9.2% more cache, 9.2% more streams

Letting vLLM plan with 97% of the card's memory instead of its default 92% gives a cache of 113.23 GiB against 103.65. The staircase then fills at 591 streams against 541: 50 more, the same 5.22 streams for each GiB of cache. Output at the top stays about the same.

Qwen3.8-27B · one AMD Instinct MI300X · vLLM 0.27.1 **Run** 20261125T225329Z-series-mem97
Changed --gpu-memory-utilization 0.97 (97% of the card's memory instead of 92%), on the fast attention kernel
Load 64 tokens in, 512 out, 256 to 2,048 streams

The staircase stops letting streams in when the cache is full. So I let vLLM plan with 97% of the card's memory instead of its default 92%, on the fast attention kernel, and played the top of the staircase again. The cache grew from 1,679,893 to 1,835,008 token slots, 9.2% more, and the most streams running at once went from 541 to 591: 50 more, which is 9.2%. Output at the top stayed about the same. One run each.

9.2%
more cache: 1,835,008 token slots against 1,679,893, 113.23 GiB against 103.65

9.2%
more streams running at once: 591 against 541

196 MiB
of cache for each running stream, in both servers: computed from each server's log

3,941
answer tokens a second with 2,048 streams asked for; 3,921 with 92% of the card

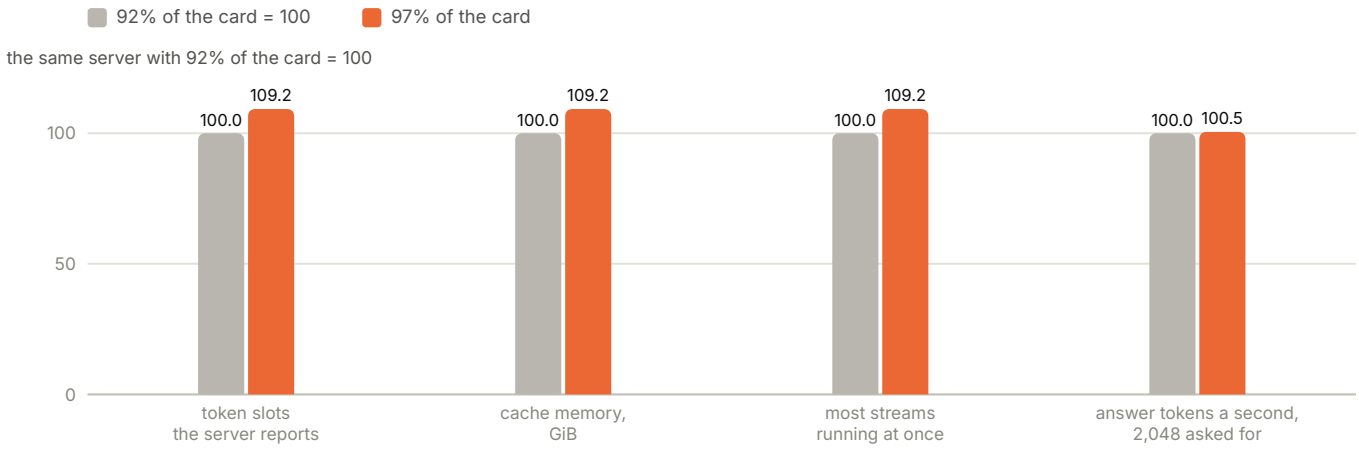


Figure 1 The server with 97% of the card against the same server with 92%, each measure with the 92% server set to 100. Token slots, cache memory and the streams that fit rise together. The output does not.
Source: summary.json of this run and of run 20261126T013018Z-series-fa-seqs2048 (server.kv_cache_tokens, stages[stage=up-2048].running_max, server_output_tok_s); cache memory from each server.log ("Available KV cache memory")

COMPUTED
For each GiB of cache both servers ran 5.22 streams. More memory bought streams in proportion: the cache is what limits how many short streams run at once, and nothing else got in the way first.

KEEP IN MIND
One thing differs between the two servers: both had 2,048 stream slots, so that the stream limit could not be the ceiling. The first version of this document compared with a server of 1,024 slots and showed 5.7% (591 against 559). That server sets less memory aside for its busiest step, 16.31 GiB against 19.64, so its cache was larger to begin with: 107.05 GiB.

The machine, and the one setting

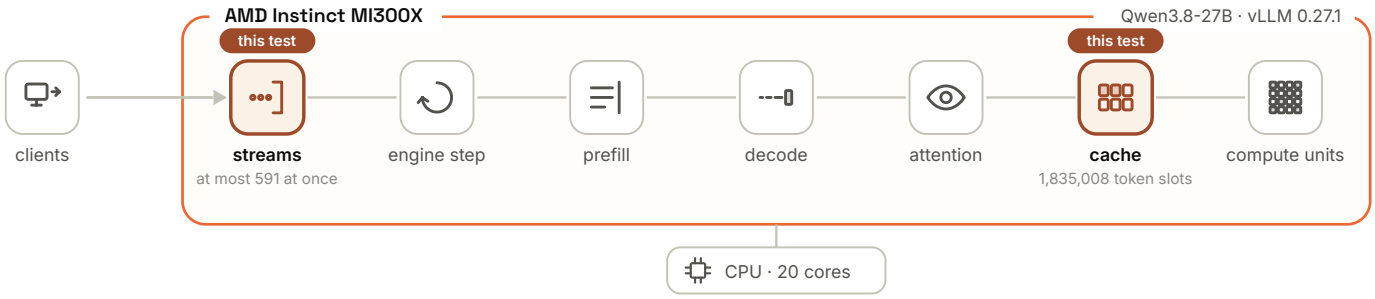


Figure 2 The card as every document in this series draws it: seven parts wired left to right, the clients outside, the host CPU below. This test gives the server a larger share of the card's memory, and asks how many more streams then fit in the cache.
Source: FACTS.json; summary.json server.kv_cache_tokens, plans.plan-staircase-top.stages[stage=up-2048].running_max

The top of the staircase, twice

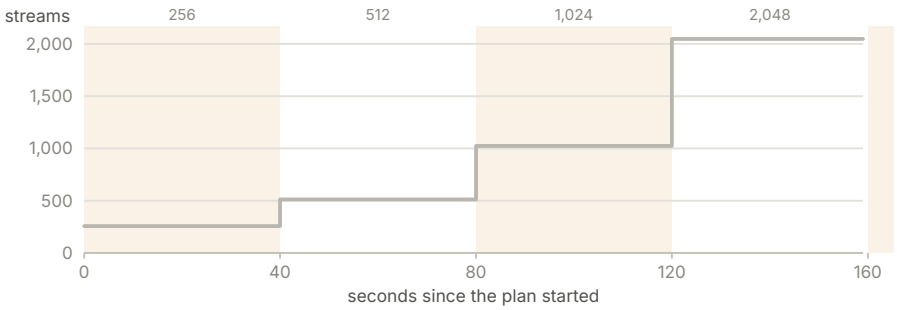


Figure 3 The plan: streams asked for over time, on an empty server. Each stair lasts 40 seconds; then the generator waits 60 seconds for the line to empty and stops.
Source: plan-staircase-top/timeline.json (target); plans/staircase-top.json

1. A closed loop: the load generator keeps 256, then 512, 1,024 and 2,048 requests open, 40 seconds each. A new request starts when one ends.
2. Every prompt is 64 random token ids, so no two share a prefix; every answer is forced to 512 tokens.
3. Two servers played it, both on the fast attention kernel: one with vLLM's default share of the card (run `20261126T013018Z-series-fa-seqs2048`), this one with `--gpu-memory-utilization 0.97`.
4. Both had 2,048 stream slots. Nothing else differs between them.
5. Beside each run a sampler read the server's counters, the card and the CPU once a second.

KEEP IN MIND
The server did start with 97%, and played this plan without failing. That is all this test says about how far the share can be pushed.

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	20261125T225329Z-series-mem97, started 2026-11-25T22:53:42Z, server ready after 103 s

KEEP IN MIND
The load is made on the same machine as the server and shares its CPU cores. This server was ready after 103 s, the other after 100 s.

Words used

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.

"Cache for N tokens"
what the server reports at start: how many token slots fit in the cache (about 1.82 million with default settings here).

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.

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`.

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

The larger cache lets 50 more streams in, and no more

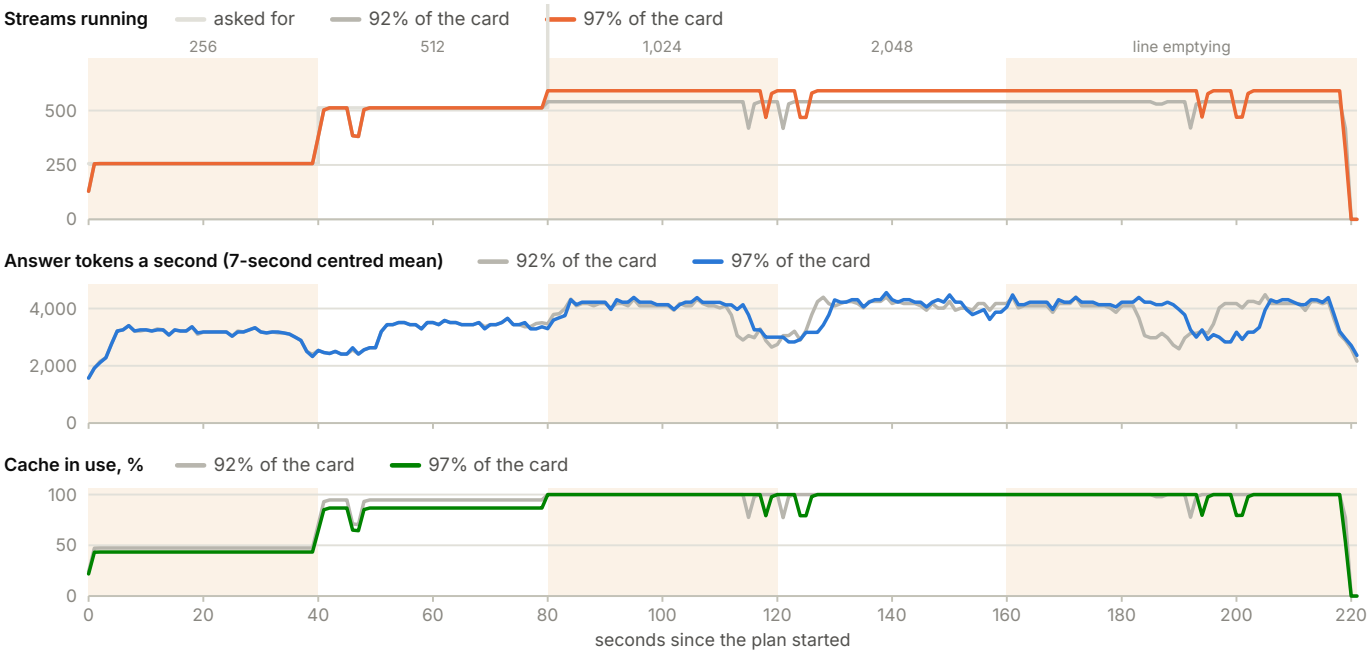


Figure 4 The two runs on one clock: this one in colour, the server with 92% of the card in grey. Streams running follow the request (thin steps) to 512; above that this server stops at 591, the other at 541. Output is a 7-second centred mean. Shaded bands are the stairs, labelled with the streams asked for; in the last band nothing new is sent.
Source: plan-staircase-top/timeline.json of both runs (vLLM's gauges and counters, sampled every second)

The exact record

SERVER	RUNNING AT MOST	WAITING AT MOST	CACHE AT MOST, %	ANSWER TOKENS/S	PROMPT TOKENS/S	ENGINE STEPS/S	TOKENS PER STEP	FIRST TOKEN, MEDIAN S	TFLOPS, EST.
256 STREAMS ASKED FOR									
92% of the card	256	127	47.3	3,115	429	12.18	291	2.13	164
97% of the card	256	127	43.3	3,116	430	12.18	291	2.13	164
512 STREAMS ASKED FOR									
92% of the card	512	100	94.6	3,189	862	6.26	647	2.06	185
97% of the card	512	132	86.6	3,189	862	6.26	647	2.07	185
1,024 STREAMS ASKED FOR									
92% of the card	541	483	100.0	3,923	431	7.26	599	42.71	202
97% of the card	591	433	100.0	4,102	133	6.95	610	40.49	198
2,048 STREAMS ASKED FOR									
92% of the card	541	1,507	100.0	3,921	480	7.26	606	≥ 99.90	203
97% of the card	591	1,457	100.0	3,941	564	6.68	674	≥ 95.18	208

One row per stair and server, from the two `summary.json` files. Running, waiting and cache are the most seen in the stair; output, prompts and steps are the server's counters over it; the first-token time is the median for requests sent in that stair. A stage row spans 39 s of a 40 s stair, first to last one-second sample. TFLOPS is vLLM's estimate.

MEASURED

At 512 streams the larger cache is 86.6% full where the smaller one is 94.6% full. With 1,024 and 2,048 asked for, 591 streams ran at most and 1,457 waited; on the other server 541 ran and 1,507 waited.

NOT A RESULT

1,536 of this run's 2,895 requests were "cut at the end of the plan": still open when the generator's 60 s ran out, and closed by it. They are not refusals. All 2,895 got HTTP 200.

The cache gets what is left of the share

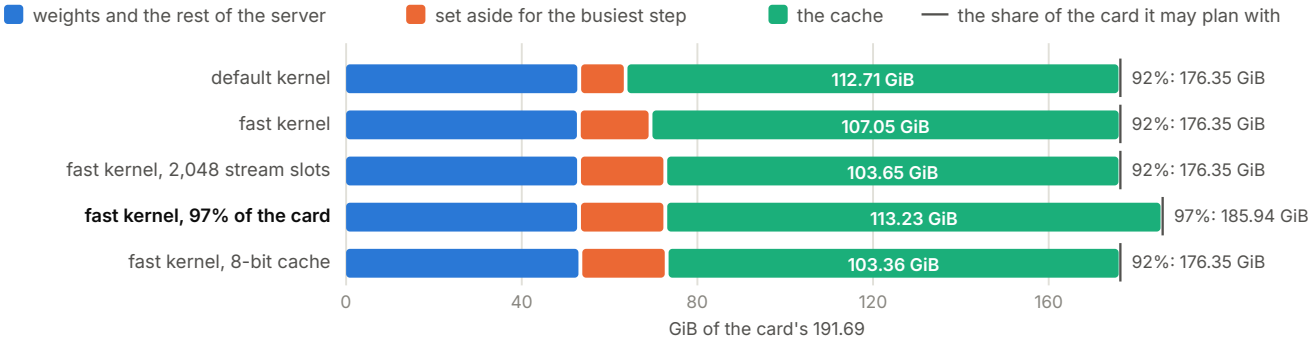


Figure 5 Where each of the five servers that played this plan put the card's memory, by its own log: what the server itself consumed, what it set aside for its busiest step, and the cache, which gets the rest of the share it may plan with. This run is in bold.
Source: server.log of each run ("Desired GPU memory utilization is ... Actual usage is ...", "Available KV cache memory")

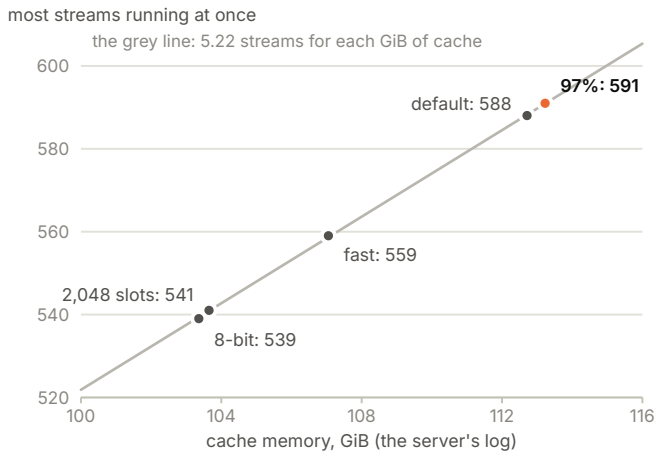


Figure 6 The most streams running at once against the cache's size in GiB, five servers: the default attention kernel, the fast kernel, the fast kernel with 2,048 stream slots, this one, and the 8-bit cache, which sits on top of its 16-bit twin. They lie on one line through zero: the streams that fit follow the bytes.
Source: summary.json of each run (running_max at 2,048 asked for); cache GiB from each server.log

GIB, BY EACH SERVER'S OWN LOG	92% OF THE CARD	97% OF THE CARD	DIFFERENCE
Share the server may plan with	92%: 176.35	97%: 185.94	+9.59
Consumed by the server itself (weights and the rest)	53.06	53.06	+0.00
Set aside for the busiest step	19.64	19.64	+0.00
The cache: what is left of the share	103.65	113.23	+9.58
Compiled graphs, taken after the cache was sized	12.90	12.90	+0.00
All four together, on a card of 191.69	189.25	198.83	+9.58

COMPUTED

On paper this server has promised more than the card holds. Its own log lists 198.83 GiB in all, once the 12.9 GiB of compiled graphs are counted that it takes after sizing the cache; the card has 191.69. What it sets aside for the busiest step is a reserve, not memory in use, and this plan never needed all of it.

COMPUTED

The share grew by 9.59 GiB and the cache by 9.58: all of it went to the cache. In all five servers a running stream of this plan costs about **196 MiB of cache**.

READING, NOT MEASUREMENT

My reading: with streams this short the cache is filled by a fixed amount of state for each stream, in bytes, so memory buys streams one for one. How much further the share can go before a heavy step no longer fits, this plan does not show: at most 591 streams ran, with 2,048 slots allowed.

The five servers, exactly

SERVER	SHARE OF THE CARD, %	CONSUMED, GIB	SET ASIDE, GIB	CACHE, GIB	TOKEN SLOTS	MOST STREAMS RUNNING	STREAMS PER GIB	MIB OF CACHE PER STREAM	SLOTS PER STREAM	ANSWER TOKENS/S: 1,024 ASKED	2,048 ASKED
default kernel	92	53.07	10.57	112.71	1,826,476	588	5.22	196.3	3,106	3,856	4,096
fast kernel	92	53.00	16.31	107.05	1,734,958	559	5.22	196.1	3,104	3,949	3,993
fast, 2,048 stream slots	92	53.06	19.64	103.65	1,679,893	541	5.22	196.2	3,105	3,923	3,921
fast, 97% of the card	97	53.06	19.64	113.23	1,835,008	591	5.22	196.2	3,105	4,102	3,941
fast, 8-bit cache	92	53.36	19.64	103.36	3,309,759	539	5.21	196.4	6,141	3,962	4,182

Memory is each server's own log; slots and streams are each run's `summary.json`; streams per GiB, MiB per stream and slots per stream are computed. The default-kernel server played the full staircase, the four others its top only.

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

What I take from it

● With 97% of the card the cache had 9.2% more token slots and the staircase filled at 591 streams instead of 541, 9.2% more.

○ Both servers ran 5.22 streams for each GiB of cache, about 196 MiB a stream.

● Output at the top stayed about the same: 4,102 and 3,941 answer tokens a second at the two top stairs, against 3,923 and 3,921.

◌ More memory lets more short streams run at once. It does not make the card write more: the extra streams share the same output, and each gets a little less.

What it does not show

- How far the share can be pushed. One value was tried, on one plan, and it held.
- More than one run of each server, or another plan: this is the top of one staircase.
- Anything about long streams, where the cache holds mostly tokens.
- One run each. A stair's output swings with the moment at which new prompts are read: at 1,024 asked for this server read 133 prompt tokens a second, the other 431.

Underneath: the card did the same work

SERVER	COMPUTE, TFLOPS (EST.)	MEMORY READ, TB/S (EST.)	MATRIX UNITS IN USE, %	WATTS	CPU BUSY, %	ENGINE CORES
1,024 STREAMS ASKED FOR						
92% of the card	202.2	0.80	19.0	750	11.6	2.00
97% of the card	197.6	0.79	27.7	749	11.9	2.00
2,048 STREAMS ASKED FOR						
92% of the card	203.0	0.61	36.0	749	11.7	1.99
97% of the card	207.7	0.60	40.3	749	11.8	1.98

Stage rows of the two summaries. Compute and memory read are vLLM's estimates from the model's shape, not hardware counters; matrix units and watts are readings of AMD's device exporter.

● 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/staircase-top.json` 7671a9348c18 · `runs/20261125T201346Z-series/server.log` c008d4941d1a · `runs/20261125T201346Z-series/summary.json` aab3940eb12e · `runs/20261125T215049Z-series-fa/server.log` a2aa2d010fcc · `runs/20261125T215049Z-series-fa/summary.json` ec0c01913c37 · `runs/20261125T224501Z-series-kvfp8/server.log` 12a55b33336a · `runs/20261125T224501Z-series-kvfp8/summary.json` 561b6d9e405a · `runs/20261125T225329Z-series-mem97/plan-staircase-top/timeline.json` 2bc691659666 · `runs/20261125T225329Z-series-mem97/server.log` 18247a4dd0d4 · `runs/20261125T225329Z-series-mem97/summary.json` ab4cc85f7ebd · `runs/20261126T013018Z-series-fa-seqs2048/plan-staircase-top/timeline.json` e5d5a112cf56 · `runs/20261126T013018Z-series-fa-seqs2048/server.log` 5a8a691ca70f · `runs/20261126T013018Z-series-fa-seqs2048/summary.json` 94ecccaf84b3

62 values bound by path, 6 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