The two profiled loads are not throughput results. With the profiler recording they ran slower: 52.3 answer tokens a second at one stream and 1,074.8 at 256 (with 256 requests, not 1,024).

KEEP IN MIND

The 256-stream trace happened to hold both kinds of step: answers only, and steps with new prompts in them. The 1-stream table of the profiler was overwritten on the machine; its trace was kept, and the breakdown is from the traces.

Outside the card: the CPU and the network are almost idle

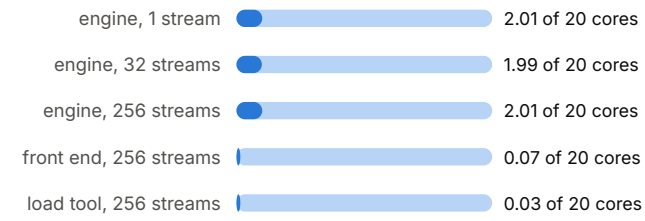

Figure 11 CPU cores in use by each vLLM process, against the machine's 20. The engine holds the same two cores with one stream as with 256.
Source: summary.json levels[].underneath.cpu_cores_in_use (CPU time of each process across the level)

LEVEL	CPU BUSY	NETWORK IN, BYTES	OUT, BYTES
1 stream	10.5%	374	396
32 streams	10.7%	1,166	3,817
256 streams	10.9%	3,300	2,686

MEASURED
A few thousand bytes on the outside network during a whole level: that is the remote session that drives the test, and whatever else the machine does on its own. The load tool reaches the server over the container's own loopback, which this run did not sample; later runs do.

READING, NOT MEASUREMENT
Two cores busy whenever a stream is running, and no more with 256 than with one: that looks like threads that spin while they wait for the card, not like work that grows with the load. In both profiler traces GPU kernels fill the traced span: the card is not waiting for the CPU.

A test of the reading: take compute units away

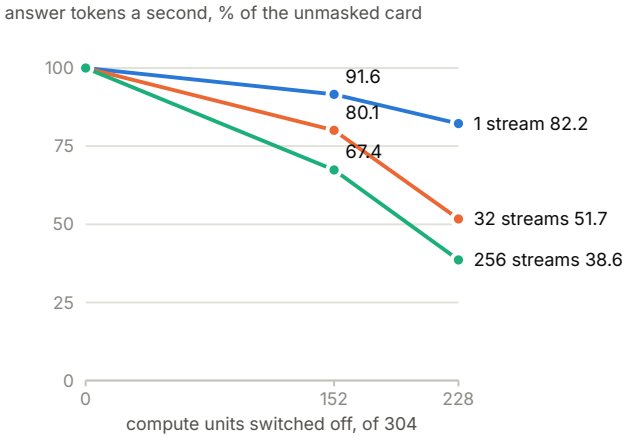

Figure 12 The same load with half and with three quarters of the card's compute units switched off, as a share of this run's output. One stream hardly notices; 256 streams lose most.
Source: summary.json of this run and of 20261125T194207Z-cu152 and 20261125T194900Z-cu76 (sweeps.random-1k-256.levels[].output_tok_s)

COMPUTED
With 152 of 304 compute units left, one stream loses 8.3% and 256 streams lose 32.5%. With 76 left it is 17.7% and 61.3%.

READING, NOT MEASUREMENT
That is what the reading predicts. A load that waits for memory keeps most of its speed when arithmetic is taken away; a load that waits for arithmetic does not. Each of the two masking runs has its own document.

COMPUTE UNITS ON	1 STREAM	32 STREAMS	256 STREAMS
304 (this run)	55.3	745.9	1,324.6
152	50.7	597.8	893.8
76	45.5	386.1	512.2

Answer tokens a second. One setting apart: the mask on the server's environment. The masking runs used 2 requests per stream (512 at 256 streams) where this run used 4 (1,024); at one stream all three ran 8.

Not results, and limits

NOT A RESULT
The two profiled loads ran slower with the profiler recording (52.3 and 1,074.8 answer tokens a second). They are here for their kernel times only.

KEEP IN MIND
What this does not say: that the card could do four times as much at 256 streams. The shares are of AMD's stated peaks, which nothing here confirms, and vLLM's figures are arithmetic on the model's shape. They say what kind of load it is, not how much is left.

- One sweep, each level played once. Prompts are random tokens: this is speed and behaviour, nothing about the quality of answers.
- The exporter's level means are means of a few readings that differ widely.
- The 256-stream level is two loads taking turns; its means describe the blend.
- vLLM's bytes read at 32 and 256 streams are a little high in the summary, as the page on the swap explains.
- The profiler saw thirty engine steps of each load. The 1-stream table was overwritten; its trace was kept.

What I take from it

READING, NOT MEASUREMENT

◌ One stream is bound by memory traffic: about half the stated ceiling, and nearly two thirds of GPU time in kernels that read a whole matrix for a few vectors. Many streams are bound by the kernels themselves: the card is always busy, its matrix multiplies run at roughly two fifths of the stated peak by the profiler's count, and about a fifth of the time goes to linear-attention kernels. The CPU is not the limit. If that holds, one stream gains from fewer bytes per token and many streams from faster kernels; more CPU would change nothing.

The grounds, side by side

INSTRUMENT, AND WHAT IT REPORTS	1 STREAM	256 STREAMS
THE MEMORY SIDE		
vLLM's estimate: bytes read, of AMD's stated peak	48.9%	14.5%
AMD's exporter: memory activity	55.5%	22.8%
profiler: the largest share of GPU kernel time	matrix times few vectors, 64.0%	matrix multiply, batched, 65.6%
THE ARITHMETIC SIDE		
vLLM's estimate: arithmetic, of AMD's stated peak	0.9%	23.2%
AMD's exporter: matrix units in use	2.4%	30.8%
profiler: its matrix multiplies while they run, of AMD's stated peak	no table kept	42.6%
BUSY, AND HELD UP BY NOTHING OUTSIDE THE CARD		
AMD's exporter: card busy	100.0%	100.0%
profiler: GPU kernels, of the traced span	100%	100%
sampler: CPU cores used by the engine, of 20	2.01	2.01

● measured: read from a run file · ○ computed: arithmetic on measured values · ◑ estimate: vLLM's, from the model's shape · ◌ my reading: interpretation, not measurement

Words used

Engine step
one pass of the model over a batch: one new token for every decoding stream, plus a bite of prefill.

Matrix units in use
(`amd_gpu_prof_tensor_active_percent`, "MFMA utilization") how much of the time the card's matrix-multiply hardware is busy, as AMD's exporter reports it.

Memory activity
(`amd_gpu_umc_activity`) AMD's exporter value "memory engine usage in percent".

TFLOPS
trillions of arithmetic operations per second. Here always vLLM's estimate from the model's shape (`--enable-mfu-metrics`), not a hardware counter. It counts attention in all 64 layers where this model has it in 16: 1.7% high for short prompts, over half too high from 128,000 tokens of context.

Every number in this document was read from these files at build time (sha256, first 12 characters):

`FACTS.json` 160f5ca0d495 · `GLOSSARY.md` 98323be796af · `runs/20261125T184034Z-recipe/summary.json` f93dd689d428 · `runs/20261125T192827Z-probe/random-1k-256/c1.after.prom` 04337507a2c2 · `runs/20261125T192827Z-probe/random-1k-256/c1.before.prom` 9a8e580b65c3 · `runs/20261125T192827Z-probe/random-1k-256/c1.t0` e3ff8c964b8e · `runs/20261125T192827Z-probe/random-1k-256/c1.t1` 3c34cdd192e1 · `runs/20261125T192827Z-probe/random-1k-256/c256.after.prom` ce6eb11c3507 · `runs/20261125T192827Z-probe/random-1k-256/c256.before.prom` 845e4233ffa0 · `runs/20261125T192827Z-probe/random-1k-256/c256.json` 2306cd81b021 · `runs/20261125T192827Z-probe/random-1k-256/c256.t0` 58d92a41a894 · `runs/20261125T192827Z-probe/random-1k-256/c256.t1` ffca5acf21be · `runs/20261125T192827Z-probe/random-1k-256/c32.after.prom` 845e4233ffa0 · `runs/20261125T192827Z-probe/random-1k-256/c32.before.prom` 04337507a2c2 · `runs/20261125T192827Z-probe/random-1k-256/c32.json` e5ea8f77b051 · `runs/20261125T192827Z-probe/random-1k-256/c32.t0` f040d3d66eda · `runs/20261125T192827Z-probe/random-1k-256/c32.t1` c1a4dfc065d3 · `runs/20261125T192827Z-probe/random-1k-256/samples.jsonl` 9ef52684e4fc · `runs/20261125T192827Z-probe/summary.json` 064f25b0020b · `runs/20261125T192827Z-probe/torch-profile/breakdown.txt` 0a6e4963bcfc · `runs/20261125T192827Z-probe/torch-profile/profiled-c1.start` ee8e9c04c94f · `runs/20261125T192827Z-probe/torch-profile/profiled-c1.stop` 4ae3679d2c9c · `runs/20261125T192827Z-probe/torch-profile/profiled-c256.profiler_out_0.txt` ff1aa0b043ea · `runs/20261125T192827Z-probe/torch-profile/profiled-c256.start` 77893dc3799d · `runs/20261125T192827Z-probe/torch-profile/profiled-c256.stop` 60edd0ffa4d3 · `runs/20261125T194207Z-cu152/summary.json` 1bdedd55da09 · `runs/20261125T194900Z-cu76/summary.json` 3015428c9534

107 values bound by path, 12 figures drawn from data. built 2026-11-26 02:37Z, explainer-pdf-system v2.



CREATED AND DIRECTED BY

Peter Styk

HPC & AI Platform Engineer

GPU inference, data and performance

linkedin.com/in/peterstyk · github.com/styk-tv

Questions about a run? Connect on LinkedIn.

LinkedIn