The long-prompt cap lets more requests start, and almost none finish

The mixed load on the default kernel with one setting added: a prompt may take at most 2,048 tokens of an engine step. More requests are let in and get a first token. The engine still makes under one step a second, and of the requests sent after the first minute one ends inside the plan.

Qwen3.8-27B · one AMD Instinct MI300X · vLLM 0.27.1 **Run** 20261125T213149Z-series-longprefill2k

Changed a long-prompt cap of 2,048 tokens per step, against the series run

Load 70% short, 25% medium, 5% long prompts; 8, 32, 128, 256 streams

vLLM has a control made for this load: a prompt longer than 2,048 tokens may take at most 2,048 tokens of an engine step, and the rest of the step is left to others. I switched it on and played the same mixed plan. More requests were let in: up to 52 and 125 ran at 128 and 256 streams, where the server without the cap stayed at 32. But from 32 streams up the engine still made 0.22 to 0.57 steps a second, and every stream gets one token per step.

125
requests running at most at 256 streams; 32 without the cap

0.33
engine steps a second at 32 streams, 0.22 at 128, 0.57 at 256

1 of 248
requests sent after the first minute ended inside the plan

119.0 s
until the first long prompt's first token; the same prompt took 83.9 s without the cap

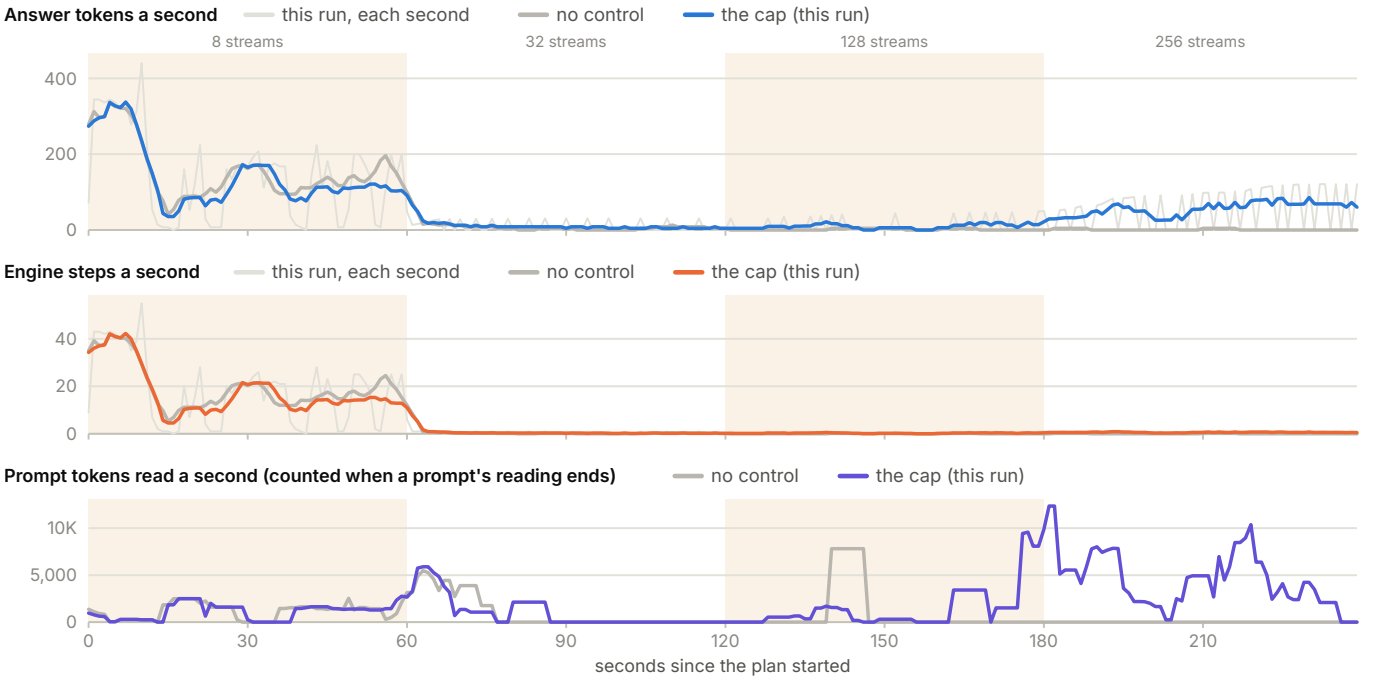

Figure 1 The four loaded minutes. Colour: this run, with its single seconds thin and grey behind the 7-second centred mean. Grey line: the same plan on the same kernel without the control. With the cap more prompts are read from the third minute on, and answers rise a little in the fourth.
Source: plan-mixed-overlap/timeline.json of this run and of 20261125T201346Z-series

MEASURED

At 128 and 256 streams the server wrote 9.0 and 54.4 answer tokens a second and left 2 and 121 of the requests sent in those minutes without a first token. Without the cap it wrote 1.6 and 1.1 and left 33 and 128 without one.

KEEP IN MIND

One setting differs from the grey run. At 8 streams both were sent the same 21 requests. After that a closed loop sends each server its own: at 32 streams this run was sent one long prompt and the grey run two, so I write no ratio there. At that stair this run pairs with the other control, which was also sent one.

The machine, and the part that was changed

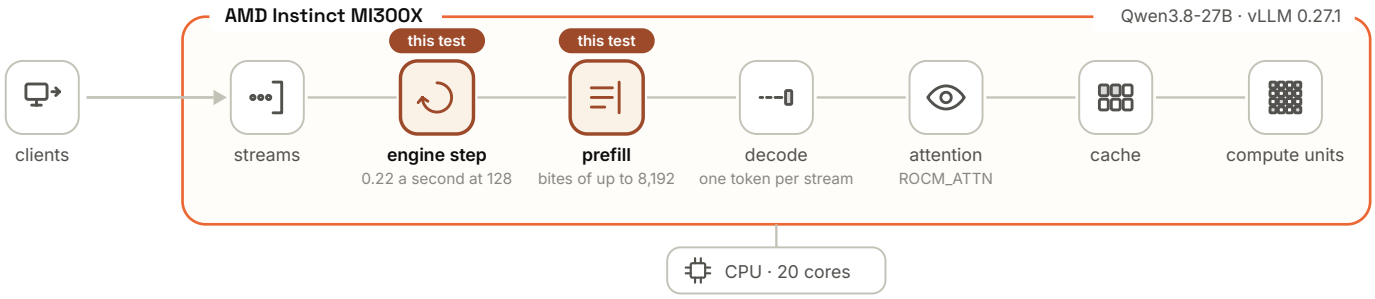

Figure 2 The card as every document in this series draws it. An engine step carries one new token for every stream that is answering and a bite of a prompt that is being read. This run caps how much of one long prompt a step may carry.
Source: FACTS.json; summary.json config and plans.plan-mixed-overlap.stages[stage=mix-128]

What differs between the two runs

Long-prompt cap	at most 2,048 tokens of one prompt in a step here; none in the run before
Room in one engine step	8,192 tokens in both
Attention kernel	ROCM_ATTN in both, by vLLM's own preference order
Cache	1,826,476 token slots here; 1,826,476 before
The run before	20261125T201346Z-series

The plan

1. Four stages of 60 seconds: 8, 32, 128, then 256 requests kept open. A new one starts only when one ends.
2. Each request draws its prompt from a mix: 70% short (256 to 2,000 tokens), 25% medium (4,000 to 16,000), 5% long (32,000 to 64,000); answers of 64 to 1,024 tokens.
3. Then 120 seconds with no new requests; what is still open after that is closed by the load generator.

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	20261125T213149Z-series-longprefill2k, started 2026-11-25T21:32:02Z, server ready after 94 s

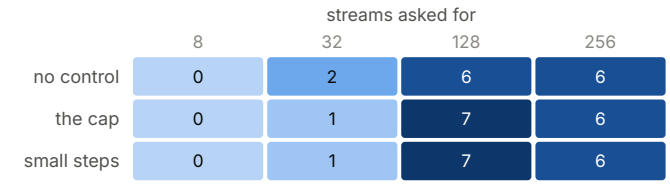

Figure 3 Long prompts sent in each minute, in the three runs on the default kernel. Rows that agree carried a like load in that minute. At 32 streams the two controls agree with each other, not with the run without one.
Source: plan-mixed-overlap/requests.jsonl of the three runs (stage, asked_in)

KEEP IN MIND

The plan is a closed loop: a new request is sent when one ends, so the moment each is sent depends on the server. Request number N has the same lengths in every run. "Ended inside the plan" is a fact about each run, not a ratio between runs.

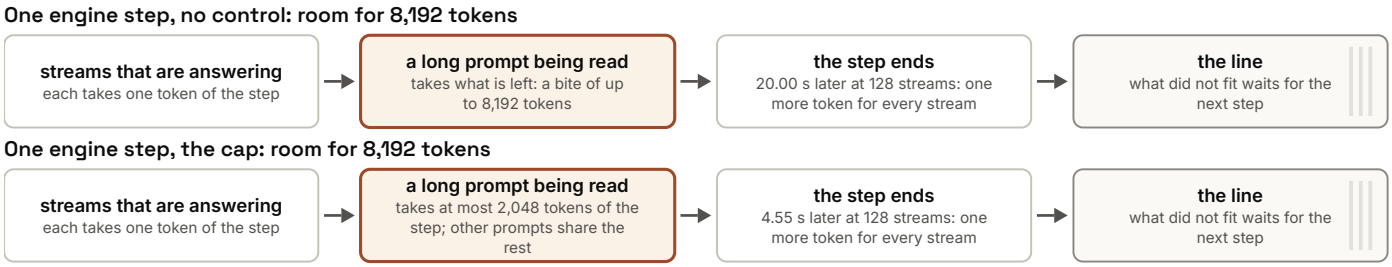

Figure 4 What one engine step holds in the two runs. The time is computed: one second divided by the engine steps a second at 128 streams.
Source: each run's summary.json (config, plans.plan-mixed-overlap.stages[stage=mix-128].engine_steps_per_s); FACTS.json

Minute by minute, with and without the control

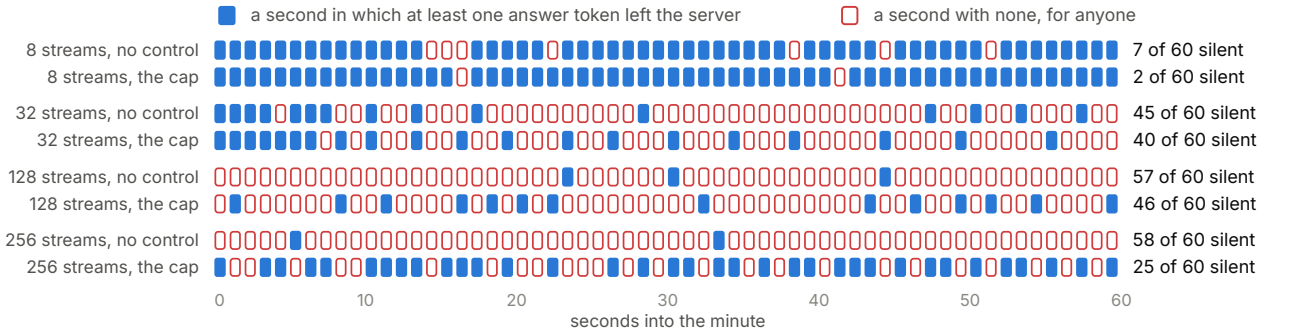

Figure 5 Every second of the four loaded minutes in both runs. The first minute is the same 21 requests in both. The rows at 32 streams are not like for like: one long prompt here, two in the other run. With the cap fewer seconds are silent at 128 and 256 streams, and most still are.
Source: plan-mixed-overlap/timeline.json of both runs (output_tok_s per one-second row)

Answer tokens a second, all streams together

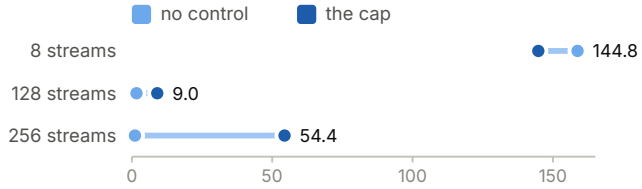

Figure 6 Answer tokens a second in the three minutes in which both runs were sent about as many long prompts.
Source: both runs' summary.json stages[].server_output_tok_s

Engine steps a second (log scale)

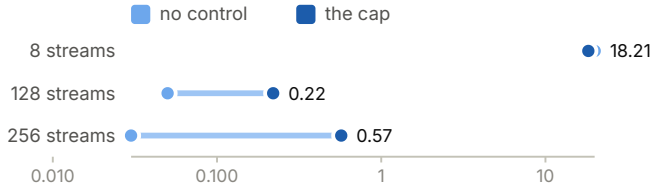

Figure 7 Engine steps a second in the same minutes. Every running stream gets one token per step.
Source: both runs' summary.json stages[].engine_steps_per_s

The exact record of both runs

	ANSWER TOKENS/S	PROMPT TOKENS/S	ENGINE STEPS/S	SECONDS WITH NO ANSWER TOKEN	RUNNING AT MOST	WAITING AT MOST	CACHE AT MOST, %	FIRST TOKEN, MEDIAN S	REQUESTS SENT	CARD, W
8 STREAMS										
no control	158.8	1,066	19.91	7 of 60	8	0	3.7	0.69	21	413
the cap	144.8	1,033	18.21	2 of 60	8	0	3.7	0.73	21	394
32 STREAMS										
no control	6.3	1,104	0.24	45 of 60	32	18	13.5	5.57	25	505
the cap	8.9	1,130	0.33	40 of 60	32	18	12.0	4.31	24	447
128 STREAMS										
no control	1.6	943	0.05	57 of 60	32	96	14.9	≥ 211.93	96	366
the cap	9.0	898	0.22	46 of 60	52	90	20.4	≥ 73.73	96	342
256 STREAMS										
no control	1.1	0	0.03	58 of 60	32	224	15.8	≥ 179.91	128	303
the cap	54.4	4,280	0.57	25 of 60	125	199	46.9	≥ 179.91	128	620
DRAIN: NO NEW REQUESTS										
no control	14.2	2,204	0.24	93 of 121	96	224	37.8	-	0	359
the cap	37.5	2,299	0.30	85 of 121	137	131	65.0	-	0	691

From each run's `summary.json`; seconds with no answer token are counted from its `timeline.json`. Tinted rows are this run. A stage row spans 59 s of its 60 s. The server counts a prompt's tokens when its reading ends. Running and waiting are the server's gauges, which move only when a step ends: with steps this long they are often old.

Like for like at 32 streams: the two controls, one long prompt each

AT 32 STREAMS	ANSWER TOKENS/S	ENGINE STEPS/S	SECONDS WITH NO ANSWER TOKEN	FIRST TOKEN, MEDIAN S	RUNNING AT MOST
the cap	8.9	0.33	40 of 60	4.31	32
small steps	18.8	0.76	34 of 60	5.64	32

A smaller bite of the long prompt, and the step is still slow

	ROOM IN A STEP, TOKENS	CAP ON ONE PROMPT PER STEP	THE PROMPT, TOKENS	SENT AT SECOND	ITS FIRST ANSWER TOKEN AFTER, S	ENGINE STEPS IN BETWEEN	ANSWER TOKENS WRITTEN MEANWHILE, FOR EVERYONE	THAT IS, A SECOND
no control	8,192	none	54,698	60	83.9	15	395	4.7
the cap	8,192	2,048	54,698	60	119.0	33	1,091	9.2
small steps	2,048	none	54,698	60	181.3	60	1,591	8.8

The first long prompt of the plan is the same request in every run and was sent in the same second. Steps and answer tokens are counted from each run's `timeline.json` between its sending and its first token (computed). The tinted row is this run.

Seconds per engine step: no control

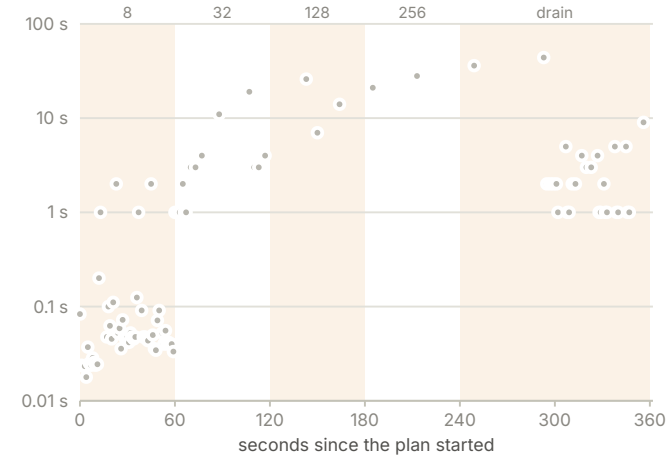

Figure 8 Without the control: from the moment a long prompt is read, a step takes seconds, then tens of seconds.
Source: timeline.json of 20261125T201346Z-series (steps_s)

Seconds per engine step: the cap

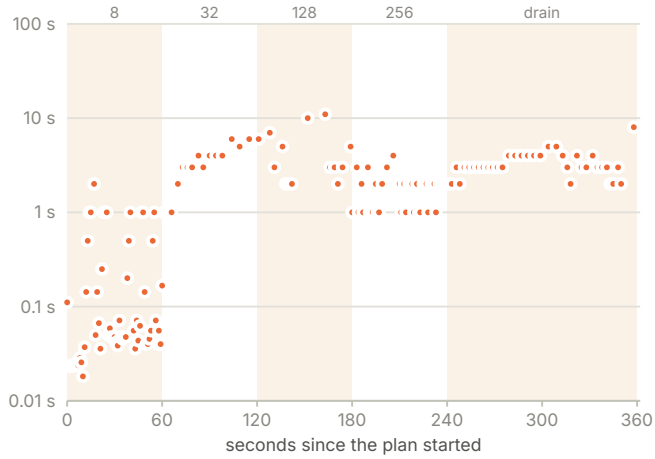

Figure 9 With the cap: once long prompts are in the mix a step takes 1 to 11 seconds, 3 at the median. Computed from the step counter: one second divided by the steps in it, or the seconds since the last step.
Source: plan-mixed-overlap/timeline.json (steps_s)

MEASURED

The same prompt of 54,698 tokens, sent in the same second: with the cap its first token came after 119.0 s, without it after 83.9 s. The reading needed about twice the engine steps, and the server wrote about twice the answer tokens a second for everyone meanwhile: still under ten.

MEASURED

What the cap changes is who gets in. The rest of each step goes to other prompts, so 52 and 125 requests were running at most at 128 and 256 streams instead of 32.

READING, NOT MEASUREMENT

My reading: capping the bite does not make the step short, because the slow part is attention over the long context with the default kernel. Every stream that was let in then gets a token only when a step ends, and a step still takes seconds.

KEEP IN MIND

The size of a bite is the server's setting. Its "tokens per step" counter is not the bite: it counts a whole prompt in the step that finishes reading it.

What the clients saw

Every request: no control

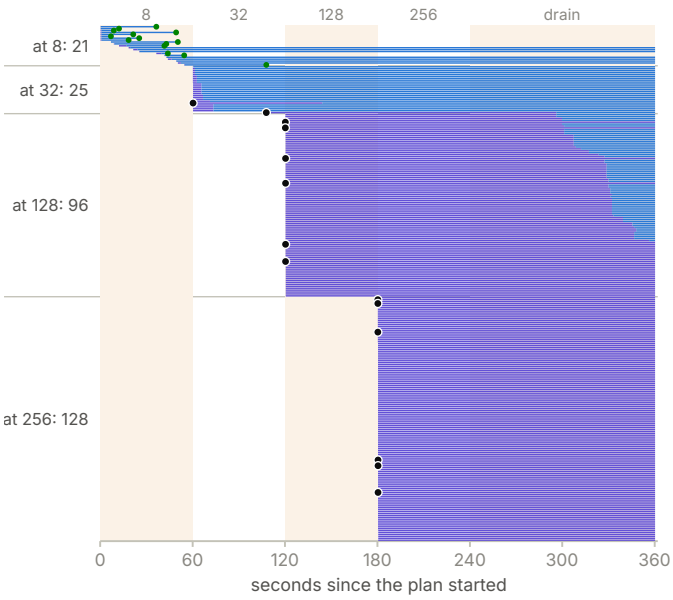

Figure 10 Every request of the run without the control as one line, in the order sent. Violet: no answer token yet. Blue: answering. Green dot: ended well. Black dot: a long prompt.
Source: requests.jsonl of 20261125T201346Z-series

Every request: the cap

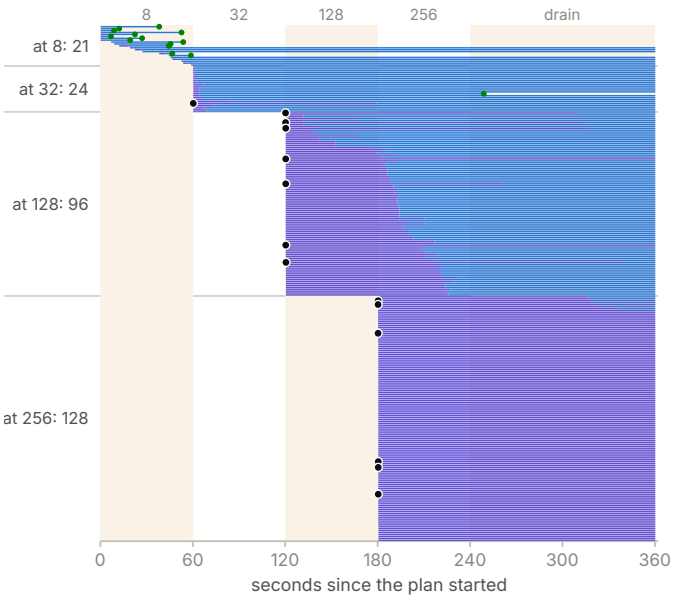

Figure 11 The same for this run. More lines turn blue, and sooner. Hardly one of them ends in a green dot after the first minute.
Source: plan-mixed-overlap/requests.jsonl (t_sent, t_first, t_end, end)

	REQUESTS SENT	LONG PROMPTS AMONG THEM	FIRST TOKEN, MEDIAN S	LONGEST, S	ENDED INSIDE THE PLAN	CUT WHILE ANSWERING	CUT BEFORE A FIRST TOKEN
SENT AT 8 STREAMS							
no control	21	0	0.7	5.0	14	7	0
the cap	21	0	0.7	6.1	13	8	0
SENT AT 32 STREAMS							
no control	25	2	5.6	188.3	0	25	0
the cap	24	1	4.3	119.0	1	23	0
SENT AT 128 STREAMS							
no control	96	6	209.8	236.0	0	63	33
the cap	96	7	73.7	221.5	0	94	2
SENT AT 256 STREAMS							
no control	128	6	-	-	0	0	128
the cap	128	6	142.0	178.4	0	7	121
ALL REQUESTS OF THE PLAN							
no control	270	14	187.2	236.0	14	95	161
the cap	269	14	67.6	221.5	14	132	123

From each run's `requests.jsonl`. Tinted rows are this run. The median and the longest first-token time are over the requests that got a first token. The client takes the first piece of visible text as the first token, so with steps seconds apart a first-token time can be a step or two late.

MEASURED

With the cap 1 of the 248 requests sent after the first minute ended inside the plan. Of the 96 sent at 128 streams, all but 2 got a first token, after 73.7 s at the median.

NOT A RESULT

Requests "cut at the end of the plan" (255 here) are not refusals and not failures: the load generator closed them when the plan's time was over. All 269 requests got HTTP 200.

More of the card at work at 256 streams, to little effect

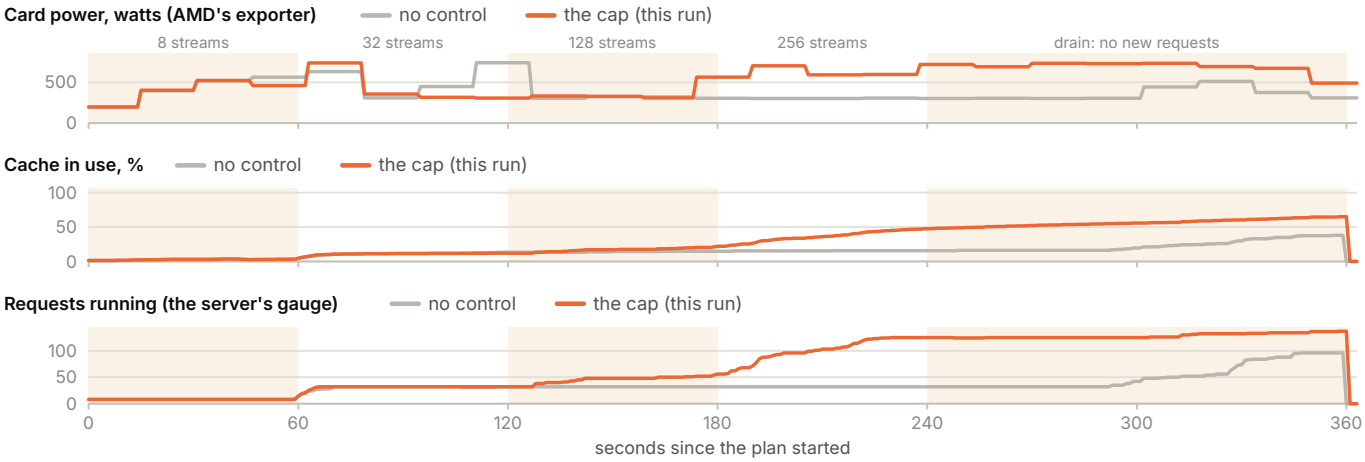

Figure 12 Both runs over the whole plan. Grey: without the control. Colour: this run. Power is AMD's device exporter (a new reading every 16 s); cache and running are the server's gauges.
Source: plan-mixed-overlap/timeline.json of both runs (watts, cache_percent, running)

MEASURED

At 256 streams the card drew 620 W with the cap and 303 W without it. The cache was 46.9% full at most; without the cap 15.8%.

ESTIMATE

◑ vLLM's estimate of the compute at 256 streams: 246 TFLOPS with the cap, 26.6 without. Both overcount long prompts: attention is counted in all 64 layers and the model has it in 16.

What I take from it

- ● The cap lets more requests in and gives more of them a first token.
- ● The engine stays under one step a second from 32 streams up, and 1 request sent after the first minute ends inside the plan.
- ● The long prompt itself is read more slowly: 119.0 s instead of 83.9 for the same prompt.
- ◌ Capping the bite does not make the step short: the slow part is attention over the long context with the default kernel.

Not results, and limits

- At 32 streams this run and the run without the control were not sent the same long prompts; no ratio is written there.
- A closed loop: "ended inside the plan" is a fact about each run. The bench has a second plan with the same requests at the same moments for every server.
- Default attention kernel throughout. Prompts are random tokens; nothing here is about the quality of an answer.

● 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.

Chunked prefill
a long prompt is read in bites across several engine steps, at most `max_num_batched_tokens` (8,192 here by default) per step for all requests together.

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

`FACTS.json` 160f5ca0d495 · `GLOSSARY.md` 98323be796af · `plans/mixed-overlap.json` b86c1f4ef044 · `runs/20261125T201346Z-series/plan-mixed-overlap/requests.jsonl` ec7778099e60 · `runs/20261125T201346Z-series/plan-mixed-overlap/t0` f79227400789 · `runs/20261125T201346Z-series/plan-mixed-overlap/timeline.json` 328dd803edea · `runs/20261125T201346Z-series/summary.json` aab3940eb12e · `runs/20261125T213149Z-series-longprefill2k/plan-mixed-overlap/requests.jsonl` 774120396827 · `runs/20261125T213149Z-series-longprefill2k/plan-mixed-overlap/t0` c945be467c45 · `runs/20261125T213149Z-series-longprefill2k/plan-mixed-overlap/timeline.json` 37662cd0f155 · `runs/20261125T213149Z-series-longprefill2k/summary.json` 110822c67acb · `runs/20261125T214036Z-series-batch2k/plan-mixed-overlap/requests.jsonl` d80dc5271519 · `runs/20261125T214036Z-series-batch2k/plan-mixed-overlap/t0` fffb5e8db961 · `runs/20261125T214036Z-series-batch2k/plan-mixed-overlap/timeline.json` ced017b86f78 · `runs/20261125T214036Z-series-batch2k/summary.json` 3d4fa8438e60

80 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