A plan that missed its goal: long prompts loaded too slowly to fill the cache

The plan was to fill the cache with 32,000-token prompts and push past it. The cache peaked at 74.1% and not one of the 80 requests ended inside the plan. As a test of a full cache it is not a result. What it records is how slowly long prompts load on the default attention kernel.

Qwen3.8-27B · one AMD Instinct MI300X · vLLM 0.27.1 **Run** 20261125T201346Z-series

Changed nothing: baseline settings, default attention kernel **Load** 32,000 tokens in, 256 out, 8 to 80 sessions

This plan was meant to fill the cache and go past it: prompts of 32,000 tokens, more and more of them open at once, up to 80. It did not get there. I had sized it on the pace at which this server reads short prompts; these it read at 1,103 to 2,759 tokens a second. The cache peaked at 74.1%, and not one of the 80 requests ended inside the plan's ten minutes.

74.1%
of the cache in use at most: the plan needed more than all of it

0
of 80 requests ended inside the plan; none was refused

2,759
prompt tokens read a second at best; the plan counted on 5,300

47 s
median wait for a first token with 8 sessions open; 240 s with 32

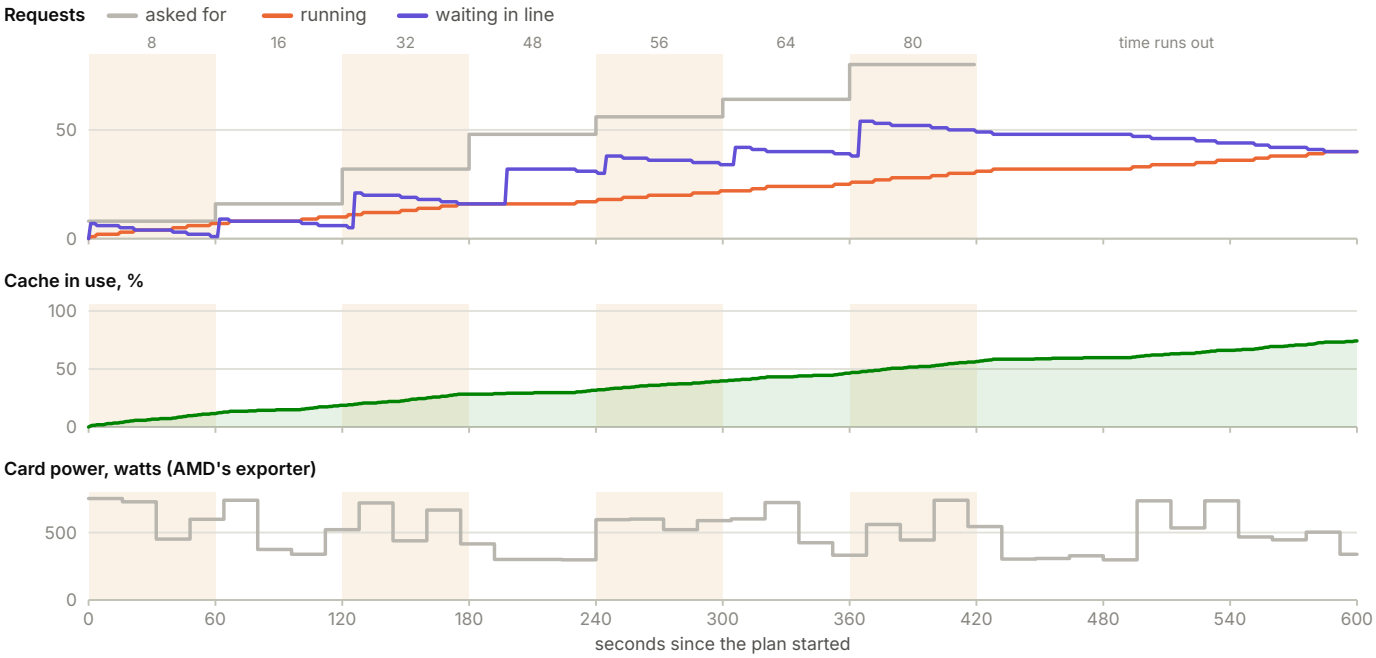


Figure 1 The whole plan, second by second. Top: sessions asked for (grey steps), requests running and waiting. Middle: the cache, which never fills. Bottom: the card's power, which sits well under the 750 W it draws when it is busy. Shaded bands are the stairs, labelled with the sessions asked for; in the last band the generator waits and then closes what is open.
Source: plan-kv-pressure/timeline.json (vLLM's gauges, AMD's device exporter)

NOT A RESULT

As a test of a full cache this plan is not a result: the cache never filled and nothing was thrown out (the preemption counter stayed at 0). The plan that does overfill the cache is the cache-growth plan of this run.

NOT A RESULT

All 80 requests were "cut at the end of the plan": still open when the generator's time ran out, and closed by it. They are not refusals: every one got HTTP 200. 39 had received a first token, 41 had not.

Long prompts load at well under half the pace the plan counted on

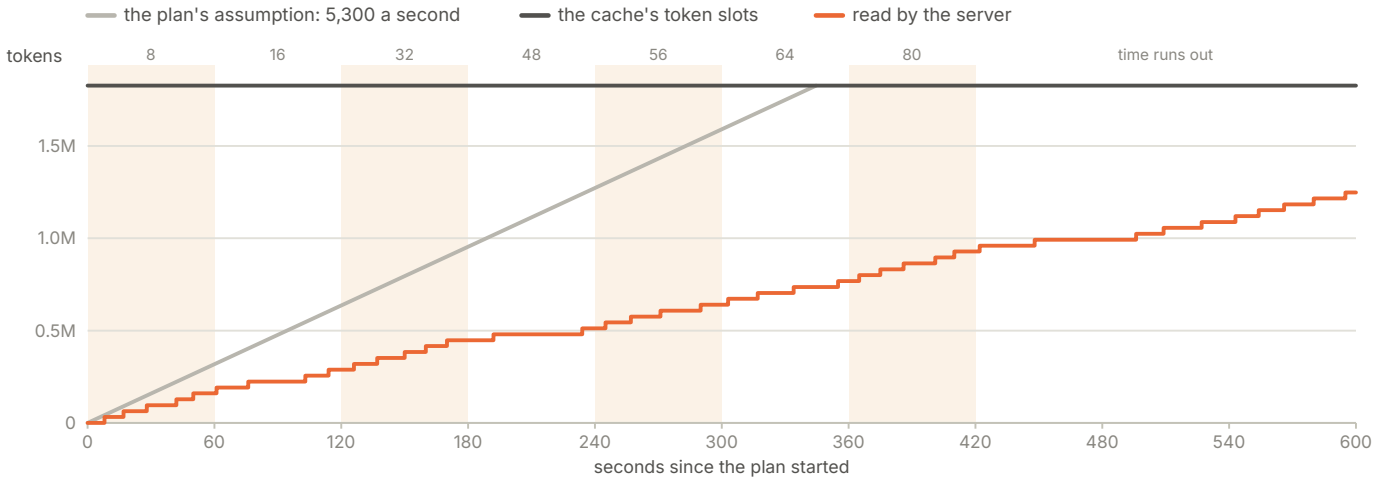


Figure 2 Prompt tokens read, counted up. Grey: the pace the plan was sized on, which would have filled the cache's token slots (the dark line) before the fifth stair. Orange: what the server read. It steps, because the counter moves when a whole prompt is done.
Source: plan-kv-pressure/timeline.json (vllm:prompt_tokens_total per second); summary.json server.kv_cache_tokens; the assumed pace is README section 18

The exact record

SESSIONS ASKED FOR	SENT	ENDED WELL	RUNNING AT MOST	WAITING AT MOST	CACHE AT MOST, %	PROMPT TOKENS/S	ENGINE STEPS/S	TOKENS PER STEP	FIRST TOKEN, MEDIAN S	SENT, NO FIRST TOKEN	TFLOPS, EST.	WATTS
8	8	0	7	7	11.6	2,758	0.34	8,003	47	0	241	635
16	8	0	10	9	18.2	2,207	0.24	9,150	96	0	145	496
32	16	0	16	21	28.3	2,758	0.34	8,012	240	0	212	586
48	16	0	17	32	31.6	1,103	0.12	9,158	≥ 420	9	72	323
56	8	0	21	38	39.2	2,206	0.28	8,018	≥ 360	8	169	575
64	8	0	25	42	46.4	2,206	0.24	9,165	≥ 300	8	145	553
80	16	0	30	54	55.9	2,759	0.34	8,027	≥ 240	16	211	545
none: time runs out	0	0	40	49	74.1	1,788	0.21	8,683	-	0	126	465

One row per stair, from `summary.json`. Running, waiting and cache are the most seen in the stair; prompt tokens and steps are the server's counters over it; the first-token time is the median of the requests sent in that stair that got one. A stage row spans 59 s of a 60 s stair. TFLOPS is vLLM's estimate, and for prompts this long about a quarter too high: it counts attention in all 64 layers and this model has it in 16. Watts are AMD's exporter, a new reading every 16 s.

MEASURED

Over the whole plan the server read 1,248,000 prompt tokens, which is 39 whole prompts: 2,080 a second on average. Its engine made 0.12 to 0.34 steps a second, each of 8,003 to 9,165 tokens: three seconds and more for one step.

MEASURED

While one prompt has the step's token budget the others wait, and the server gives "capacity" as its reason with the cache 11.6% full: with 8 sessions asked for, 7 ran at most and 7 waited at most. The wait for a first token grows with the line: 47 s at the median with 8 sessions, 96 s with 16, 240 s with 32.

MEASURED

The card was not busy. It drew 323 to 635 W as stage means, where it draws about 750 W under the other loads of this run, and its matrix units were in use 0.4 to 12.0% of the time.

READING, NOT MEASUREMENT

My reading: with the default attention kernel a long prompt is slow to read because every bite looks back over all that came before it, and between bites the card waits. What it waits for I did not find here. How long prompts load on the fast kernel is measured in another run, on other plans.

The machine, and the plan's mistake

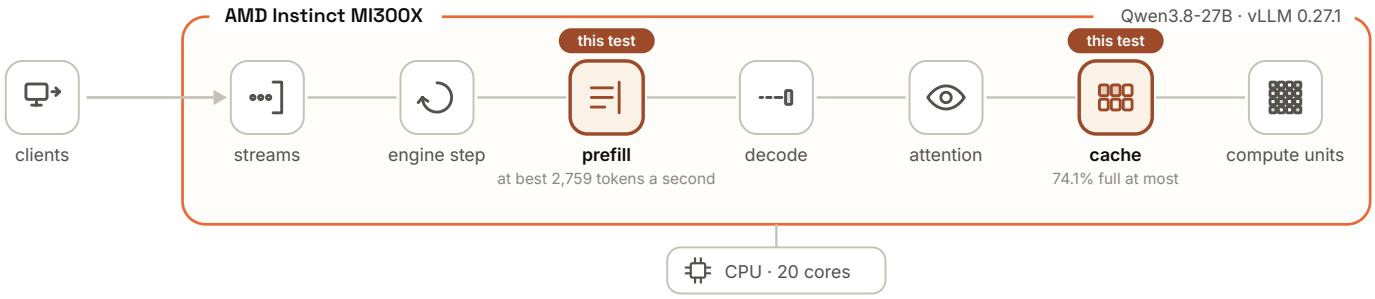


Figure 3 The card as every document in this series draws it. The plan aimed at the cache; what it measured is prefill, the reading of the prompt.
Source: FACTS.json; summary.json plans.plan-kv-pressure.stages[].server_prompt_tok_s, cache_in_use_max

How it was run

1. A closed loop: the load generator keeps 8, then 16, 32, 48, 56, 64 and 80 requests open, 60 seconds each, then waits 180 seconds and closes what is still open.
2. Every prompt is 32,000 random token ids; every answer would be 256 tokens.
3. The plan's own sum: the cache holds about 1.83 million tokens, so about 56 such requests fit, and the stairs go past that.
4. Baseline settings, default attention kernel. Beside it a sampler read the server's counters, the card and the CPU once a second.

KEEP IN MIND

The mistake was mine: I took the pace of reading short prompts for the pace of reading long ones. At 5,300 tokens a second the cache's 1,826,476 token slots fill in 5 min 45 s; at the pace measured here they would take 14 min 38 s, longer than the plan.

What I take from it

- With 32,000-token prompts on the default kernel this server read at most 2,759 prompt tokens a second.
- No request was refused and none failed. None ended either: the plan was too short for its own load.
- A plan is only as good as the rate it is sized on. The cache-growth plan, with shorter prompts and growth during the answer, was written because of this one.

What it does not show

- Anything about a full cache.
- How long one 32,000-token prompt takes alone. Here up to 40 shared the engine; the context ladder of this run sends one at a time.
- The fast attention kernel. This plan was not played on it.

● 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

Prefill
the server reading the prompt. Every prompt token passes through the model once before the first answer token.

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`.

Every number in this document was read from these files at build time (sha256, first 12 characters):
`FACTS.json` 160f5ca0d495 · `GLOSSARY.md` 98323be796af · `plans/kv-pressure.json` 14f77e84b345 · `runs/20261125T201346Z-series/plan-kv-pressure/requests.jsonl` 4af190f61697 · `runs/20261125T201346Z-series/plan-kv-pressure/timeline.jsonl` fb78341f868e · `runs/20261125T201346Z-series/summary.json` aab3940eb12e
47 values bound by path, 3 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