A ceiling at 588 streams, and about 4,000 answer tokens a second

Short prompts, long answers, and twice as many streams every forty seconds. Output stops growing near 512 streams, the server never runs more than 588 at once, and everything above waits in a line that has no limit.

Qwen3.8-27B · one AMD Instinct MI300X · vLLM 0.27.1 **Run** 20261125T201346Z-series **Changed** nothing: baseline settings, 1,024 stream slots
Load 64 tokens in, 512 out, 1 to 2,048 streams

I asked one server for more and more streams at once, doubling every 40 seconds, with a prompt of 64 tokens and an answer of 512. Output grew until about 512 streams and then stayed near **4,096 answer tokens a second**. The server never ran more than 588 streams at once: its cache was full. Whatever I asked for above that waited.

4,096
answer tokens a second at the top, all streams together

588
streams running at once, at most, of 1,024 allowed

1,460
requests waiting at most, none refused

145 s
or more: the median wait for a first token at the top stair, where 236 of 1,176 requests never got one

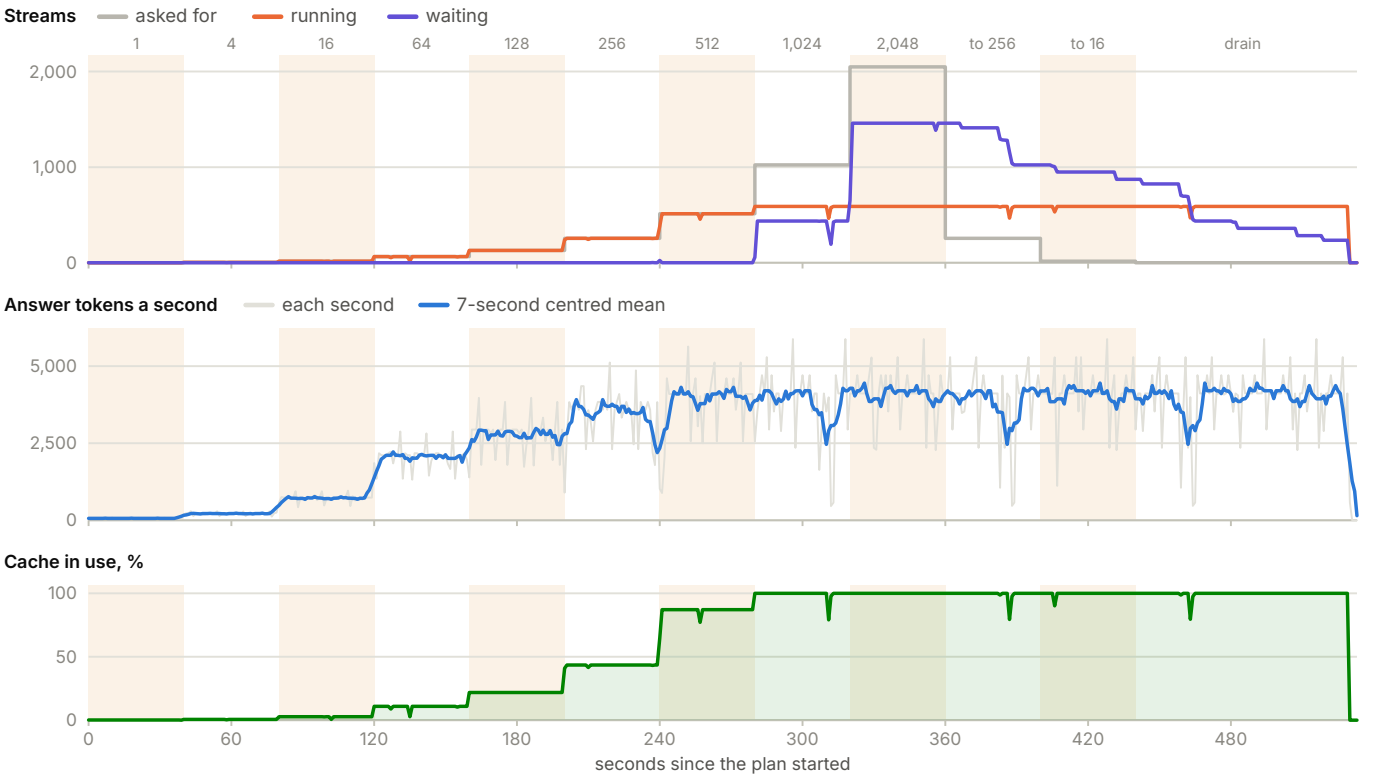


Figure 1 The whole plan, second by second. Top: streams asked for (grey steps), running and waiting. Middle: answer tokens a second, each second in grey and a 7-second centred mean in blue. Bottom: the cache. Shaded bands are the stairs, labelled with the streams asked for.
Source: plan-staircase-short/timeline.json (vLLM's counters and gauges sampled every second)

MEASURED

Running streams follow the request up to 512, then stop at 588 with the cache 99.9% full. The server's own reason for every waiting request is "capacity".

MEASURED

Nothing was refused and nothing failed: all 3,361 requests got HTTP 200. 2,537 ended inside the plan; 824 were still open when its time ran out and were closed by the load generator.

The machine, and the two parts this test leans on

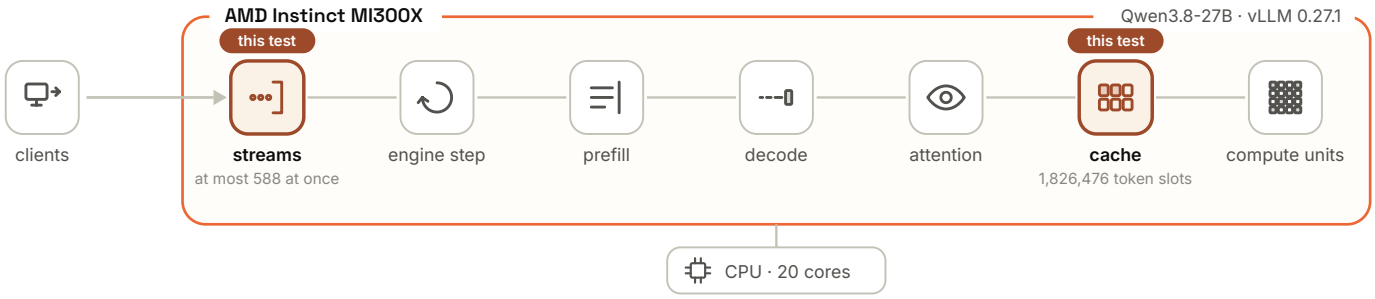


Figure 2 The card as every document in this series draws it: seven parts wired left to right, the clients outside, the host CPU below. This test is about how many streams fit and what the cache does when they do not.
Source: FACTS.json; summary.json server.kv_cache_tokens

One loop, repeated

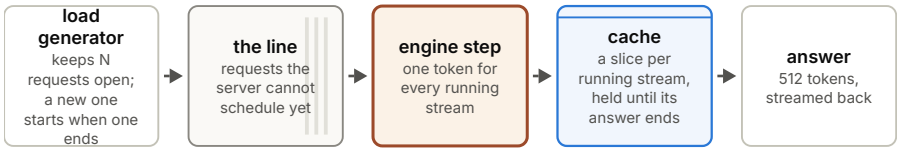


Figure 3 A closed loop: the load generator keeps a fixed number of requests open. Each engine step gives every running stream one more token; a stream holds its slice of the cache until its answer ends.
Source: loadgen.py; plans/staircase-short.json

1. The plan plays eleven stages back to back, with no pause: streams go 1, 4, 16, 64, 128, 256, 512, 1,024, 2,048, then down to 256 and 16.
2. In each stage the generator keeps that many requests open. Going down it cuts nothing: open requests are left to finish.
3. Every prompt is 64 random token ids, so no two prompts share a prefix; every answer is forced to 512 tokens.
4. Beside it a sampler reads the server's counters, the card and the CPU once a second.

Settings

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	20261125T201346Z-series, started 2026-11-25T20:13:59Z, server ready after 94 s

KEEP IN MIND

The load is made on the same machine as the server and shares its CPU cores. The generator runs inside the server's container and talks to it on the loopback address: no request leaves the machine.

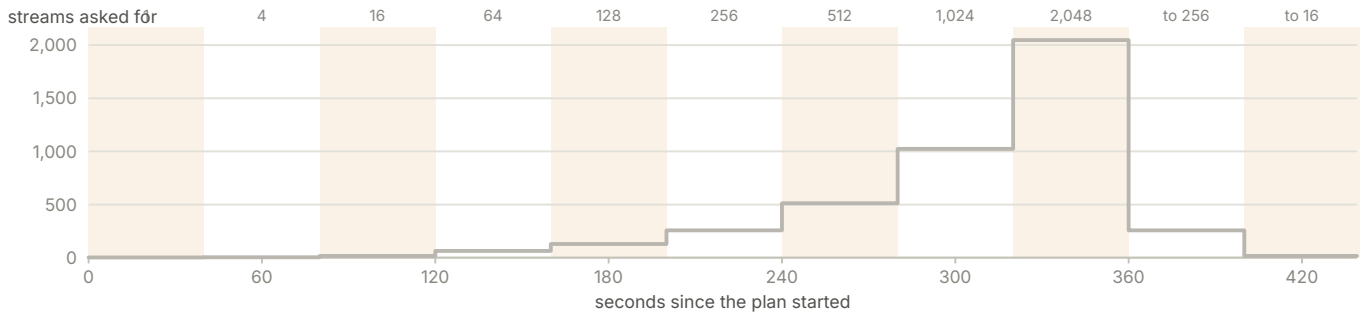


Figure 4 The plan: streams asked for over time. Each stair lasts 40 seconds.
Source: plan-staircase-short/timeline.json (target)

More streams, more answers, until the card is busy

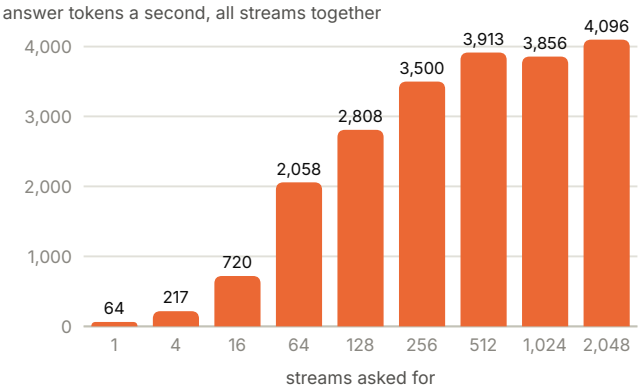


Figure 5 Answer tokens a second for all streams together, by stair (going up).
Source: summary.json plans.plan-staircase-short.stages[].server_output_tok_s

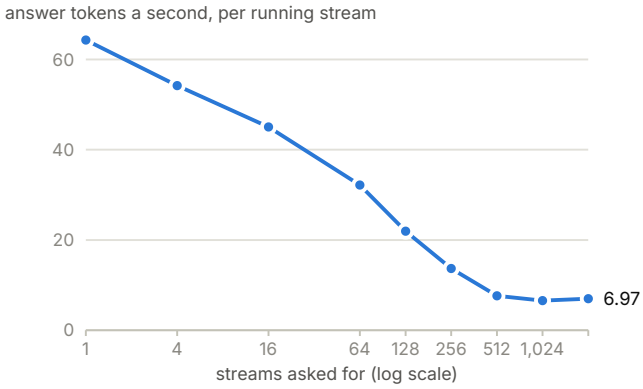


Figure 6 The same output divided by the streams running: what one stream gets.
Source: server_output_tok_s / running_max, computed

MEASURED

One stream alone gets 64.3 answer tokens a second. At 512 streams the card gives 3,913 in all, 60.8 times what one stream gets, and each stream 7.6.

MEASURED

The engine makes 64.3 steps a second for one stream and 7.0 at the top, where each step handles about 624 tokens.

The exact record

STREAMS ASKED FOR	RUNNING AT MOST	WAITING AT MOST	CACHE AT MOST, %	ANSWER TOKENS/S	PER RUNNING STREAM	ENGINE STEPS/S	TOKENS PER STEP	FIRST TOKEN, MEDIAN S	TFLOPS, EST.
1	1	0	0.2	64.3	64.3	64.32	1.1	0.05	3.3
4	4	0	0.7	216.8	54.2	54.43	4.5	0.10	11.3
16	16	0	2.7	720.1	45.0	45.18	17.7	0.14	37.1
64	64	0	10.9	2,057.9	32.2	32.26	70.7	0.42	105.6
128	128	0	21.7	2,808.4	21.9	22.02	141.0	0.47	143.8
256	256	2	43.5	3,499.7	13.7	13.74	301.9	1.02	191.1
512	512	25	87.0	3,913.1	7.6	7.66	584.0	1.20	206.2
1,024	588	436	99.9	3,855.6	6.6	6.58	717.3	33.63	217.3
2,048	588	1,460	99.9	4,096.5	7.0	6.97	624.1	≥ 144.63	202.0
to 256	588	1,460	99.9	3,826.0	6.5	6.53	698.8	-	210.3
to 16	588	1,024	99.9	4,080.0	6.9	6.95	624.3	-	201.3

One row per stair, from `summary.json`. Output and steps are the server's counters over the stair; running, waiting and cache are the most seen in it; the first-token time is the median for requests sent in that stair. Going down ("to 256", "to 16") the server is still working through the line, so those rows look like the top. Stage rows span 39 s of each 40 s stage: the first to the last one-second sample.

The ceiling is the cache, not the stream setting

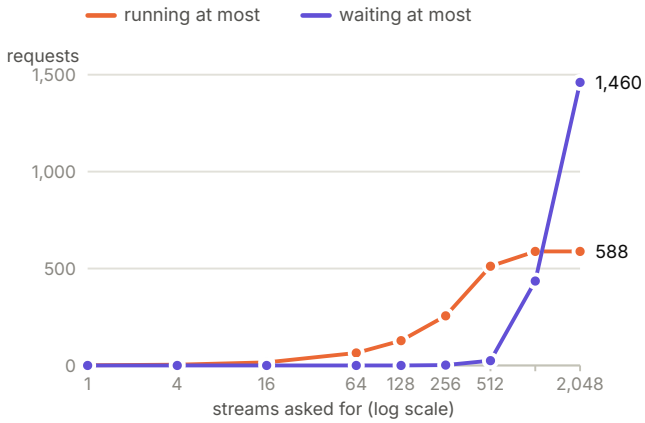


Figure 7 Asked for against delivered: the most streams running and the most waiting, by stair.
Source: summary.json stages[].running_max, waiting_max

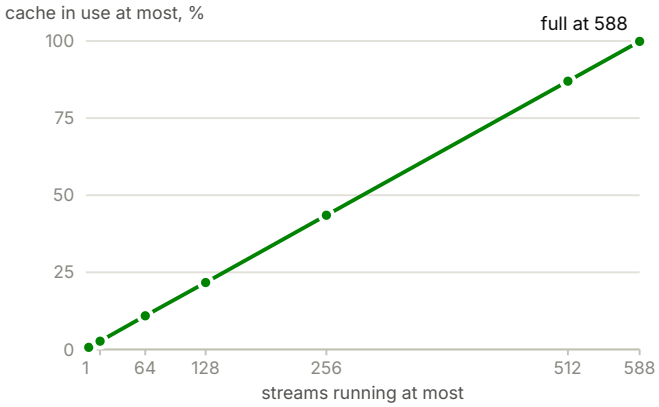


Figure 8 The cache against the streams running: it is full when 588 run.
Source: summary.json stages[].cache_in_use_max, running_max

MEASURED

The server allowed 1,024 streams and ran 588. Its cache holds 1,826,476 token slots; 588 streams of at most 576 tokens are about 338,688 tokens. So with the cache full, each running stream takes about 3,106 slots: several times its own tokens.

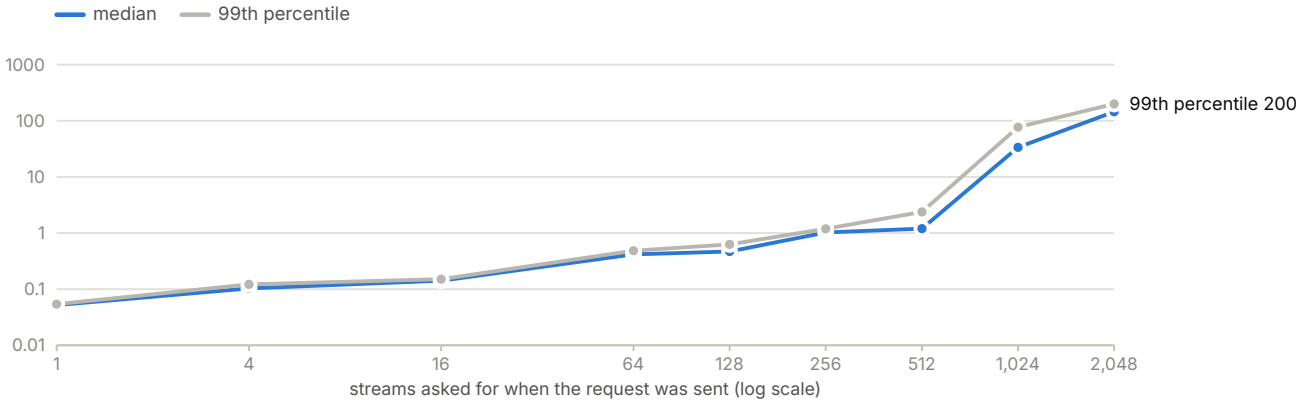


Figure 9 What the line costs a client: seconds from sending a request to its first token, by the stair in which it was sent. Both axes are logarithmic. At the top stair the median is a lower bound: requests that never got a first token count with the wait they had when the plan ended.
Source: requests.jsonl (t_first - t_sent), in summary.json stages[].first_token_s_median_at_least and _p99

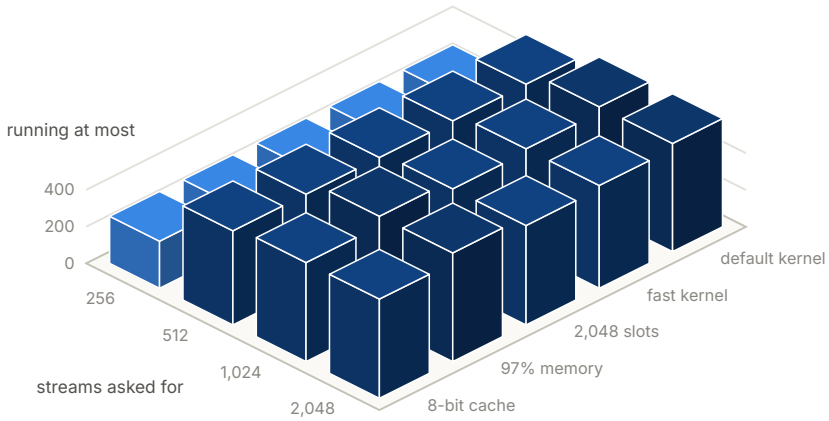
READING, NOT MEASUREMENT

A stream seems to hold a fixed amount of state in the cache whatever its length: roughly 2,500 to 2,800 slots besides its tokens. I derived that from two maxima and the cache size; I did not find it in vLLM's source. This model keeps a fixed-size state per stream in most of its layers, which would explain it.

KEEP IN MIND

The line is first come, first served, and has no limit: I found no setting in this vLLM for a longest line, for refusing work, or for a request timeout. Waiting is the only thing an overloaded server does to a client.

Five servers: the streams follow the cache



	streams asked for 256	512	1,024	2,048
default kernel	256	512	588	588
fast kernel	256	512	559	559
2,048 slots	256	512	541	541
97% memory	256	512	591	591
8-bit cache	256	512	539	539

darker = more: streams running at most

Figure 10 The most streams running at once, by stair, on five servers. A view only: the exact values are in the matrix beside it.
Source: cross/staircase.json (each run's summary.json)

Figure 11 The same values, exactly.
Source: cross/staircase.json

Rows: this run (default attention kernel); the same on the fast kernel; that with 2,048 stream slots instead of 1,024; that with 97% of card memory instead of the default 92%; that with an 8-bit cache. The last two also have 2,048 stream slots. The four variants ran the top four stairs only, on an empty server.

SERVER	CACHE TOKEN SLOTS	MOST STREAMS RUNNING	SLOTS PER RUNNING STREAM
default kernel (this run)	1,826,476	588	3,106
fast kernel	1,734,958	559	3,104
fast kernel, 2,048 stream slots	1,679,893	541	3,105
fast kernel, 97% of card memory	1,835,008	591	3,105
fast kernel, 8-bit cache	3,309,759	539	6,141

MEASURED

The number of streams that fit follows the size of the cache. With a 16-bit cache every running stream takes about 3,105 slots on all four servers. More memory bought proportionally more streams: against the same server at the default share, 9.2% more cache and 9.2% more streams. Stream slots cost cache: 541 streams ran with 2,048 slots where 559 ran with 1,024.

MEASURED

The 8-bit cache reports 1.9 times the token slots of the same server with a 16-bit cache, and each running stream then takes about twice as many: 539 streams ran against 541. No more streams fit.

Words used

Stream
one request being answered, from the moment it is sent until its last token.

Closed loop
the load keeps N requests in flight; a new one starts only when one ends. The server sets the pace.

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

Cache
(KV cache) the GPU memory where the server keeps what the model computed for every token of every open stream, so it need not be recomputed.

Capacity
(waiting reason) the server's label for a request it cannot schedule now, because the cache or the step's token budget is used up. Reported in `vllm:num_requests_waiting_by_reason`.

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.

What the card was doing

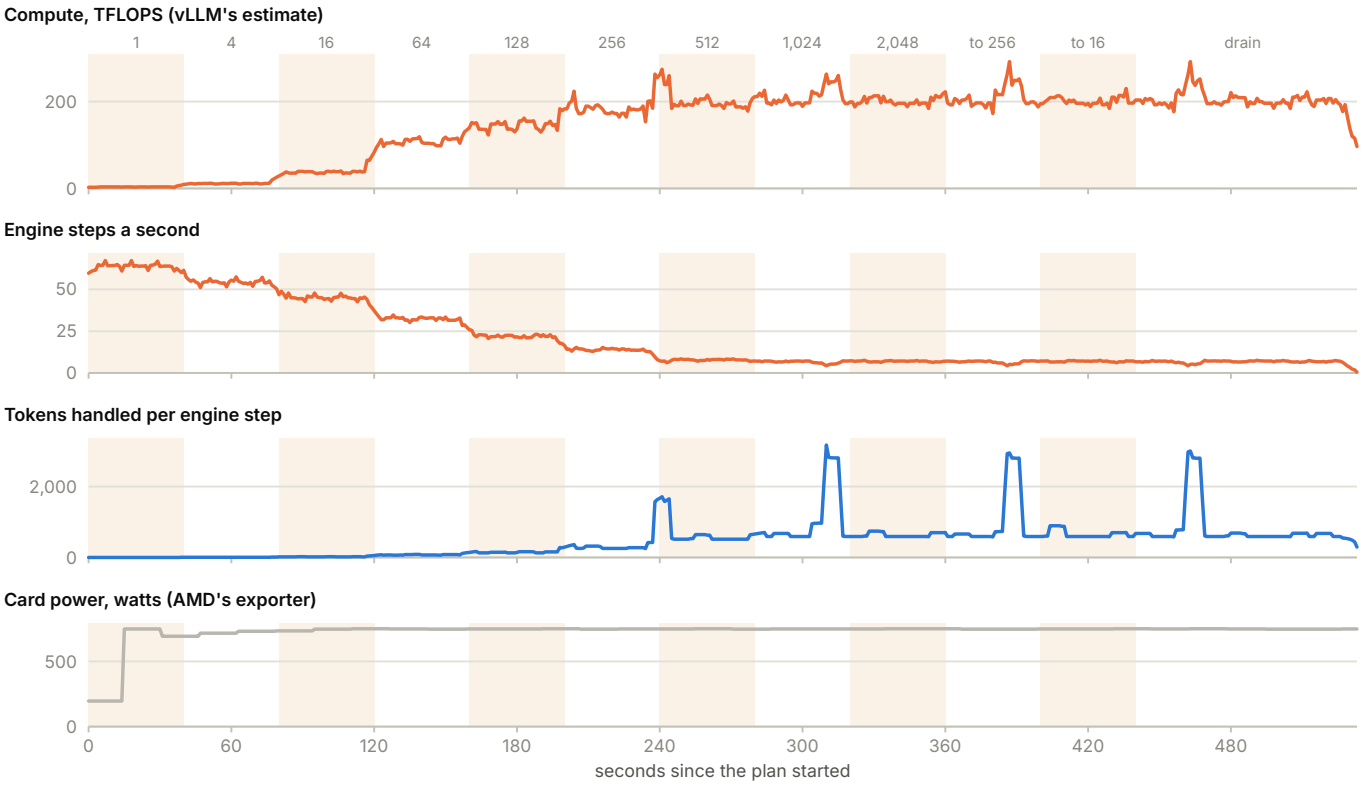


Figure 12 Underneath the same plan. Compute is vLLM's estimate from the model's shape, not a hardware counter; steps and tokens per step are the server's counters (7-second centred means); power is AMD's device exporter, which refreshes every 11 to 16 s.
Source: plan-staircase-short/timeline.json

What I take from it

- ● About 4,096 answer tokens a second is what this card gives with short prompts, from 512 streams up.
- ● 588 streams is the most it runs at once here; the cache decides that.
- ◑ vLLM's estimate of the compute at the top is 202 TFLOPS, 15.5% of AMD's stated peak for the card.
- ◌ Past the ceiling a client pays only in waiting: at least 145 s at the median for the top stair.

Not results, and limits

NOT A RESULT

The 824 requests "cut at the end of the plan" are not refusals: the generator closed them when the plan's 90 s drain ended.

- Prompts are random tokens: nothing repeats, and answers are not readable text.
- One plan, played once. The stairs going down are not a second measurement of those levels.

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

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

`FACTS.json` 160f5ca0d495 · `GLOSSARY.md` 98323be796af · `cross/staircase.json` 176e5a3b98b6 · `plans/staircase-short.json` 6a2be1bf2346 · `runs/20261125T201346Z-series/plan-staircase-short/timeline.json` 477032b70206 · `runs/20261125T201346Z-series/summary.json` aab3940eb12e · `runs/20261125T215049Z-series-fa/summary.json` ec0c01913c37 · `runs/20261125T224501Z-series-kvfp8/summary.json` 561b6d9e405a · `runs/20261125T225329Z-series-mem97/summary.json` ab4cc85f7ebd · `runs/20261126T013018Z-series-fa-seqs2048/summary.json` 94ecccaf84b3

74 values bound by path, 12 figures drawn from data. built 2026-11-26 02:39Z, 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