56 tokens a second for one stream, 1,327 for 256 together

The recipe's own command with nothing changed: the baseline every other run is judged against. 256 streams get 23.6 times the output of one, each of them a tenth of the speed, and wait six seconds on average for a first token.

Qwen3.8-27B · one AMD Instinct MI300X · vLLM 0.27.1 **Run** 20261125T184034Z-recipe **Changed** nothing: the vLLM recipe's own command
Load 1,024 tokens in, 256 out; 1, 8, 32, 128 and 256 streams

I started the server with the vLLM recipe's own command, changed nothing, and asked it for 1, 8, 32, 128 and 256 streams at once: 1,024 tokens in and 256 out, every time. One stream gets 56.2 answer tokens a second. 256 streams together get 1,327.3, 23.6 times as much, while each of them slows to 5.2 a second and waits 6.2 s on average for its first token. Not one of the 1,704 requests failed.

56.2
answer tokens a second for one stream alone
first token after 0.14 s

1,327.3
for 256 streams together
6,636.6 a second counting the prompt tokens

23.6 times
the output of one stream, from 256 times the streams

5.2
answer tokens a second for each of the 256
first token after 6.2 s on average

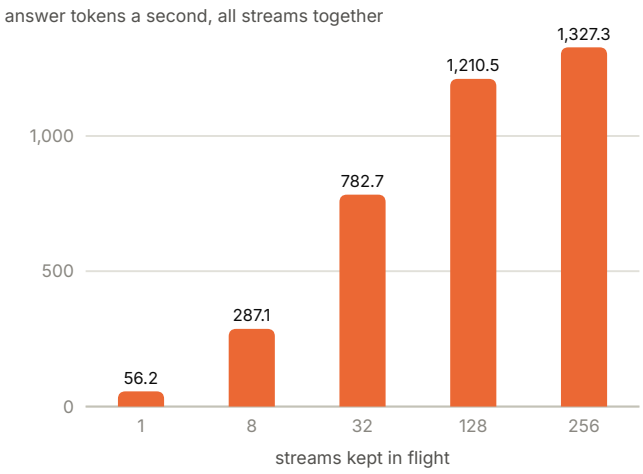

Figure 1 Answer tokens a second for all streams together, by level. Each stair holds more streams than the one before; the last two are almost level.
Source: summary.json sweeps.random-1k-256.levels[].output_tok_s (the load tool's result)

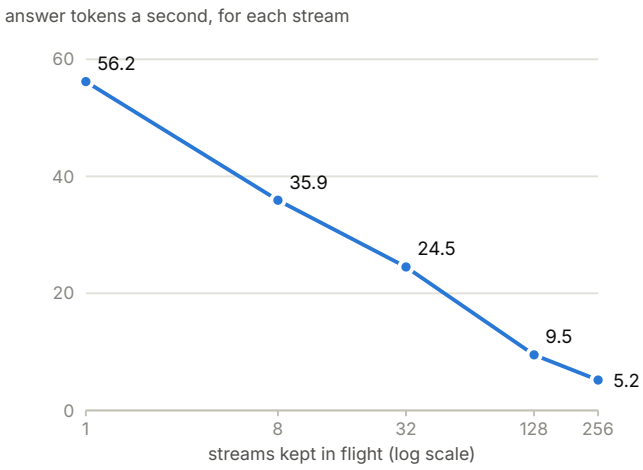

Figure 2 The same output divided by the streams: what one stream gets.
Source: summary.json sweeps.random-1k-256.levels[].output_tok_s_per_stream

1 stream
56.2 answer tokens a second together, 56.2 each. First token after 0.14 s on average.

8 streams
287.1 answer tokens a second together, 35.9 each. First token after 1.29 s on average.

32 streams
782.7 answer tokens a second together, 24.5 each. First token after 2.10 s on average.

128 streams
1,210.5 answer tokens a second together, 9.5 each. First token after 4.10 s on average.

256 streams
1,327.3 answer tokens a second together, 5.2 each. First token after 6.20 s on average.

COMPUTED
The stairs flatten. Four times the streams, from 32 to 128, bring 54.6% more answer tokens a second; twice the streams again, from 128 to 256, bring 9.6% more.

MEASURED
The price is paid by each stream: 56.2 tokens a second alone, 24.5 among 32, 5.2 among 256. A whole request takes 4.6 s alone and 49.2 s among 256.

READING, NOT MEASUREMENT
My reading of the flat top: it is not the card's top for writing answers. In this load four tokens in five are prompt tokens, and while the server takes in prompts it writes almost nothing. The page on why the stairs flatten gives the grounds.

The machine, and the part this test leans on

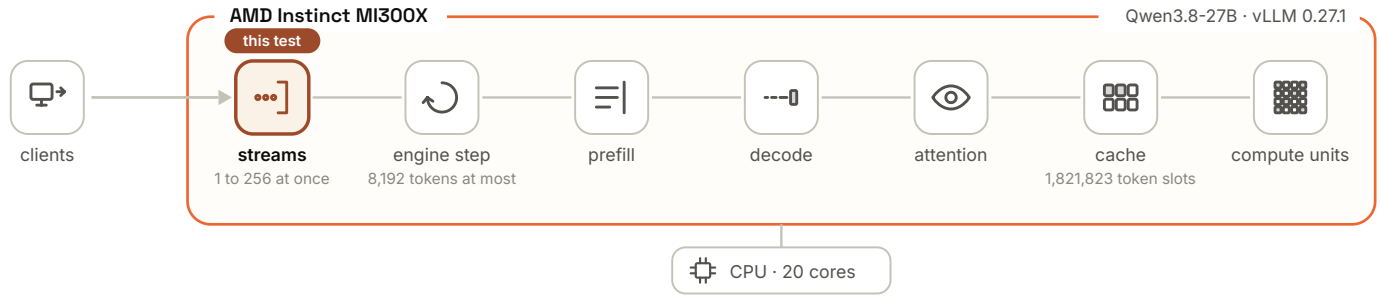

Figure 3 The card as every document in this series draws it. This test asks how many streams one card answers at once, and what each gets.
Source: FACTS.json; summary.json server.kv_cache_tokens

One loop, repeated

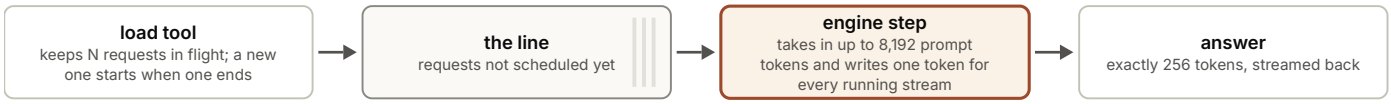

Figure 4 A closed loop: the load tool keeps a fixed number of requests in flight, and the server sets the pace.
Source: summary.json sweeps.random-1k-256.load_tool_arguments; FACTS.json server_defaults_on_this_card

1. The server is started with the recipe's own command for one card: no environment switch, no tuning argument.
2. vLLM's own load tool keeps N requests in flight: N is 1, 8, 32, 128, then 256. A level is 4 requests per stream, at least 8, after 2 warm-up requests.
3. Every prompt is 1,024 random tokens; every answer is forced to exactly 256. The plain completions endpoint: no chat template.
4. The server's counters are saved before and after each level, and the card is read every 2 s.

What was played

STREAMS	REQUESTS	FAILED	SECONDS	
1	8	0	36.4	a result
8	32	0	28.5	a result
32	128	0	41.9	a result
128	512	0	108.3	a result
256	1,024	0	197.5	a result
512				stopped: not a result

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	20261125T184034Z-recipe, started 2026-11-25T18:40:34Z, server ready after 197 s

What the server chose by itself

Attention kernel	ROCM_ATTN, vLLM's own pick
Tokens per engine step	8,192 at most
Streams at once	1,024 at most
Share of card memory	92.0%, 176.35 GiB
Cache	1,821,823 token slots in 112.41 GiB
Start	ready after 197 s, 36.3 s compiling

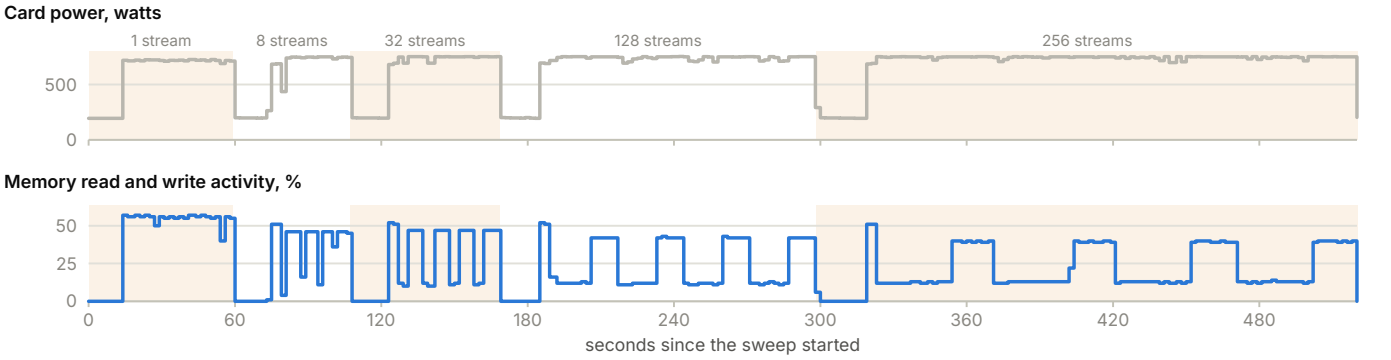

Figure 5 The sweep on the card's own meter, read every 2 s. Each band is one run of the load tool. Under load the card draws full power at every level, one stream included; at 128 and 256 streams its memory activity moves between two values, which the page on why the stairs flatten explains.
Source: random-1k-256/gpu-samples.txt (the card's power and memory activity every 2 s); level marks from c1.t0 to c256.t1

The exact record

STREAMS	ANSWER TOKENS/S	PER STREAM	WITH PROMPTS	FIRST TOKEN S: MEDIAN	MEAN	99TH	GAP MS: MEDIAN	MEAN	99TH	REQUEST S	IN LINE S	PROMPT S	ANSWER S
1	56.2	56.2	281.0	0.14	0.14	0.14	17.3	17.3	17.8	4.6	0.00	0.13	4.41
8	287.1	35.9	1,435.3	0.93	1.29	2.66	22.6	22.9	23.2	7.1	0.00	1.19	5.79
32	782.7	24.5	3,913.3	2.25	2.10	3.79	26.1	32.7	98.3	10.5	0.16	1.54	8.30
128	1,210.5	9.5	6,052.3	2.92	4.10	14.98	47.2	89.8	997.2	27.0	1.34	1.92	22.84
256	1,327.3	5.2	6,636.6	2.98	6.20	29.56	76.6	168.7	1,044.8	49.2	3.25	1.99	42.94

One row per level. Tokens a second, first-token times, gaps and whole-request times are the load tool's, measured at the client. The last three columns are the server's own timers per request, warm-up requests included. This early run kept no record per request and no CPU or network samples.

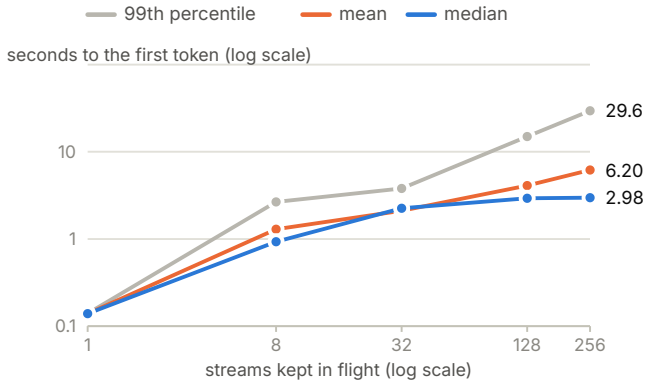

Figure 6 Seconds from sending a request to its first token. The median settles near three seconds; the mean and the slowest requests keep climbing with the level.
Source: random-1k-256/c1.json to c256.json (median_ttft_ms, mean_ttft_ms, p99_ttft_ms)

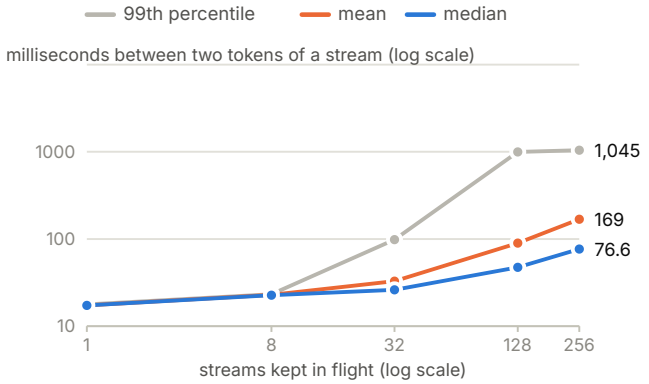

Figure 7 Milliseconds between two tokens of one stream. The median is one engine step that writes answers only. From 32 streams up the slowest gaps are several times the median, at 128 and 256 streams more than ten times: steps that also take in prompts.
Source: random-1k-256/c1.json to c256.json (median_itl_ms, mean_itl_ms, p99_itl_ms)

MEASURED

At 256 streams half of the requests have their first token within 2.98 s, one in ten waits more than 19.0 s and one in a hundred more than 29.6 s. The server's own timers put 3.25 s of that in line and 1.99 s in reading the prompt, per request on average.

MEASURED

A stream among 256 gets a token every 76.6 ms at the median, and waits 864 ms or more for one token in ten. Alone it gets one every 17.3 ms, and its slowest gap is 17.8 ms.

MEASURED

The long waits are the level's start. In the probe run's record of the same level, the first 256 requests arrive together and wait 15.3 s for a first token at the median, 31.0 s at most. The 768 that follow, each starting when another ends, wait **2.95 s at the median** and 4.1 s at most. The level's mean mixes the two: 6.2 s there, 6.2 s here.

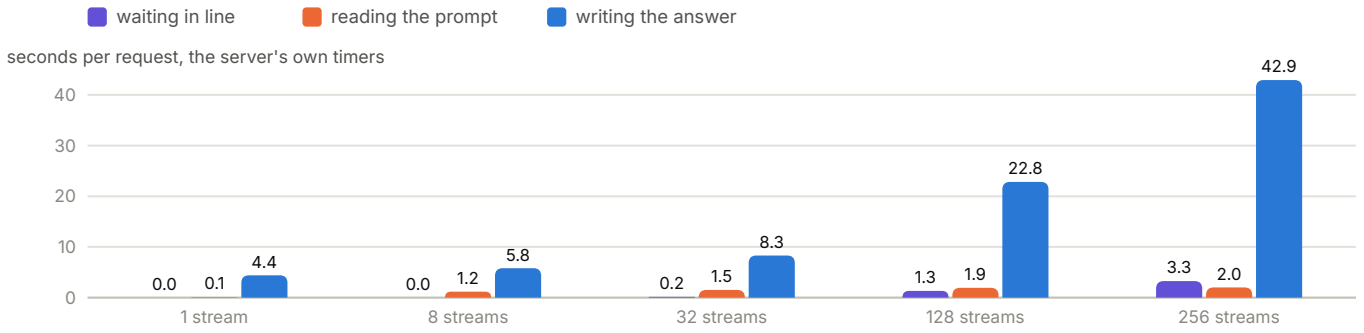

Figure 8 Where a request's time goes, by the server's own timers. Reading the prompt takes about two seconds from 32 streams up; writing the answer takes longer with every level, because each stream gets fewer steps a second and shares them with the prompts of others.
Source: summary.json sweeps.random-1k-256.levels[].server_queue_s_per_request, server_prefill_s_per_request, server_decode_s_per_request (the server's counters across each level, warm-up requests included)

A level's mean hides its best second

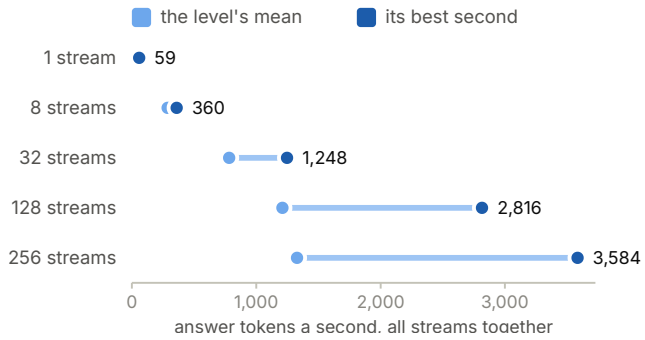

Figure 9 Each level's mean output and the most it wrote in any one second. From 32 streams up the best second is far above the mean.
Source: random-1k-256/c1.json to c256.json (output_throughput, max_output_tokens_per_s)

MEASURED

In its best second the 256-stream level wrote 3,584 answer tokens, 14 for each stream. Its mean is 1,327.3: 37.0% of that.

COMPUTED

Of the 6,636.6 tokens a second the server handles at 256 streams, 19.9% are answer tokens. The rest are prompts: four tokens in five.

STREAMS	MEAN OF THE LEVEL	BEST SECOND	BEST SECOND, PER STREAM	MEDIAN GAP, MS	THAT IS STEPS A SECOND	MEAN AS A SHARE OF THE BEST
1	56.2	59	59	17.3	57.7	95.2%
8	287.1	360	45	22.6	44.2	79.7%
32	782.7	1,248	39	26.1	38.3	62.7%
128	1,210.5	2,816	22	47.2	21.2	42.9%
256	1,327.3	3,584	14	76.6	13.1	37.0%

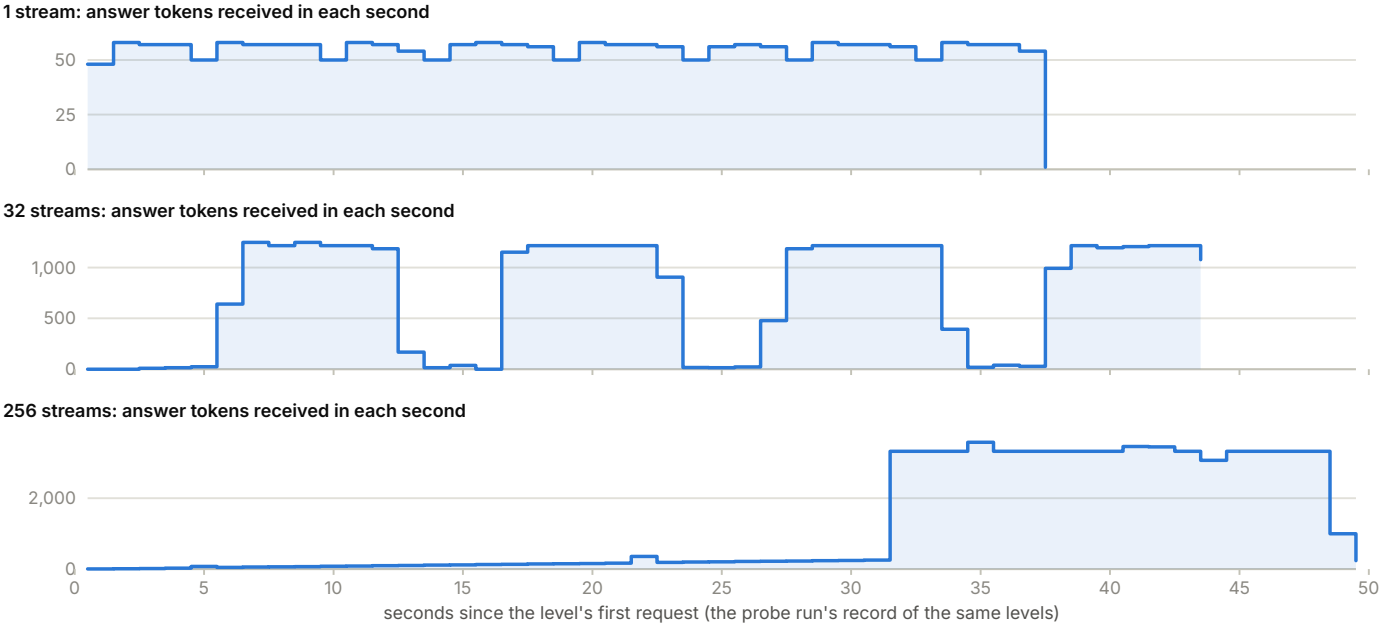

Figure 10 The first 50 seconds of three levels, second by second. One stream writes steadily. 32 streams write in bursts between short stretches of taking in prompts. 256 streams take in prompts for half a minute, writing almost nothing, and then write at full speed. This run kept no record per request: these are the same levels as the probe run played them, with the same serving settings and instruments on.
Source: runs/20261125T192827Z-probe/random-1k-256/c1.json, c32.json, c256.json (each request's first token and every gap after it)

MEASURED

In the probe run's record of the 256-stream level, 64.6% of the seconds are spent taking in prompts, with about 239 answer tokens written in a second; in the rest the card writes about **3,328 a second**. Its mean over the level was 1,324.6, against 1,327.3 here.

READING, NOT MEASUREMENT

A level here is two loads taking turns. An engine step that takes in prompts carries up to 8,192 tokens and lasts about a second; every stream already answering gets one token from it. So 1,327 a second is what is left for answers when four tokens in five are prompts, not what the card can write.

The same load on six other servers

Every run that played this load, level by level, with the baseline in colour. Each differs from the baseline in what its name says.

One stream

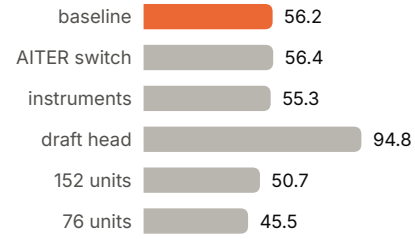

Figure 11 Answer tokens a second for one stream. Only the draft head and the masks move it.
Source: cross/levels.json

32 streams

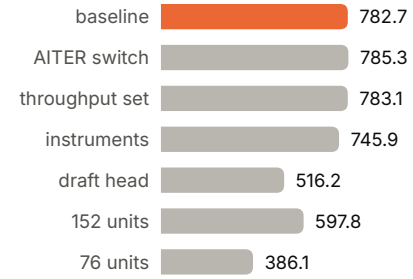

Figure 12 The one level every run played.
Source: cross/levels.json

256 streams

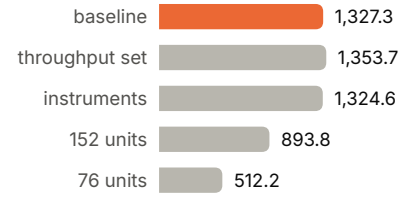

Figure 13 The top: the settings leave it where it is.
Source: cross/levels.json

streams kept in flight

	1	8	32	128	256	512
baseline	56.2	287.1	782.7	1,210.5	1,327.3	
AITER switch	56.4	306.0	785.3			
throughput set			783.1	1,222.1	1,353.7	1,384.9
instruments	55.3		745.9		1,324.6	
draft head	94.8		516.2			
152 units	50.7		597.8		893.8	
76 units	45.5		386.1		512.2	

darker = more: answer tokens a second, all streams together

Figure 14 The same values, exactly, for every level. Empty cells: that run did not play that level, or the level is not a result.
Source: cross/levels.json (each run's summary.json)

RUN	WHAT DIFFERS FROM THE BASELINE	REQUESTS PER STREAM
AITER switch	AMD's AITER kernels switched on (one environment switch)	4, as here
throughput set	AITER, throughput mode, more of the card's memory and tokens streamed in groups: four things at once	4, as here
instruments	vLLM's estimates and counters on, a sampler beside it	4, as here
draft head	the model's draft head guessing 3 tokens ahead, estimates on	4, as here
152 units, 76 units	half, and three quarters, of the card's compute units masked off; estimates on	2

MEASURED

At 256 streams the throughput settings give 1,353.7 answer tokens a second and the instruments run 1,324.6, against 1,327.3 here. At one stream the AITER switch gives 56.4 against 56.2. Its 8-stream level came out 6.5% higher, in 32 requests played once in each run.

READING, NOT MEASUREMENT

Nothing in the serving settings moved the top of this curve. What moved it was taking arithmetic away from the card, and letting the model guess ahead. Each of those runs has its own document, where it is compared like for like.

NOT A RESULT

Left out of the matrix: the 512-stream level of this run and the 128-stream level of the AITER run, both stopped by hand, and the 1,024-stream level of the throughput run, where the load tool ran out of open files and most requests failed on the client side.

What I take from it

- One stream gets 56.2 answer tokens a second, a token every 17.3 ms, with its first token after 0.14 s.
- 256 streams together get 1,327.3 a second, and 6,636.6 counting prompt tokens.
- That is 23.6 times the output of one stream, and 9.2% of its speed for each stream.
- The first token takes 6.2 s on average at 256 streams, and 29.6 s for the slowest in a hundred.
- Nothing failed and nothing was refused at any level.

READING, NOT MEASUREMENT

The flat top of this sweep belongs to this load. With prompts of 64 tokens instead of 1,024, the same card wrote 3,499.7 answer tokens a second at 256 streams (the staircase of the series run, by the server's counter). A longer prompt for the same answer moves the top down; it does not mean the card is full.

KEEP IN MIND

The baseline is a reference, not a recommendation. Its attention kernel is the one vLLM picks by itself, and later runs found a faster one for long contexts. With prompts this short the choice matters little.

Not results, and limits

NOT A RESULT

The 512-stream level was stopped by hand after the 256 level. The first attempt at the sweep finished no level: the load tool could not find the tokenizer under the server's short model name, and was started again with the full name.

- One sweep, each level played once.
- Prompts are random tokens, so nothing repeats between requests: a cache of shared prefixes or a draft head cannot show here. This is speed and behaviour, nothing about the quality of answers.
- The loop is closed: at high stream counts the first-token time is mostly waiting in line.
- No record per request, no CPU or network samples in this early run.

● 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

Level
one step of a sweep: a fixed number of streams kept in flight until the level's requests are used up.

Closed loop
the load keeps N requests in flight; a new one starts only when one ends. The server sets the pace.

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

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 · `cross/levels.json` 27321854fd2f · `plans/staircase-short.json` 6a2be1bf2346 · `runs/20261125T184034Z-recipe/random-1k-256/c1.json` 7486559e63d4 · `runs/20261125T184034Z-recipe/random-1k-256/c1.t0` 1710fab1d5f6 · `runs/20261125T184034Z-recipe/random-1k-256/c1.t1` 807160081f1d · `runs/20261125T184034Z-recipe/random-1k-256/c128.json` 45c22ee34d8b · `runs/20261125T184034Z-recipe/random-1k-256/c128.t0` 3429606d52e7 · `runs/20261125T184034Z-recipe/random-1k-256/c128.t1` 5c7c9258c6a5 · `runs/20261125T184034Z-recipe/random-1k-256/c256.json` 2150cae7d6da · `runs/20261125T184034Z-recipe/random-1k-256/c256.t0` 7d90046b581f · `runs/20261125T184034Z-recipe/random-1k-256/c256.t1` 511cc7440c7c · `runs/20261125T184034Z-recipe/random-1k-256/c32.json` 0d4fb62fe038 · `runs/20261125T184034Z-recipe/random-1k-256/c32.t0` b80b35a0a264 · `runs/20261125T184034Z-recipe/random-1k-256/c32.t1` 3837c10b2542 · `runs/20261125T184034Z-recipe/random-1k-256/c8.json` 9286ae635eb5 · `runs/20261125T184034Z-recipe/random-1k-256/c8.t0` fa95094d980f · `runs/20261125T184034Z-recipe/random-1k-256/c8.t1` 0f8fc59695b1 · `runs/20261125T184034Z-recipe/random-1k-256/gpu-samples.txt` 84524f003fa1 · `runs/20261125T184034Z-recipe/summary.json` f93dd689d428 · `runs/20261125T185511Z-aiter/summary.json` fd7bee7efee0 · `runs/20261125T190633Z-aiter-max/summary.json` 409c92f9bb01 · `runs/20261125T192827Z-probe/random-1k-256/c1.json` 37f8347fd437 · `runs/20261125T192827Z-probe/random-1k-256/c256.json` 2306cd81b021 · `runs/20261125T192827Z-probe/random-1k-256/c32.json` e5ea8f77b051 · `runs/20261125T192827Z-probe/summary.json` 064f25b0020b · `runs/20261125T201346Z-series/summary.json` aab3940eb12e

113 values bound by path, 14 figures drawn from data. built 2026-11-26 02:37Z, 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