A tenth of the cache kept free: 21 streams thrown out instead of 47

The same 400 requests, one setting apart. The server lets in 235 streams instead of 261, their answers grow into the room that was kept, and the first wave of throwing out does not happen. The second does. The price is 26 fewer streams writing at first; all 400 answers arrive, 26 seconds sooner.

Qwen3.8-27B · one AMD Instinct MI300X · vLLM 0.27.1 **Run** 20261125T230017Z-series-watermark

Changed --watermark 0.1 (a tenth of the cache kept free when letting requests in) **Load** 400 requests at once: 4,000 tokens in, 2,000 out

I played the plan that had overfilled the cache once more, with one setting changed: keep a tenth of the cache free when letting requests in. The server then let in 235 streams at first instead of 261. Their answers grew into the room that was kept, and the first wave of throwing out did not happen. The second did: the cache filled all the same. 21 streams were thrown out instead of 47, and all 400 answers had arrived 26 s sooner. One run each.

21
streams thrown out of the cache; 47 with nothing kept free

72 s
longest silence inside one answer; 171 s with nothing kept free

235
streams let in at first; 261 with nothing kept free: the price

10 min 4 s
until all 400 had ended; 10 min 30 s with nothing kept free

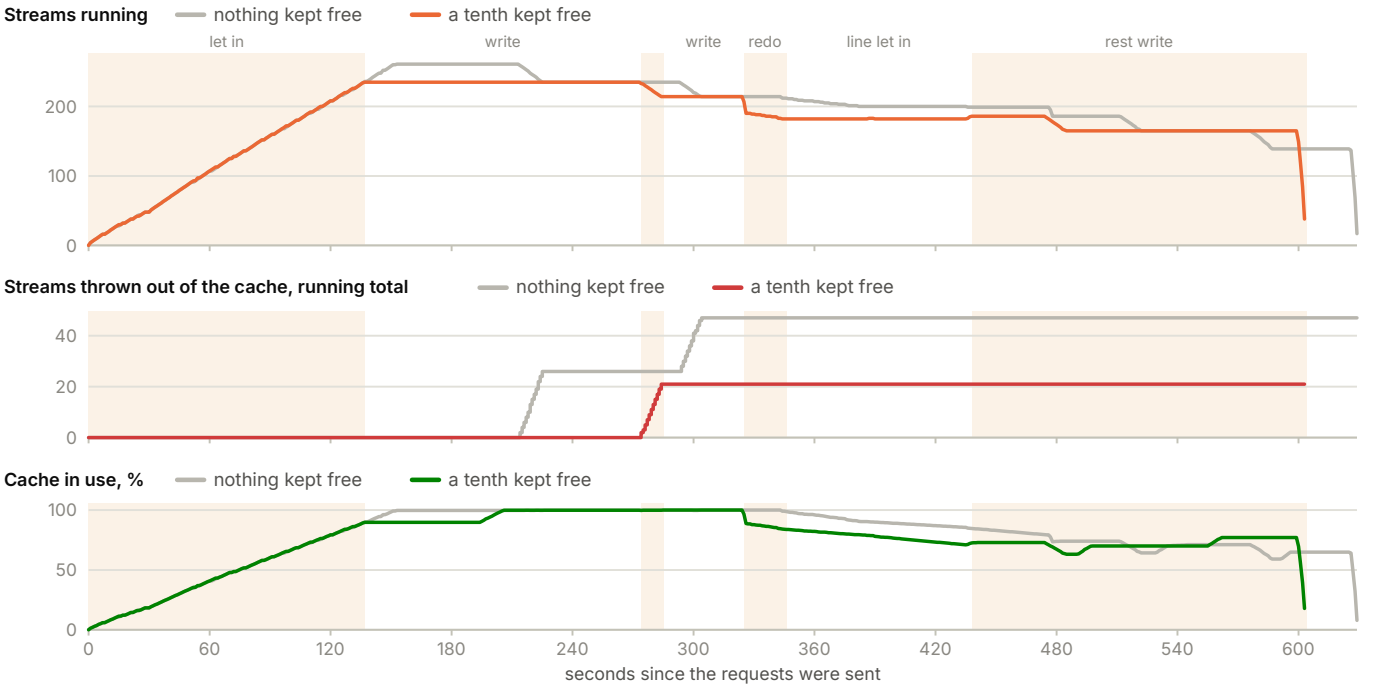

Figure 1 The two runs on one clock: this one in colour, the run with nothing kept free in grey. Top: streams running. Middle: streams thrown out of the cache, counted up. Bottom: the cache. With the room kept, the cache stops at 89.85% when requests are let in and fills later, as the answers grow. Shaded bands are the phases of this run.

Source: plan-kv-growth/timeline.json of this run and of run 20261125T201346Z-series (vLLM's gauges and preemption counter)

MEASURED

Both servers are left with the same 214 streams running once the throwing out is over, and no request had ended by then. The cache gauge reads 100% in both from there on: the room did not keep the cache from filling.

MEASURED

In this plan nothing was refused and nothing failed, in either run: all 400 requests got HTTP 200 and their full 2,000 tokens. Whoever was thrown out was let back in and finished.

The machine, and the one setting

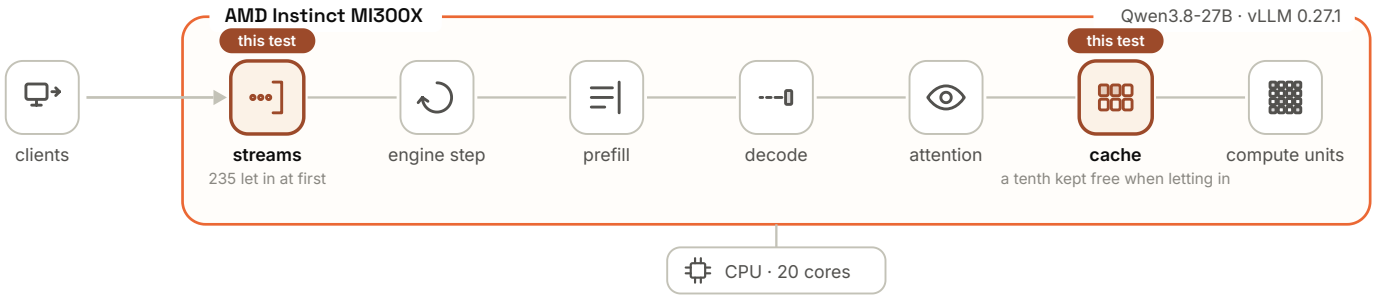

Figure 2 The card as every document in this series draws it: seven parts wired left to right, the clients outside, the host CPU below. This test changes how the cache is given out: a tenth of it is kept free when requests are let in.
Source: FACTS.json; summary.json plans.plan-kv-growth.stages[stage=grow-400].running_max, server.non_default_args

The same burst, with room kept at the door

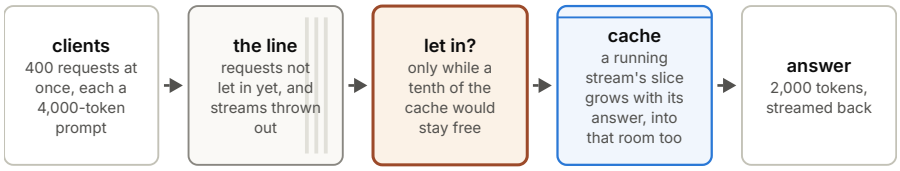

Figure 3 The rule that was added sits at the door, not inside: a waiting request is let in only while a tenth of the cache would stay free. A stream that is already running may grow into that room.
Source: config.conf (--watermark 0.1); vLLM's help text for the setting; plans/kv-growth.json

1. The same plan and the same seed as before: 400 requests in one instant, each with 4,000 tokens in and 2,000 out, the same prompts in the same order.
2. One setting differs: `--watermark 0.1`. Nothing else in the command changed.
3. The rest is the same in both runs: the default attention kernel, a cache of 1,826,476 token slots (1,826,476 in the other run), 1,024 stream slots.
4. Beside each run a sampler read the server's counters, the card and the CPU once a second.

KEEP IN MIND
One run each. The two servers were started separately, almost three hours apart, on the same machine; this one was ready after 100 s, the other after 94 s.

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	20261125T230017Z-series-watermark, started 2026-11-25T23:00:31Z, server ready after 100 s

KEEP IN MIND
The load is made on the same machine as the server and shares its CPU cores. No request leaves the machine.

Words used

Watermark
a share of the cache kept free when admitting requests (`--watermark`), to avoid preemption.

Preempted
a running stream thrown out of a full cache and put back in line; its work is redone later.

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`.

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

Decode
the server writing the answer, one token per stream per engine step.

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

The same plan, moment by moment

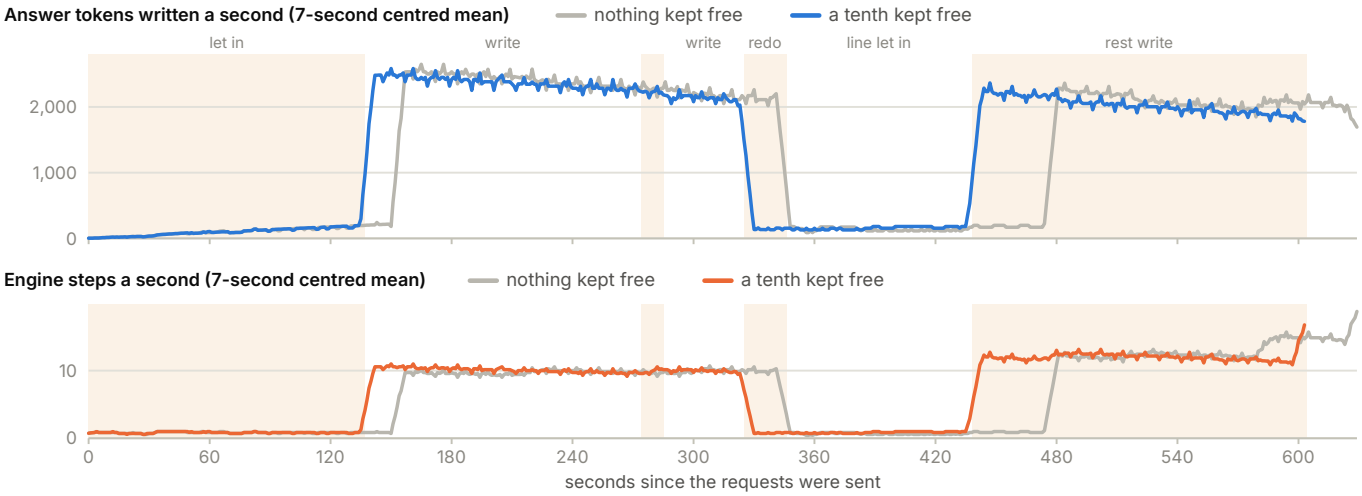

Figure 4 What the two engines did (7-second centred means): this run in colour, the other in grey. With the room kept, writing starts 16 s sooner, because fewer prompts are read first, and every later phase starts sooner too.
Source: plan-kv-growth/timeline.json of both runs (vLLM's token and step counters)

The exact record

THE SAME MOMENT IN BOTH RUNS	NOTHING KEPT FREE	A TENTH KEPT FREE
All that fit are let in, after s	153	137
Streams let in at first	261	235
Cache gauge then, %	99.79	89.85
First stream thrown out, after s	214	274
Streams thrown out, wave by wave	26 + 21	21
Streams left running after that	214	214
First answer ended, after s	345	325
No new prompt read for, s	42	21
The line empty, after s	476	438
All 400 ended, after s	629.7	603.6
Answer tokens a second over the plan	1,271.8	1,325.8
First token, median s	118.8	118.7
First token, 99th percentile s	476.0	437.2
Longest silence inside one answer, s	171.0	72.2

Moments are found in each run's per-second record: the first second with the most streams running, the first preemption, the first answer ended, the first prompt tokens read after that, the line empty.

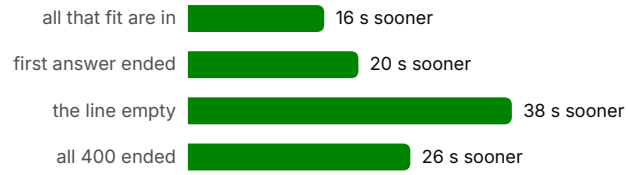

Figure 5 The gain in time: how much sooner four moments of the plan come with the room kept.
Source: timeline.json and summary.json of both runs; differences computed

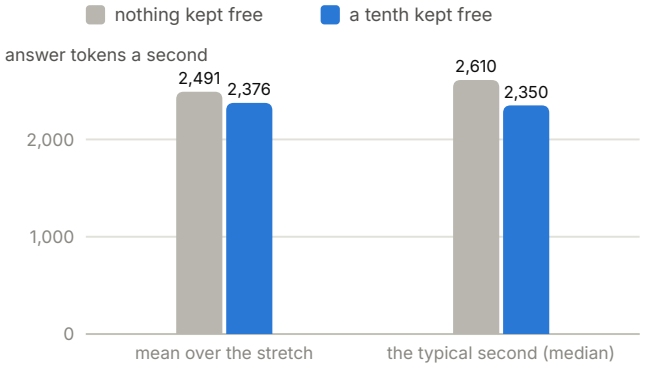

Figure 6 The price: answer tokens a second in the first stretch of writing, the seconds with the most streams running and no prompt being read (59 s with nothing kept free, 135 s with a tenth). Drawn twice: the mean over the stretch, and the typical second.
Source: plan-kv-growth/timeline.json of both runs (output_tok_s, running, prompt_tok_s)

COMPUTED

With 26 fewer streams writing, the first stretch gave 4.6% fewer answer tokens a second as a mean, and 9.9% fewer in the typical second. Over the whole plan this run wrote more: 1,326 against 1,272.

The room that was kept is what the growing answers need next

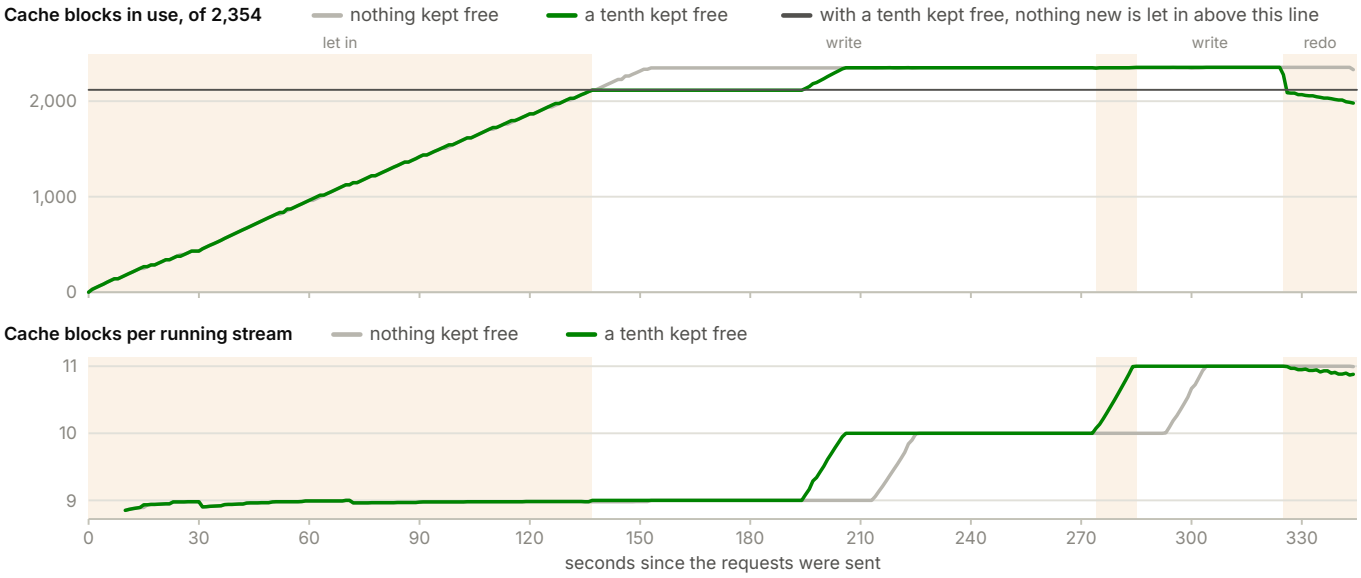

Figure 7 Until the first answers end. Top: the cache in blocks, both runs, and the level above which this run lets nothing new in. Bottom: blocks per running stream: 9, then 10, then 11 in both. With the room kept, the step from 9 to 10 fits and nobody is thrown out; the step to 11 does not fit in either run.

Source: plan-kv-growth/samples.jsonl of both runs (vllm:kv_cache_usage_perc, vllm:num_requests_running); blocks computed, see the text

What the block count asks for, and what each server did

RUN	BLOCKS	COMPUTED	MEASURED	STREAMS RUNNING AFTER	BLOCKS IN USE AFTER
LETTING IN, 9 BLOCKS A STREAM					
nothing kept free	2,354 may be used	261 fit	261 let in	261	2,349
a tenth kept free	2,118 may be used	235 fit	235 let in	235	2,115
EACH STREAM TAKES A 10TH BLOCK					
nothing kept free	5 free	26 must go	26 thrown out	235	2,350
a tenth kept free	239 free	none must go	none thrown out	235	2,350
EACH STREAM TAKES AN 11TH BLOCK					
nothing kept free	4 free	21 must go	21 thrown out	214	2,354
a tenth kept free	4 free	21 must go	21 thrown out	214	2,354

COMPUTED

Cut the cache into **2,354 equal blocks**, and every value its gauge took in these two plans is a whole number of them. A stream holds 9 blocks at first, then 10, then 11 as its answer grows. From that count alone come the streams let in and the streams thrown out, in both runs, to the stream.

MEASURED

The gauge rose from 89.85% to 99.83% between seconds 194 and 206 with nobody thrown out. The room is kept only at the door, as vLLM's help text says of the setting: blocks are kept free "when admitting waiting or preempted requests". A running stream may grow into it.

READING, NOT MEASUREMENT

My reading: the room does not make the cache larger. It keeps out the streams that would otherwise be let in and then thrown out. A tenth was what one step of growth needed in this plan; the next step still overflowed.

READING, NOT MEASUREMENT

vLLM's help text speaks of "KV cache blocks"; how many there are and what a stream takes is my arithmetic on the gauge, not read in its source. By the same count a setting of 0.2 would let in 209 streams, fewer than the 214 that fit at full length, so none would be thrown out. That is a prediction. I did not run it.

The same 400 requests, one by one

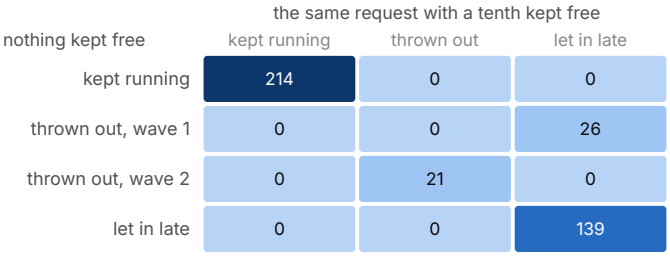

Figure 8 Every request of the burst, by what happened to it in the two runs. The requests are the same in both, in the same order.
Source: plan-kv-growth/requests.jsonl of both runs, matched by request id

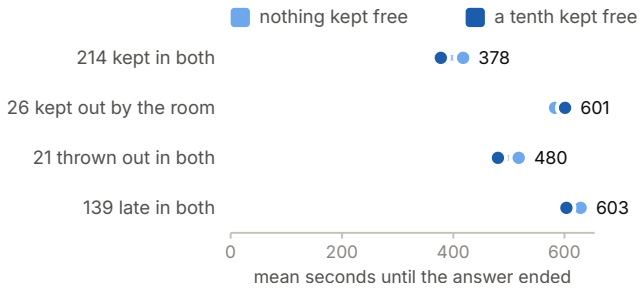

Figure 9 Mean seconds until the answer ended, for the four kinds of request. Three kinds gain; the 26 that the room kept out end later.
Source: plan-kv-growth/requests.jsonl of both runs

REQUESTS	NOTHING KEPT FREE	A TENTH KEPT FREE	FIRST TOKEN, MEAN S: WITHOUT	WITH	LONGEST SILENCE, S: WITHOUT	WITH	ANSWER ENDED, MEAN S: WITHOUT	WITH	CHANGE, S
214	Kept running	Kept running	64	64	12.9	8.2	418	378	-39.9
26	Thrown out, first wave	Let in late	147	355	171.0	4.0	583	601	+17.9
21	Thrown out, second wave	Thrown out	134	133	72.5	72.2	518	480	-37.5
139	Let in late	Let in late	437	404	6.6	5.3	629	603	-25.6

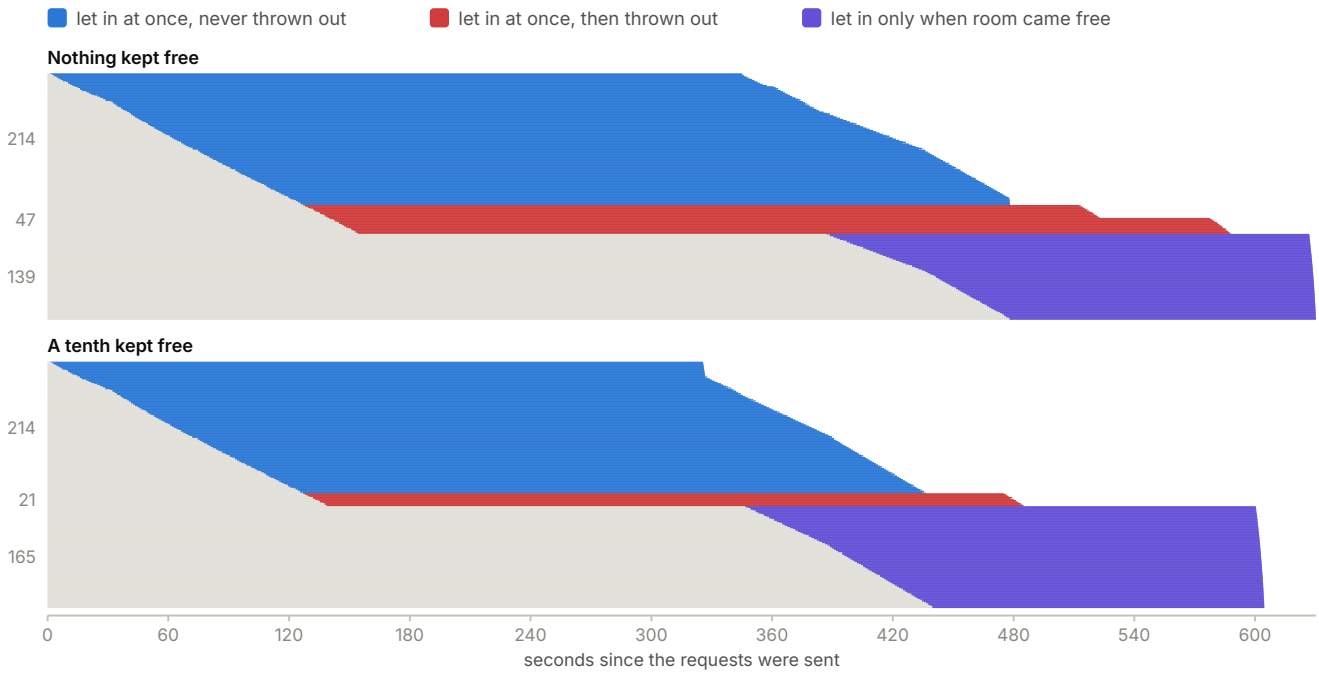

Figure 10 Every request as one thin line from its first token to the end of its answer (grey before it: waiting for the first token), both runs on one clock. The red band, streams thrown out, is narrower with the room kept; the last band, requests let in late, is wider by the same number.
Source: plan-kv-growth/requests.jsonl of both runs (t_sent, t_first, t_end, gap_max)

COMPUTED

Matched by their place in the burst, 374 of the 400 requests ended sooner with a tenth kept free, by 19 to 47 s. The other 26 ended 13 to 23 s later: they are the streams that the room kept out at first.

What I take from it, and what it does not show

What I take from it

● Keeping a tenth of the cache free more than halved the streams thrown out, 21 against 47, and the longest silence inside an answer, 72.2 s against 171.0 s.

● It did not remove them. The cache still reached 100.0% as the answers grew.

● The price was paid at the door: 26 fewer streams let in at first. The whole plan still ended 26 s sooner.

○ Both runs settle at 214 streams: 261 less 47, and 235 less 21.

◌ Throwing a stream out wastes its place twice: its work is done again, and while it was in, it held room that the others needed.

What it does not show

- One run each. A second pair could differ by a stream or a second.
- One plan, in which every stream has the same length and so needs room at the same moment. With mixed lengths the same setting may buy more, or less.
- Whether a larger share kept free would remove the throwing out. The block count says so; no run does.
- Prompts are random tokens: nothing repeats, and answers are not readable text.

Underneath: the card did the same work

MEANS OVER THE WHOLE PLAN	NOTHING KEPT FREE	A TENTH KEPT FREE
Compute, TFLOPS (vLLM's estimate)	213.3	212.9
Memory read, TB a second (vLLM's estimate)	2.00	2.09
Engine steps a second	6.47	6.75
Card power, watts (AMD's exporter)	710	724
CPU busy, % of 20 cores	10.9	10.9

Stage rows of the two summaries: means over each run's whole plan. Compute and memory read are vLLM's estimates from the model's shape, not hardware counters.

● 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 · `plans/kv-growth.json` 89c49daf377d · `runs/20261125T201346Z-series/plan-kv-growth/client.jsonl` a3a0ed7fb3ae · `runs/20261125T201346Z-series/plan-kv-growth/requests.jsonl` 38cbadbcd5d8 · `runs/20261125T201346Z-series/plan-kv-growth/samples.jsonl` 015baec3f8a7 · `runs/20261125T201346Z-series/plan-kv-growth/timeline.json` 630fd28b7c90 · `runs/20261125T201346Z-series/summary.json` aab3940eb12e · `runs/20261125T230017Z-series-watermark/plan-kv-growth/client.jsonl` 2eb5527fd787 · `runs/20261125T230017Z-series-watermark/plan-kv-growth/requests.jsonl` c6403acc2b19 · `runs/20261125T230017Z-series-watermark/plan-kv-growth/samples.jsonl` 941e5a17bd97 · `runs/20261125T230017Z-series-watermark/plan-kv-growth/timeline.json` 2144e8aaaaf2 · `runs/20261125T230017Z-series-watermark/summary.json` 55f891019fda

67 values bound by path, 10 figures drawn from data. built 2026-11-26 02:38Z, 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