The fast kernel has the same ceiling: about 4,000 answer tokens a second

With short prompts the attention kernel does not decide the top. The fast kernel's server stops at 559 streams where the default one stops at 588, because it starts with 5.0% less cache; for each GiB of cache both run 5.22 streams, and both write about 4,000 answer tokens a second.

Qwen3.8-27B · one AMD Instinct MI300X · vLLM 0.27.1 **Run** 20261125T215049Z-series-fa

Changed the decoder's attention on AMD's AITER flash-attention kernel (VLLM_ROCM_USE_AITER=1, --attention-backend ROCM_AITER_FA)

Load 64 tokens in, 512 out, 256 to 2,048 streams

For long prompts the fast attention kernel changes everything. For short ones I expected little, and played the top of the staircase on it to see: 64 tokens in, 512 out, 256 to 2,048 streams. With the server full it wrote 3,949 and 3,993 answer tokens a second, where the default kernel had written 3,856 and 4,096: about 4,000 either way. It stopped at 559 streams instead of 588, because its server starts with 5.0% less cache.

3,993
answer tokens a second with 2,048 streams asked for; 4,096 on the default kernel

559
streams running at once, at most; 588 on the default kernel

5.0%
less cache than the default kernel's server: 1,734,958 token slots against 1,826,476

5.22
streams for each GiB of cache, on both kernels: computed from each server's log

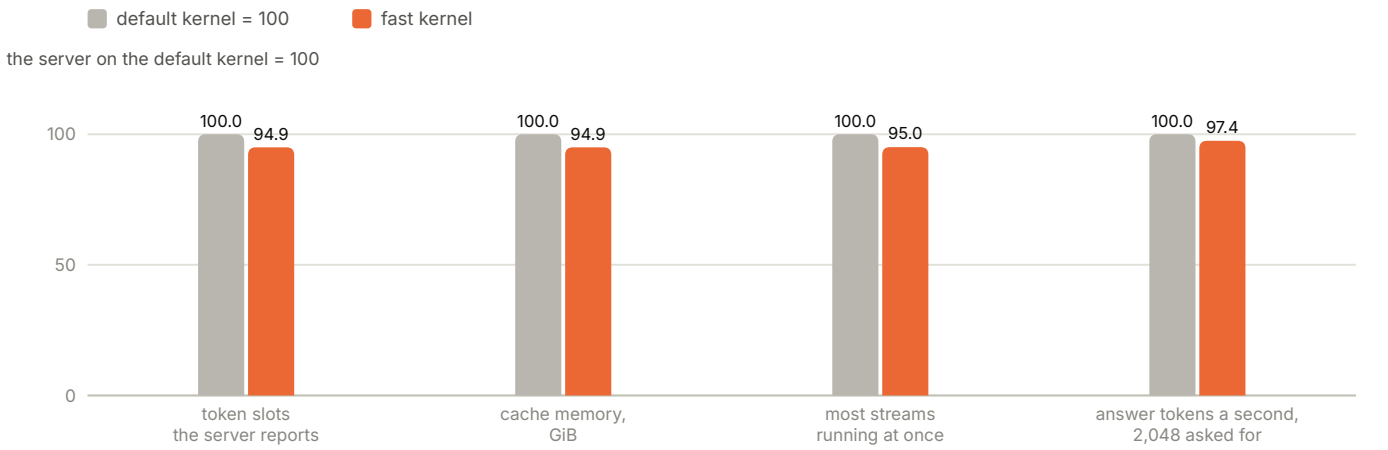

Figure 1 The fast kernel's server against the default kernel's, each measure with the default kernel set to 100. Cache and streams are a twentieth lower, together. The output with 2,048 streams asked for is within 3%.
Source: summary.json of this run and of run 20261125T201346Z-series (server.kv_cache_tokens, stages[stage=up-2048].running_max, server_output_tok_s); cache memory from each server.log ("Available KV cache memory")

MEASURED

With 1,024 and with 2,048 streams asked for both servers are full, and that is where they compare. This one wrote 3,949 and 3,993 answer tokens a second, the default kernel's 3,856 and 4,096. Neither is ahead at both stairs.

COMPUTED

The 29 streams between 559 and 588 are the cache: 107.05 GiB against 112.71. For each GiB both servers ran 5.22 streams.

The machine, and the one setting

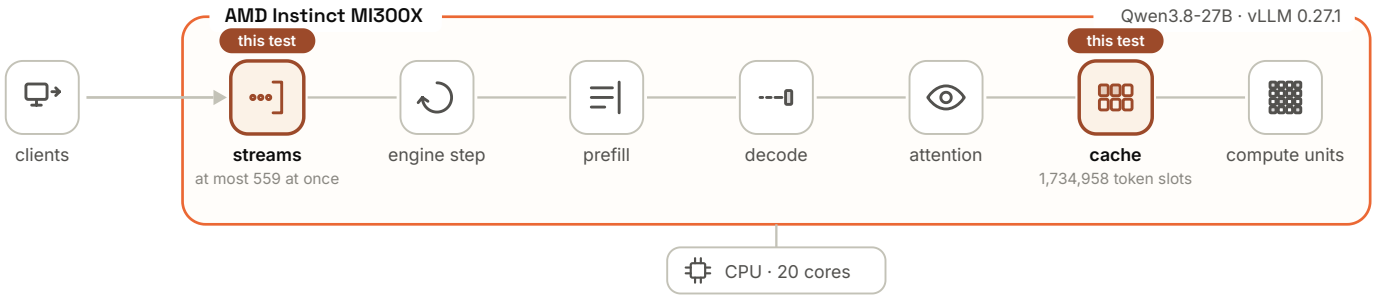

Figure 2 The card as every document in this series draws it: seven parts wired left to right, the clients outside, the host CPU below. The setting changes the attention kernel; what this plan measures is how many streams fit and what the card then writes.
Source: FACTS.json; summary.json server.kv_cache_tokens, plans.plan-staircase-top.stages[stage=up-2048].running_max

The top of the staircase, on an empty server

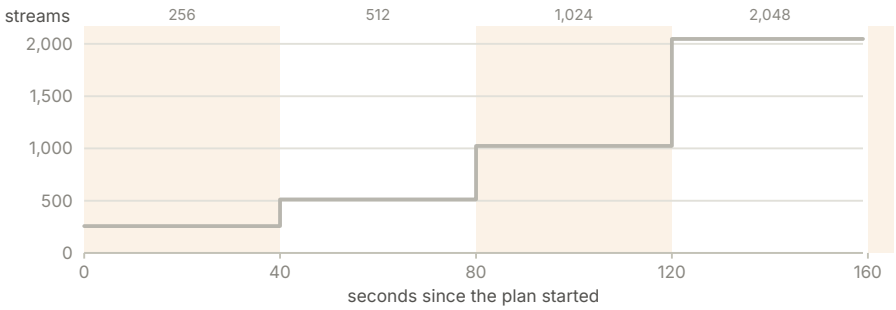

Figure 3 The plan: streams asked for over time. Each stair lasts 40 seconds; then the generator waits 60 seconds for the line to empty and stops.
Source: plan-staircase-top/timeline.json (target); plans/staircase-top.json

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. One setting differs from the series run: AMD's AITER kernels are switched on and the decoder's attention uses the flash-attention one. The server's log confirms the kernel in use.
4. The other side is not the same plan. The default kernel's server played the full staircase, from 1 stream up, and reached 256 already busy; this one starts there empty. Only the two top stairs, where both servers are full, compare one to one.
5. Beside each run a sampler read 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	20261125T215049Z-series-fa, started 2026-11-25T21:51:02Z, server ready after 97 s

KEEP IN MIND

The load is made on the same machine as the server and shares its CPU cores. Both servers had 1,024 stream slots and vLLM's default share of the card's memory, 92%.

Words used

Attention kernel
(`--attention-backend`) the GPU code that lets each token look back over the context. Four exist here: `ROCM_ATTN` (the default pick), `ROCM_AITER_FA` (AITER flash attention), `ROCM_AITER_UNIFIED_ATTN`, `TRITON_ATTN`.

AITER
AMD's library of GPU kernels for inference. Off by default in this install (`VLLM_ROCM_USE_AITER`).

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.

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`.

Full at 559 streams, and writing as much as the default kernel



Figure 4 This run in colour; in grey the same four stairs of the default kernel's full staircase, moved to the same clock. Streams running follow the request (thin steps) to 512, then stop at 559 here and at 588 there. Output is a 7-second centred mean. Shaded bands are the stairs; in the last band nothing new is sent to this server (the grey run went on with other stairs and is not drawn there).
Source: plan-staircase-top/timeline.json of this run; plan-staircase-short/timeline.json of run 20261125T201346Z-series

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									
default kernel	256	2	43.5	3,500	647	13.74	302	1.02	191
fast kernel	256	124	45.8	3,129	424	12.24	290	2.05	164
512 STREAMS ASKED FOR									
default kernel	512	25	87.0	3,913	559	7.66	584	1.20	206
fast kernel	512	132	91.6	3,203	862	6.29	646	2.13	186
1,024 STREAMS ASKED FOR									
default kernel	588	436	99.9	3,856	862	6.58	717	33.63	217
fast kernel	559	465	100.0	3,949	431	7.08	619	38.20	203
2,048 STREAMS ASKED FOR									
default kernel	588	1,460	99.9	4,096	256	6.97	624	≥ 144.63	202
fast kernel	559	1,489	100.0	3,993	510	7.16	629	≥ 99.90	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. TFLOPS is vLLM's estimate.

NOT A RESULT

The two lower stairs do not compare the kernels. Here 256 requests start in the same instant on an empty server and move together; there the server arrives at 256 with streams of every age. The lower output here (3,129 and 3,203 against 3,500 and 3,913) may be the plan as much as the kernel.

NOT A RESULT

1,536 of this run's 2,863 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 2,863 got HTTP 200.

The fast kernel keeps more memory for itself, and the cache gets less

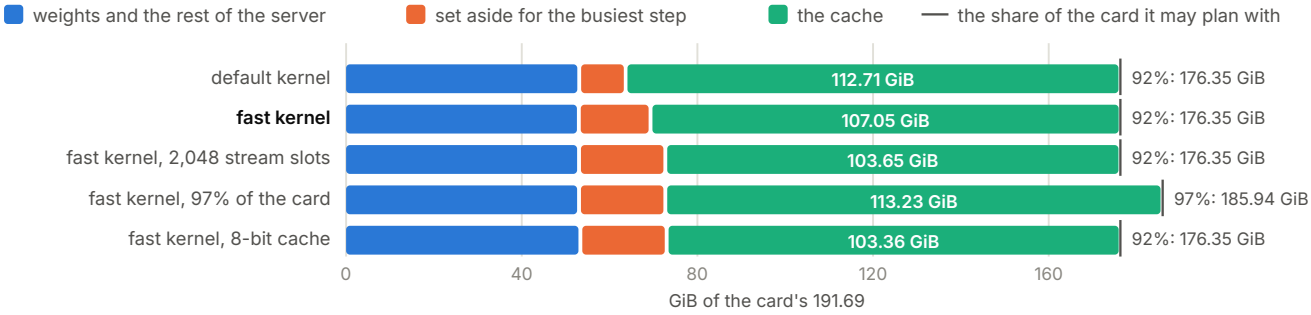

Figure 5 Where each of the five servers that played this plan put the card's memory, by its own log: what the server itself consumed, what it set aside for its busiest step, and the cache, which gets the rest of the share it may plan with. This run is in bold.
Source: server.log of each run ("Desired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

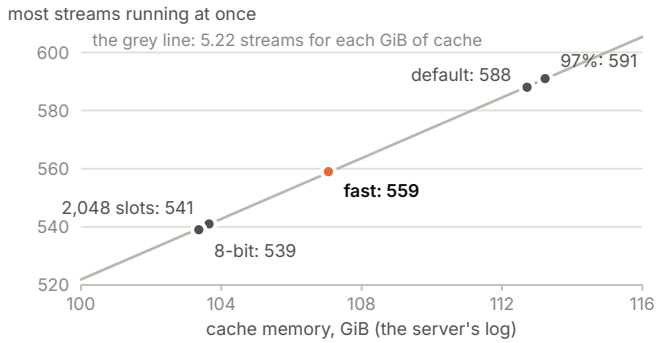

Figure 6 The most streams running at once against the cache's size in GiB, five servers: the default attention kernel, this one, and the fast kernel with 2,048 stream slots, with 97% of the card and with an 8-bit cache. They lie on one line through zero: the streams that fit follow the bytes.
Source: 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	DEFAULT KERNEL	FAST KERNEL	DIFFERENCE
Share the server may plan with	92%: 176.35	92%: 176.35	+0.00
Consumed by the server itself (weights and the rest)	53.07	53.00	-0.07
Set aside for the busiest step	10.57	16.31	+5.74
The cache: what is left of the share	112.71	107.05	-5.66
Compiled graphs, taken after the cache was sized	6.83	12.90	+6.07
All four together, on a card of 191.69	183.18	189.26	+6.08

COMPUTED

The fast kernel's server sets 16.31 GiB aside for its busiest step where the default kernel's sets 10.57 aside: 5.74 more, and its cache is 5.66 GiB smaller.

COMPUTED

This server's cache gauge moves in whole steps, 2,236 to a full cache. A running stream of this plan takes 4, and 559 × 4 = 2,236 fills it. The default kernel's gauge has 2,354 steps: 588 × 4 = 2,352, and one more stream does not fit. A step is 49.0 MiB here and 49.0 MiB there.

READING, NOT MEASUREMENT

My reading: a stream of at most 576 tokens gives an attention kernel little context to look back over, so the kernel does not decide the top. It matters where the context is long, which other plans of this run show.

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`; the three ratios are computed.

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

What I take from it

● Full of short streams, the card writes about 4,000 answer tokens a second on either attention kernel.

● The fast kernel's server runs 559 streams at most, the default kernel's 588.

○ That difference is the cache, 5.0% smaller on the fast kernel: both servers run 5.22 streams for each GiB of it.

◌ Choosing the fast kernel costs nothing at the top with short prompts, and a twentieth of the streams that fit. Where it pays is long context.

What it does not show

- The kernels below the ceiling. The stairs at 256 and 512 streams were reached in two different ways and do not compare.
- One stream alone, or a few: this plan starts at 256.
- Anything about long prompts. Those are this run's context ladder and mixed plan.
- One run each. A stair's output swings with the moment at which new prompts are read.

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						
default kernel	217.3	0.71	24.8	750	11.7	2.02
fast kernel	203.4	0.79	36.7	749	11.7	1.98
2,048 STREAMS ASKED FOR						
default kernel	202.0	0.68	50.0	751	11.7	2.00
fast kernel	207.8	0.61	17.3	748	11.8	2.00

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/plan-context-ladder/samples.jsonl` f2696347bb5b · `runs/20261125T201346Z-series/plan-kv-growth/samples.jsonl` 015baec3f8a7 · `runs/20261125T201346Z-series/plan-kv-pressure/samples.jsonl` 17363b9b3ac4 · `runs/20261125T201346Z-series/plan-mixed-overlap/samples.jsonl` 0d25a29a74d9 · `runs/20261125T201346Z-series/plan-overload-open/samples.jsonl` 497385ff88bf · `runs/20261125T201346Z-series/plan-smoke/samples.jsonl` b48c999d88a9 · `runs/20261125T201346Z-series/plan-staircase-short/samples.jsonl` cede4b79f5ca · `runs/20261125T201346Z-series/plan-staircase-short/timeline.json` 477032b70206 · `runs/20261125T201346Z-series/server.log` c008d4941d1a · `runs/20261125T201346Z-series/summary.json` aab3940eb12e · `runs/20261125T215049Z-series-fa/plan-context-ladder/samples.jsonl` a5711b1d5bc5 · `runs/20261125T215049Z-series-fa/plan-mixed-overlap/samples.jsonl` 793988ae70b7 · `runs/20261125T215049Z-series-fa/plan-smoke/samples.jsonl` 3b722aa306b9 · `runs/20261125T215049Z-series-fa/plan-staircase-top/samples.jsonl` fb46e627e0cb · `runs/20261125T215049Z-series-fa/plan-staircase-top/timeline.json` 067529026b74 · `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/server.log` 18247a4dd0d4 · `runs/20261125T225329Z-series-mem97/summary.json` ab4cc85f7ebd · `runs/20261126T013018Z-series-fa-seqs2048/server.log` 5a8a691ca70f · `runs/20261126T013018Z-series-fa-seqs2048/summary.json` 94ecccaf84b3

61 values bound by path, 6 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