On the fast kernel the full ladder holds its speed

The same nine requests as on the default kernel, with AMD's flash-attention kernel chosen by hand. With the model's context full the answer is written at 45.7 tokens a second instead of 1.8, and the first token comes after 73 seconds instead of 292.

Qwen3.8-27B · one AMD Instinct MI300X · vLLM 0.27.1 **Run** 20261125T215049Z-series-fa
Changed the attention kernel: ROCM_AITER_FA chosen by hand, with AITER switched on
Load 1,000 to 262,000 tokens in, 64 out, one request at a time, then two that do not fit

I played the context ladder again on a server that differs in one thing: the attention kernel. Instead of the one vLLM picks by itself I chose AMD's flash-attention kernel, `ROCM_AITER_FA`. The ladder holds its speed. With the model's context full the answer is written at **45.7 tokens a second**, where the default kernel gave 1.8, and the first token comes after 1 min 13 s instead of 4 min 52 s.

45.7
answer tokens a second with 262,000 tokens of context; 65.5 with 4,000

24.7 times
as fast in writing as the default kernel at 262,000 tokens, computed from the two servers' writing seconds

1 min 13 s
before the first token at 262,000 tokens; 4 min 52 s on the default kernel

3,596
prompt tokens read a second at 262,000, averaged over the prompt; 898 on the default kernel

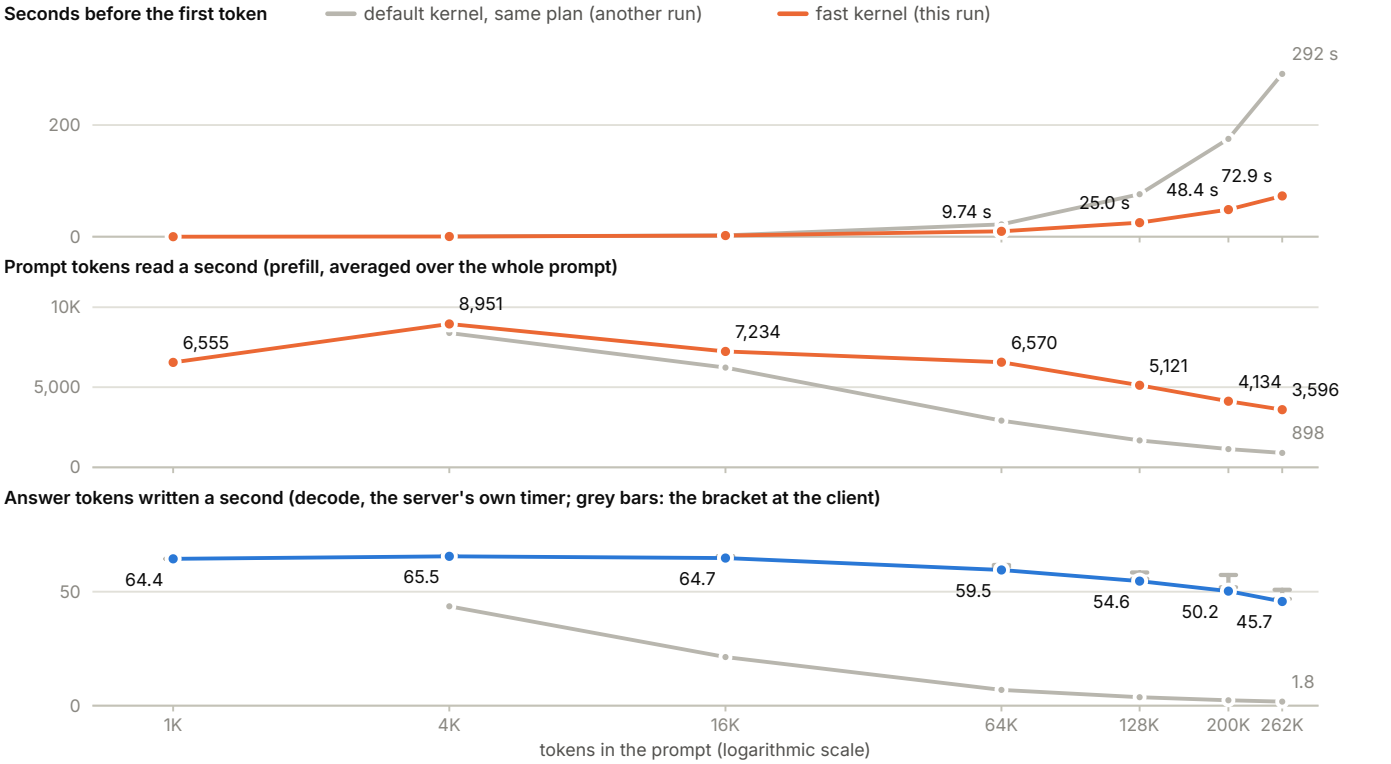


Figure 1 The seven rungs on the fast kernel in colour, and in grey the same plan on the default kernel (its last value is written in grey). Top: the wait for the first token. Middle: how fast the prompt was read, averaged over the whole prompt. Bottom: how fast the answer was written; the grey bars are the two figures the client can compute, shown only as a bracket.
Source: summary.json plans.plan-context-ladder.single_requests[] of this run and, in grey, of run 20261125T201346Z-series, both read through cross/ladder.json

MEASURED
Writing stays between 45.7 and 65.5 answer tokens a second from 4,000 to 262,000 tokens of context. All 7 requests that fit were answered, and the server counted exactly the tokens I asked for.

MEASURED
What still falls is reading: 8,951 prompt tokens a second at 4,000 tokens, 3,596 at 262,000. The 2 requests past the limit were refused at once with HTTP 400, as on the default kernel.

The machine, and the one part that was changed

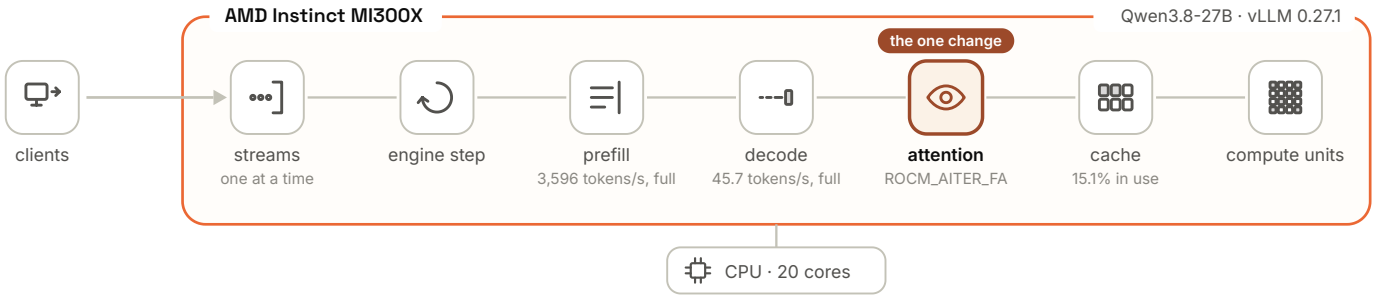


Figure 2 The card as every document in this series draws it. The plan, the card and the server are those of the context ladder on the default kernel; the attention kernel is the one change. "Full" is the 262,000-token rung.
Source: FACTS.json; summary.json server.attention_kernel and plans.plan-context-ladder

The one change

WHAT EACH SERVER REPORTED	DEFAULT KERNEL (20261125T201346Z-SERIES)	FAST KERNEL (THIS RUN)
attention kernel	ROCM_ATTN	ROCM_AITER_FA
chosen by	vLLM's own preference order	the --attention-backend setting
environment set	none	`VLLM_ROCM_USE_AITER=1`
cache, token slots	1,826,476	1,734,958
card memory taken by the start-up graphs	6.83 GiB	12.90 GiB
server ready after, s	94	97

The kernel is chosen with `--attention-backend ROCM_AITER_FA`, and that kernel needs AMD's AITER library switched on (`VLLM_ROCM_USE_AITER=1`): two settings, one change. With AITER on, the server's start-up graphs take more of the card's memory and it starts with 5.0% fewer cache slots. One request at a time never comes near that: it uses 15.1% of the cache at most. Both servers found their graphs in vLLM's compile cache, so they started in about the same time.

1. The plan sends 9 requests, one at a time: a prompt of 1,000 tokens, then 4,000, 16,000, 64,000, 128,000, 200,000 and 262,000, then two that cannot fit (262,100 and 263,000). It is the same plan file with the same seed as on the default kernel.
2. Every prompt is a list of random token ids, so its length is exact and nothing repeats. Every answer is forced to 64 tokens.
3. The next, longer prompt is sent only when the last token of the answer before has arrived. Nothing else runs on the server.
4. Beside it a sampler reads the server's counters, the card and the CPU once a second.

Prompt plus answer of each request, against the limit of 262,144 tokens

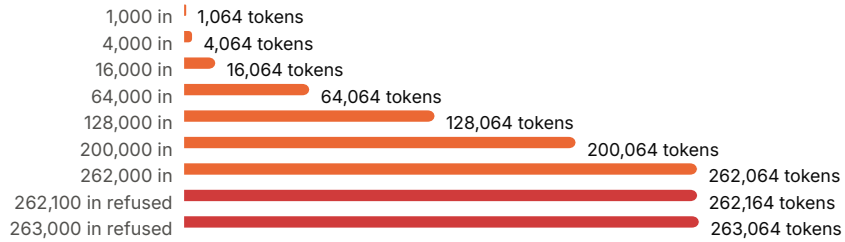


Figure 3 The plan. The prompt and the answer asked for count together, so 262,000 tokens in still fit and 262,100 do not. Red: refused.
Source: plans/context-ladder.json; summary.json server.max_model_len; cross/ladder.json (ended)

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. The generator runs inside the server's container and talks to it on the loopback address: no request leaves the machine.

Rung by rung: where the seconds go

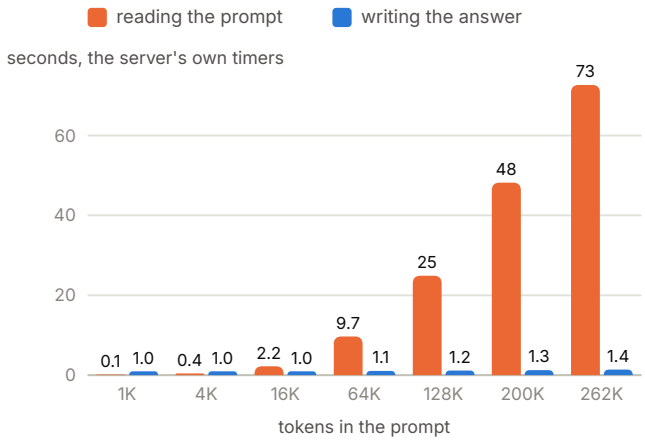


Figure 4 Seconds this server spent on each request: reading the prompt (orange) and writing 64 answer tokens (blue). Writing takes about a second at every rung.
Source: summary.json single_requests[].server_prefill_s and .server_decode_s (the server's own timers per request)

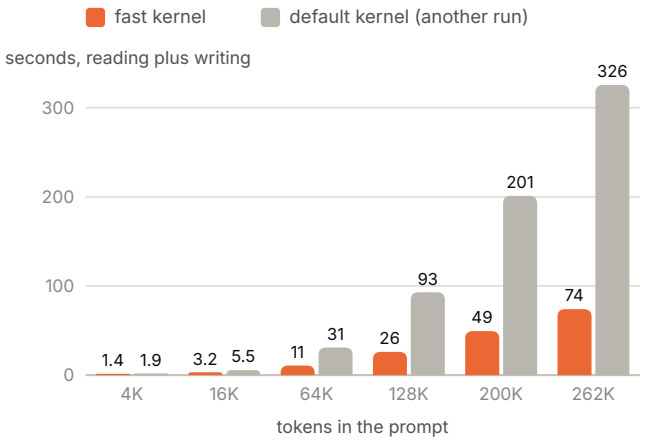


Figure 5 The whole request, reading plus writing, on the two kernels and on one scale.
Source: the same two timers, this run and run 20261125T201346Z-series (cross/ladder.json)

The exact record

PROMPT TOKENS	FIRST TOKEN, S	READING, S (SERVER)	PROMPT TOKENS/S	WRITING, S (SERVER)	ANSWER TOKENS/S (SERVER)	BRACKET AT THE CLIENT	MS PER ANSWER TOKEN	CACHE IN USE, %
1,000	0.15	0.15	6,555	0.979	64.4	64.4 to 64.4	15.5	-
4,000	0.45	0.44	8,951	0.962	65.5	65.7 to 65.7	15.3	-
16,000	2.21	2.19	7,234	0.973	64.7	65.5 to 65.5	15.4	1.1
64,000	9.74	9.69	6,570	1.059	59.5	60.8 to 61.7	16.8	3.8
128,000	25.00	24.89	5,121	1.153	54.6	55.6 to 58.4	18.3	7.5
200,000	48.38	48.18	4,134	1.254	50.2	51.9 to 57.3	19.9	11.6
262,000	72.85	72.66	3,596	1.379	45.7	46.8 to 50.8	21.9	15.1
PAST THE LIMIT: REFUSED AT ONCE (THE SERVER'S MESSAGE IS BELOW)								
262,100	HTTP 400 after 0.04 s	-	-	-	-	-	-	-
263,000	HTTP 400 after 0.04 s	-	-	-	-	-	-	-

One row per request, from `summary.json` and `cross/ladder.json`. First token: at the client, from sending to the first text chunk. Reading and writing seconds: the server's own prefill and decode timers for that request. Prompt tokens a second: the prompt's tokens divided by the first-token time. Answer tokens a second: the answer's tokens less one, divided by the writing seconds. Cache: the most seen while the rung played (one sample a second, so the shortest rungs have none).

Past the limit

MEASURED

262,100 tokens in, 64 asked back: HTTP 400 after 0.04 s, before any of the prompt was read. The server said: "This model's maximum context length is 262144 tokens. However, you requested 64 output tokens and your prompt contains 262100 input tokens, for a total of 262164 tokens. Please reduce the length of the input prompt or the number of requested output tokens. (parameter=input_tokens, value=262100)"

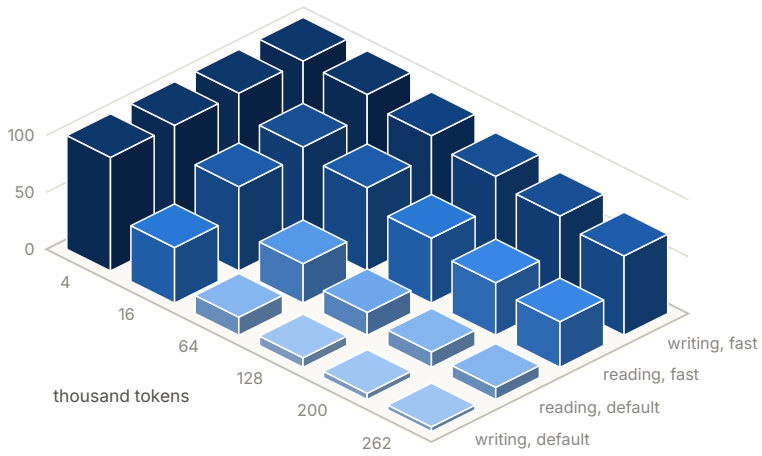
263,000 tokens in: HTTP 400 after 0.04 s, the same message with its own figures. The default kernel's server refused the same two requests in the same way.

KEEP IN MIND

Why the server's timer, and what the bracket is. These answers are random tokens, and a token that carries no visible text sends no chunk to the client: at 262,000 tokens the client received fewer chunks than the 64 tokens the server counted. Counting chunks gives too low a speed, spreading all the tokens over the time from the first chunk to the last too high a one. I quote the server's own decode timer and show those two only as a bracket. On this run the server's figure sits just below the bracket at most rungs, because tokens without text at either end of an answer shorten the client's window.

How much speed is kept, on both kernels

Speed kept, % of the speed at 4,000 tokens



thousand tokens in the prompt

	4	16	64	128	200	262
writing, fast	100	99	91	83	77	70
reading, fast	100	81	73	57	46	40
reading, default	100	74	35	20	14	11
writing, default	100	49	16	8.4	5.5	4.2

darker = more: speed kept, %

Figure 7 The same shares, exactly.
Source: computed from cross/ladder.json

Writing is compared by the server's writing seconds, which is exact where the rounded speeds are not: every request writes the same 64 tokens.

Figure 6 Each speed as a share of its own value at 4,000 tokens: writing and reading on the fast kernel (the two back rows) and on the default kernel (the two front rows; "default" is the kernel vLLM picks by itself). A view only: the exact values are in the matrix beside it.
Source: computed from single_requests[].prefill_tok_s and .server_decode_s of this run and of run 20261125T201346Z-series

COMPUTED

From 4,000 to 262,000 tokens the fast kernel keeps 69.7% of its writing speed, the default kernel 4.2%.

COMPUTED

In reading the fast kernel keeps 40.1% of its speed and the default kernel 10.7%.

Rung against rung

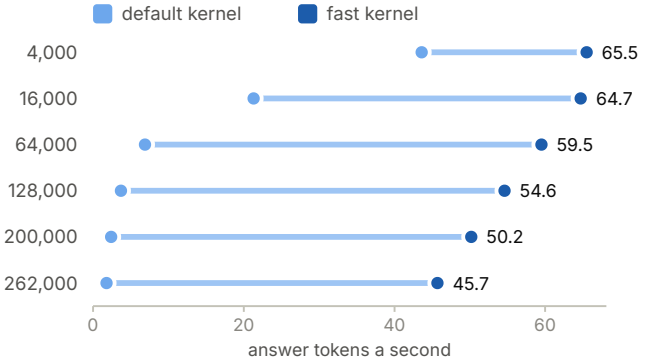


Figure 8 Answer tokens a second at each rung on the two kernels. The number is the fast kernel's.
Source: summary.json single_requests[].decode_tok_s_server of this run and of run 20261125T201346Z-series

PROMPT TOKENS	FIRST TOKEN, S: DEFAULT	FAST	SOONER BY	ANSWER TOKENS/S: DEFAULT	FAST	FASTER BY
4,000	0.48	0.45	1.0×	43.6	65.5	1.5×
16,000	2.57	2.21	1.1×	21.3	64.7	3.0×
64,000	21.96	9.74	2.2×	6.9	59.5	8.5×
128,000	75.94	25.00	3.0×	3.7	54.6	14.9×
200,000	175.19	48.38	3.6×	2.4	50.2	21.0×
262,000	291.61	72.85	4.0×	1.8	45.7	24.7×

"Faster by" divides the two servers' writing seconds for the same 64 tokens. Multiples are cut to one decimal, never rounded up.

COMPUTED

With the context full the fast kernel writes 24.7 times as fast and reaches the first token 4.0 times as soon. At 4,000 tokens the two kernels are close: 1.5 times in writing, 1.0 times in the wait.

KEEP IN MIND

Like for like: the same plan file and seed, the same card, one request at a time. What differs is the kernel, with AITER switched on for it. The default kernel's 1,000-token rung is left out of every comparison: its first-token time was disturbed by the plan before it.

Why it holds: the look back is still there, and far cheaper

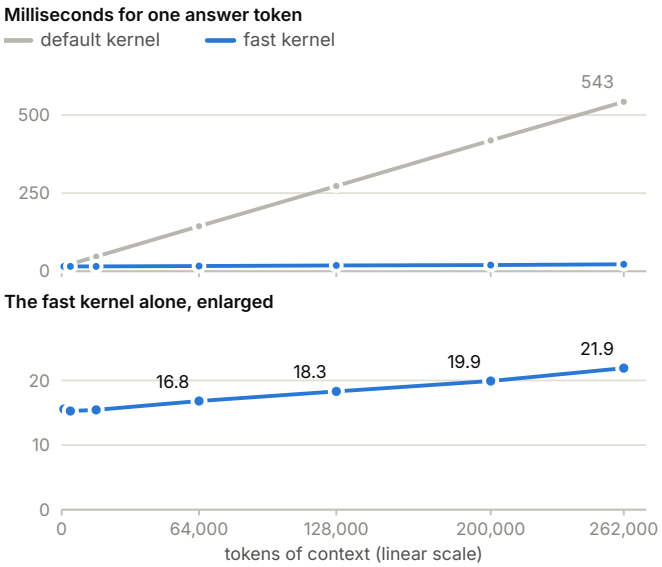


Figure 9 Writing. The time for one answer token against the context, on plain scales. Top: both kernels on one scale. Bottom: the fast kernel alone, enlarged: its line rises too, gently.
Source: computed from single_requests[].server_decode_s of this run and of run 20261125T201346Z-series (cross/ladder.json)

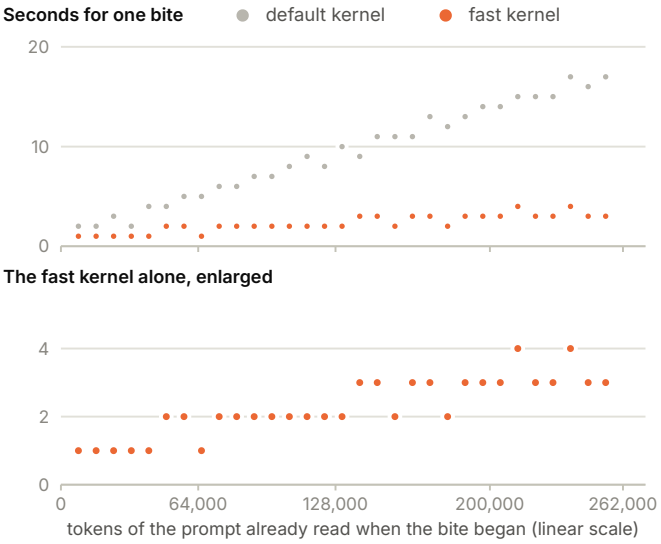


Figure 10 Reading. The seconds each bite of the 262,000-token prompt took, against what had been read before it. Whole seconds: the record has one sample a second, which shows as steps on the fast kernel.
Source: plan-context-ladder/timeline.json of both runs: the seconds between two moves of the server's work counter (est_tflops), which moves once per bite

MEASURED

The 262,000-token prompt went in as 32 bites of at most 8,192 tokens. The first three took 1.0 s each on average, the last three 3.3 s; on the default kernel the same three took 16.7 s. The last 62,000 tokens of the prompt went in at 2,533 tokens a second.

COMPUTED

One answer token took 15.3 ms at 4,000 tokens of context and 21.9 ms at 262,000: 0.026 ms more for every 1,000 tokens of context. On the default kernel that figure is 2.01 ms, 78 times as much.

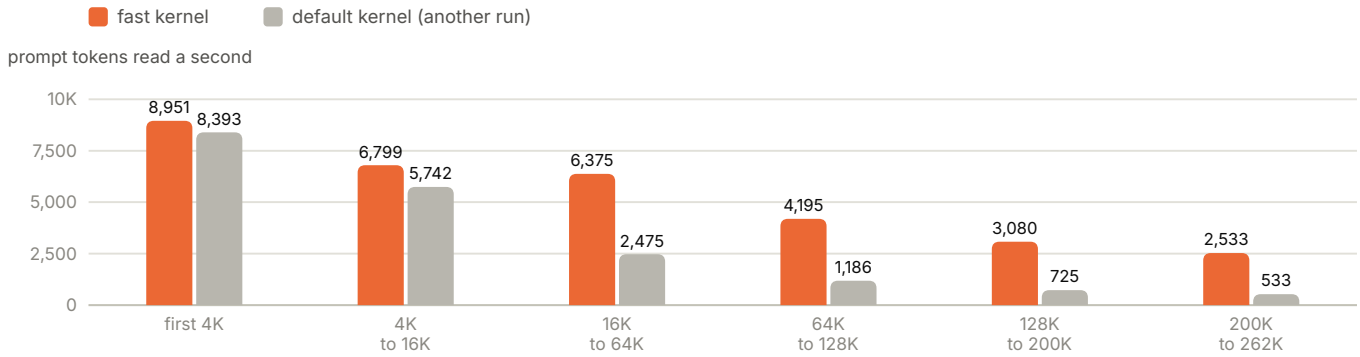


Figure 11 Reading once more, from the first-token times alone: how fast the tokens that each rung adds went in, on the two kernels. The first 4,000 tokens are read at nearly the same speed on both; the gap opens with the context.
Source: computed from single_requests[].first_token_s of neighbouring rungs: tokens added divided by seconds added, this run and run 20261125T201346Z-series

READING, NOT MEASUREMENT

The fast kernel does not remove the look back over the context: each bite and each answer token still costs more as the context grows. It makes that look back far cheaper. What is left is reading, which still slows to 40.1% of its first speed: a long prompt is more work on any kernel.

COMPUTED

The 262,000-token prompt is 4.0 times as long as the 64,000-token one. On the fast kernel it took 7.4 times as long to read; on the default kernel 13.3 times.

The same settings on another start of the server

PROMPT TOKENS	FIRST TOKEN, S: THE OTHER START	THIS RUN	DIFFERENCE	WRITING, S: THE OTHER START	THIS RUN	DIFFERENCE
4,000	0.445	0.447	+0.4%	0.957	0.962	+0.5%
16,000	2.224	2.212	-0.5%	0.972	0.973	+0.1%
64,000	9.691	9.742	+0.5%	1.053	1.059	+0.6%
128,000	24.458	24.997	+2.2%	1.149	1.153	+0.3%

The same kernel and the same settings on another start of the server (run 20261125T212632Z-series-attn-rocm-aiter-fa), the same four prompts. The two starts differ by at most 2.2% in the wait for the first token and 0.6% in the writing seconds: that is the spread to keep in mind when two numbers in this document are close.

The same kernel past the model's own limit

MEASURED

Another run stretched the context limit to 1,000,000 tokens on this kernel and sent three longer prompts. The first token came after 3 min 38 s at 500,000 tokens, 7 min 28 s at 750,000 and 12 min 26 s at 990,000; the answers were written at 37.2, 30.8 and 26.4 tokens a second.

KEEP IN MIND

That run (20261125T220624Z-ctx1m) has its own document. It shares the kernel and the card with this one; the context setting, the plan's seed and the size of the cache differ. Its numbers carry on the curves of this ladder, they are not a rung of it.

Underneath

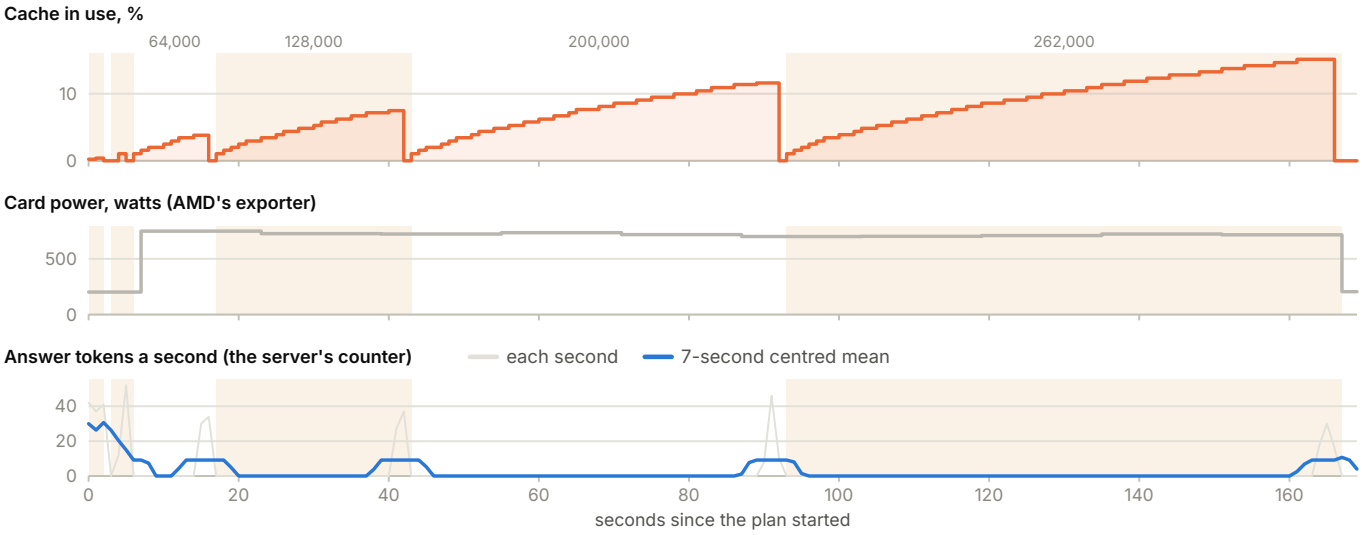


Figure 12 The whole plan, second by second. Top: the cache filling as each prompt goes in. Middle: the card's power. Bottom: answer tokens, each second in grey and a 7-second centred mean in blue. Shaded bands are the rungs, labelled with their prompt length.
Source: plan-context-ladder/timeline.json (the server's gauges and counters every second; AMD's device exporter, which refreshes every 11 to 16 s)

PROMPT TOKENS	WATTS WHILE READING	TFLOPS, VLLM'S ESTIMATE	% OF AMD'S STATED PEAK	THE SAME ON THE DEFAULT KERNEL	CPU BUSY, %	ENGINE CORES
128,000	724	784	60.3	207	12.0	2.29
200,000	721	854	65.7	206	11.3	2.14
262,000	711	926	71.2	206	11.0	2.10

The rungs long enough for the exporter. Watts are the mean of its readings while the prompt was read, leaving out the first 13 seconds. TFLOPS is vLLM's estimate from the model's shape over the whole request, not a hardware counter, and the peak it is divided by is AMD's stated figure. The estimated work is the same on both kernels, so the two TFLOPS columns differ exactly as the times do.

READING, NOT MEASUREMENT

vLLM's estimate counts too much attention for this model. What it adds for every bite already read is attention over the context in 64 layers; the model's configuration has full attention in 16 of its 64. At 262,000 tokens 80% of the estimated work is that attention. Counted for 16 layers, the same arithmetic gives 359 TFLOPS, 27.6% of AMD's stated peak. My reading, not vLLM's figure.

What I take from it

● With AMD's flash-attention kernel chosen by hand, a request that fills the model's context waits 1 min 13 s for its first token and is then written at 45.7 tokens a second.

○ That is 24.7 times as fast in writing as the kernel vLLM picks by itself, and 4.0 times as soon to the first token.

● Writing holds between 45.7 and 65.5 tokens a second over the whole ladder. Reading still falls, from 8,951 to 3,596 prompt tokens a second.

● Requests past the limit are refused at once on this kernel too.

◌ The collapse on the default kernel was the kernel's, not the model's or the card's.

Not results, and limits

- Every request of this plan is a result: 7 answered, 2 refused. Nothing was cut and nothing failed.
- Speed and behaviour only. Prompts are random tokens and answers are forced to 64 tokens, so nothing here says how well the model answers at this length.
- One request at a time on an otherwise idle server, each rung played once. Writing speed is taken over 64 tokens, about a second.
- This server starts with fewer cache slots than the default kernel's. That matters for how many streams fit at once, which is another test of this series.

● 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.

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

First token
(time to first token) from sending the request to receiving the first answer token. It contains waiting in line and prefill.

Context
all tokens a request holds: prompt plus answer so far. This model's native limit is 262,144.

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.

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`).

Linear-attention layers
48 of this model's 64 layers keep a fixed-size state per stream instead of looking back over every token; the other 16 are full attention.

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.

TFLOPS
trillions of arithmetic operations per second. Here always vLLM's estimate from the model's shape (`--enable-mfu-metrics`), not a hardware counter. It counts attention in all 64 layers where this model has it in 16: 1.7% high for short prompts, over half too high from 128,000 tokens of context.

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

`FACTS.json` 160f5ca0d495 · `GLOSSARY.md` 98323be796af · `cross/compute-estimate.json` b36121710947 · `cross/ladder.json` 3511faec1269 · `facts/box/20261125T192138Z/model-config.json` 191e0af23210 · `plans/context-ladder.json` 2fe320742325 · `runs/20261125T201346Z-series/plan-context-ladder/timeline.json` 7ee787305964 · `runs/20261125T201346Z-series/summary.json` aab3940eb12e · `runs/20261125T215049Z-series-fa/plan-context-ladder/requests.jsonl` 02d7bb77f5a9 · `runs/20261125T215049Z-series-fa/plan-context-ladder/timeline.json` 6d9043e58e9a · `runs/20261125T215049Z-series-fa/summary.json` ec0c01913c37 · `runs/20261125T220624Z-ctx1m/summary.json` dc630cf24d26

72 values bound by path, 12 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