PURETENSOR
INTELLIGENCE DISTILLED

Twenty Thousand Tokens a Second, Thirteen for Each Reader

Aggregate throughput on six workstation GPUs, and one prompt that became a 1,477-page handbook in 4

Paper: PT-R-2026-073 v1.0

Author: PureTensor Ltd

Date: 13 November 2026

Status: Published

Abstract

In its best second, six workstation GPUs, each running its own copy of Qwen3.8-27B in NVIDIA's 4-bit build, generated 20,456.0 tokens per second. Over a 30-second window the sustained rate was 19,662.7 tokens per second, and across the 1,536 concurrent streams that produced it each stream received 12.8 tokens per second. A stream running alone on one card gets 73.1. The headline is real, yet no reader of the output sees anything like it: it is the sum over 1,536 streams, each running far slower than a lone stream.

Such a figure is aggregate throughput: the total tokens generated per second by all requests served at the same moment (a stream is one request receiving its answer token by token). "Sustained" is the average over a window in which the system was kept full; "best second" is the highest one-second span inside it. On one card the total rises from 73.1 tokens per second at 1 stream to 3,368.7 at 256 while each stream falls to 18% of its solo rate, and a decoding step costs 12.94 ms plus 0.2451 ms per stream. Past 256 streams the total falls. The model's own drafting head (speculative decoding, where cheap guesses are verified in bulk) more than doubles a lone stream, 160.1 against 73.1, and loses to plain decoding from 64 streams.

Six cards gave 5.71 times the best single-card figure. One prompt fanned out into 1,536 section requests finished in 43.61 seconds as a 1,477-page handbook, which we publish unedited as a PDF with this paper; Section 8 lists its defects. We made two measurement faults on the way, and Section 9 describes both. The evidence is one day, three runs for the headline, short prompts and capped answers; Section 10 states what survives.

1. What a Large Number Is Made Of

Throughput figures for local inference tend to arrive as one large number of tokens per second, often a peak, for several requests served at once. A reader cannot tell how many requests produced it, how long it was held, or what any one request experienced. We wanted our own figure for a current open-weight model, measured in a way a reader could check.

In August we argued that such a figure must be measured as a curve against concurrency, because a single point cannot tell batching from queueing. This paper takes an engine that does batch and asks what the total then means.

A model generates text one token (a word or word fragment) at a time, and an engine can advance many requests in the same pass over the weights, sharing the cost of reading them from memory. That is why one card can produce thousands of tokens per second in total while one request on it gets tens. The total says how much text the hardware can produce; the per-stream rate says how long one reader waits.

We ask how the total and the per-stream rate move as streams are added on one card, where the ceiling is and what sets it, and what the total becomes when the model is replicated on six cards.

2. How We Counted

The model is Qwen3.8-27B, a dense 27-billion-parameter model with hybrid linear attention, in NVIDIA's public 4-bit NVFP4 build (about 22 GB on disk). We served it with vLLM, a public nightly build, one server process per GPU, text only, a 32,768-token context limit and up to 512 concurrent sequences allowed. Every card runs its own complete copy, so cards never wait for one another.

A four-GPU node holds four RTX PRO 6000 Blackwell Max-Q cards of 96 GB at their 325 W maximum. A two-GPU node holds two RTX PRO 6000 Blackwell Workstation Edition cards of 96 GB at 600 W, with a standing graphics-clock ceiling of 2,800 MHz (maximum 3,090 MHz) kept as a mitigation for a bus-drop fault seen on one of them earlier in the year.

The load generator is our own single-threaded asynchronous client. For N streams it keeps N streaming chat requests in flight for the whole window, starting a new, distinct request the moment one finishes. Each request is one user message, thinking disabled, temperature 0, at most 512 output tokens. The 512 prompts are short tasks, each numbered so no two share a prefix; the engine's prefix-cache hit rate was 0.0% throughout. The median prompt was 48 tokens and the median completion 512, the cap.

What is counted is the server's own generation-token counter, a Prometheus metric, sampled every 0.5 seconds. Sustained is the counter's increase over the full-occupancy window divided by its length. Best second is the largest increase over any one-second span inside it. For multi-card runs every client starts on the same absolute schedule and the counters are interpolated onto a common window and summed, so the best second is a simultaneous peak and not a sum of separate peaks. Per-stream rate is sustained divided by streams. GPU power is the card's reported draw, averaged over the window.

3. One Card: The Total Climbs While Each Reader Slows

Table 1 is the ladder on one Workstation Edition card with default settings and plain decoding. Step time is streams divided by sustained rate: the time for one decoding step that advances every stream by one token. TTFT is time to first token.

Streams	Sustained tok/s	Best second	Per stream tok/s	Step time ms	TTFT median ms	GPU power W
1	73.1	74.3	73.1	13.68	23	384.6
4	279.0	283.7	69.8	14.34	44	398.9
8	523.2	535.7	65.4	15.29	55	403.4
16	960.9	992.0	60.1	16.65	114	424.5
32	1,584.0	1,666.2	49.5	20.20	203	430.8

Streams	Sustained tok/s	Best second	Per stream tok/s	Step time ms	TTFT median ms	GPU power W
64	2,288.4	2,433.7	35.8	27.97	554	465.0
128	2,899.0	3,070.6	22.6	44.15	449	489.3
256	3,368.7	3,589.9	13.2	75.99	935	493.7
512 asked	3,274.4	3,424.4	n/a	n/a	n/a	481.1

Table 1. One card, plain decoding, default settings, 40-second windows. At 512 requested streams the engine ran 309 at once and held 203 waiting.

From 1 to 256 streams the total rose 46.1-fold (3,368.7 against 73.1) while each stream fell to 18% of its solo rate (13.2 against 73.1).

At 512 requested streams the key-value cache (the memory holding each stream's context) was reported 100% full, and the total was below the 256 figure. At 256 the cache was 82.8% full and nothing waited.

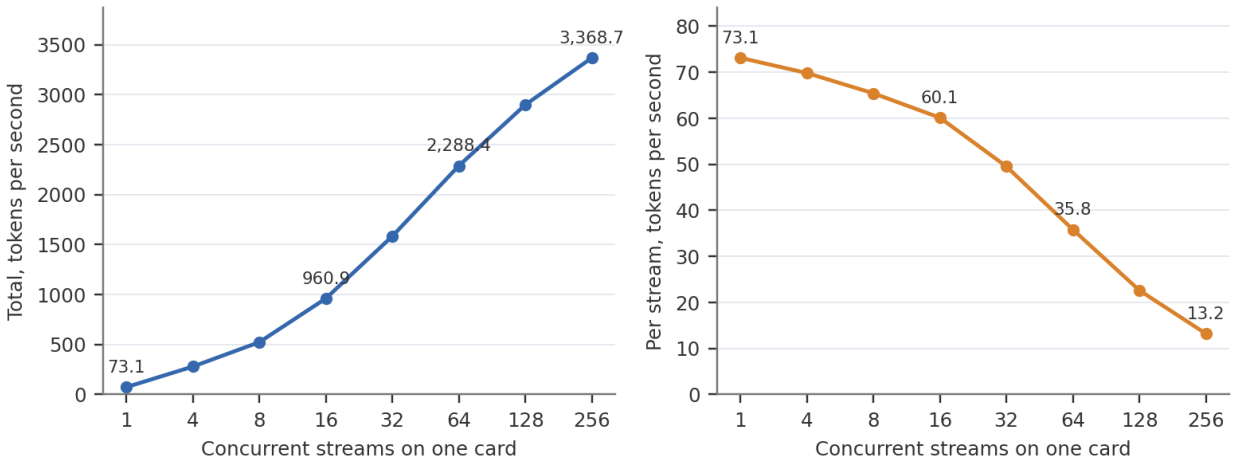


Figure 1. One card, plain decoding. Left, total tokens per second against concurrent streams; right, tokens per second received by each stream. Values are in Table 1.

4. A Decoding Step Costs a Fixed Amount Plus a Little for Each Stream

The step times in Table 1 are close to a straight line. A least-squares fit through the eight levels is 12.94 ms plus 0.2451 ms per stream, predicting 76, 287, 537, 949, 1,540, 2,236, 2,889 and 3,382 tokens per second against measured values from 73.1 to 3,368.7. It describes the curve; the measured single-stream step is 13.68 ms.

The line explains the shape. The fixed 12.94 ms is paid once per step however many streams ride in it, so the second stream is nearly free. The 0.2451 ms is what each rider adds. Total throughput is streams divided by step time, so it climbs steeply while the fixed term dominates and flattens as the per-stream term takes over. The slope alone implies an upper bound of about 4,080 tokens per second per card (1 divided by 0.2451 ms) if streams could be added without limit.

For a reader, every stream added makes every other wait 0.2451 ms longer per token.

5. Past 256 Streams, and Settings That Did Nothing

We expected the total to keep rising slowly, and raised the memory fraction from 0.90 to 0.97 so that the cache would hold more streams. In throughput mode with 25-second windows nothing waited at any level, yet the card delivered 3,454.0 tokens per second at 256 streams, 3,252.0 at 288, 3,312.4 at 320 and 3,369.1 at 336. The line from Section 4 predicts 3,448 at 288 and 3,502 at 320. So the cost per stream rises past 256 on this build. We did not find out why; one candidate is that the engine's kernel configurations are tuned at power-of-two batch sizes, which we did not test. Our setting for this model and build is 256 streams per card.

We also tried engine settings. Throughput mode gave 2.2% at 256 streams (3,443.7 against 3,368.7). Adding four API server processes gave 3,443.9, and a streaming interval of 8 on top gave 3,442.9. The number of API server processes and the streaming interval (how many tokens are batched into each network message) gave nothing, so the HTTP front end was not the limit; the decoding step is. From here, 3,443.7 is the one-card figure in the same engine mode as the multi-card runs.

6. Drafting: Double for One Reader, Loss for Many

The model ships with a draft head (multi-token-prediction): it proposes several tokens ahead and the full model verifies them together, so an easy stretch of text costs one pass instead of several. We enabled it with 6 draft tokens and repeated the ladder. We reported the same exchange in September on two mixture-of-experts seats; this is the curve for a dense model.

Streams	Plain tok/s	Drafting tok/s	Ratio	Drafting TTFT median ms	Running / waiting (drafting)
1	73.1	160.1	2.19	47	1 / 0
4	279.0	586.3	2.10	72	4 / 0
8	523.2	989.5	1.89	84	8 / 0
16	960.9	1,472.8	1.53	113	16 / 0
32	1,584.0	1,914.5	1.21	171	32 / 0
64	2,288.4	2,076.9	0.91	3,141	51 / 13
128	2,899.0	2,063.0	0.71	19,516	51 / 77

Table 2. One card, plain decoding against decoding with the draft head. Ratio is drafting divided by plain.

A lone stream with drafting sustained 160.1 tokens per second, 2.19 times plain; its median decode rate from first to last token was 168.9. The gain shrinks as streams are added: 1.21 at 32 streams, 0.91 at 64.

With drafting the engine ran at most 51 sequences at once at this setting, so from 64 streams requests queued, and median time to first token rose to 3.1 seconds and then 19.5 seconds at 128. Drafting wins up to 32 streams and loses from 64; the crossover lies between.

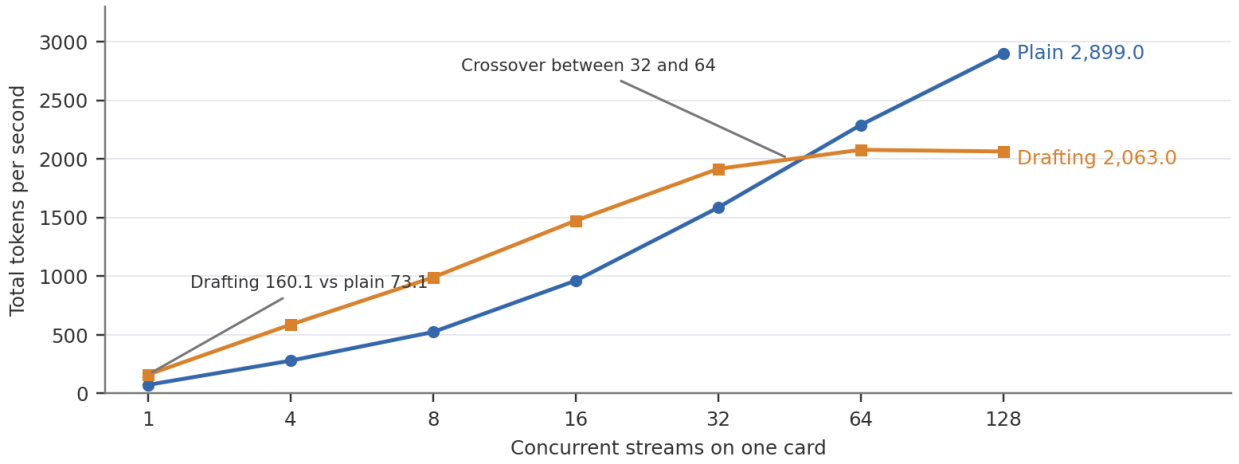


Figure 2. One card, tokens per second against concurrent streams for plain decoding and for decoding with the draft head. The curves cross between 32 and 64 streams. Values are in Table 2.

7. Six Cards: Close to Linear

Replicating the server on more cards is the simplest route to a larger total, because nothing is shared. We ran five cards first (four Max-Q and one Workstation), then six, in throughput mode with synchronised 30-second windows.

Streams (per card)	Sustained tok/s	Best second	Per stream	GPU power W
320 (64)	11,133.5	11,701.8	34.8	1,739
640 (128)	13,972.5	14,717.4	21.8	1,776
1,280 (256)	16,087.0	16,620.9	12.6	1,783
1,500 (300)	14,991.0	15,492.8	10.0	1,775

Table 3. Five cards. The 1,280-stream level was repeated: 16,121.0 (best second 16,863.5) and 16,085.4 (16,623.6).

The curve has one card's shape, including the fall at 300 streams per card.

Run	Clock ceiling on the two Workstation cards	Sustained tok/s	Best second	Per stream	GPU power W
A	in place (2,800 MHz)	19,662.7	20,456.0	12.8	2,250
B1	lifted	19,632.1	20,449.4	12.8	2,284
B2	lifted	19,569.6	20,368.9	12.7	2,302

Table 4. Six cards, 256 streams per card, 1,536 streams in each run, stream interval 4.

The three sustained figures differ by 0.47% end to end. The best second is 4.0% above the sustained figure in run A (20,456.0 against 19,662.7). Per card in run A, the Workstation cards gave 3,440.6 and 3,423.2 tokens per second; the Max-Q mean, 3,199.8, is 93.2% of the Workstation mean, 3,431.9, at 325.0 W against 482.6 and 467.6 W.

Runs B1 and B2 lifted the clock ceiling on the two Workstation cards, with the operator's approval and for those runs only. Their mean graphics clock under load went from 2,688 and 2,676 MHz to about 2,720 MHz and their power rose to between 484.1 and 511.6 W, yet their throughput did not move: 3,440.6 and 3,423.2 before, 3,440.8 and 3,440.4 in B1, 3,440.2 and 3,440.6 in B2. The ceiling was not the limit at this load, and no bus-drop event was logged.

Six cards gave 5.71 times the best one-card figure in the same engine mode (19,662.7 against 3,443.7) and 1.22 times the five-card figure. Tokens per joule of GPU power were 8.74 for six cards and 9.02 for five.

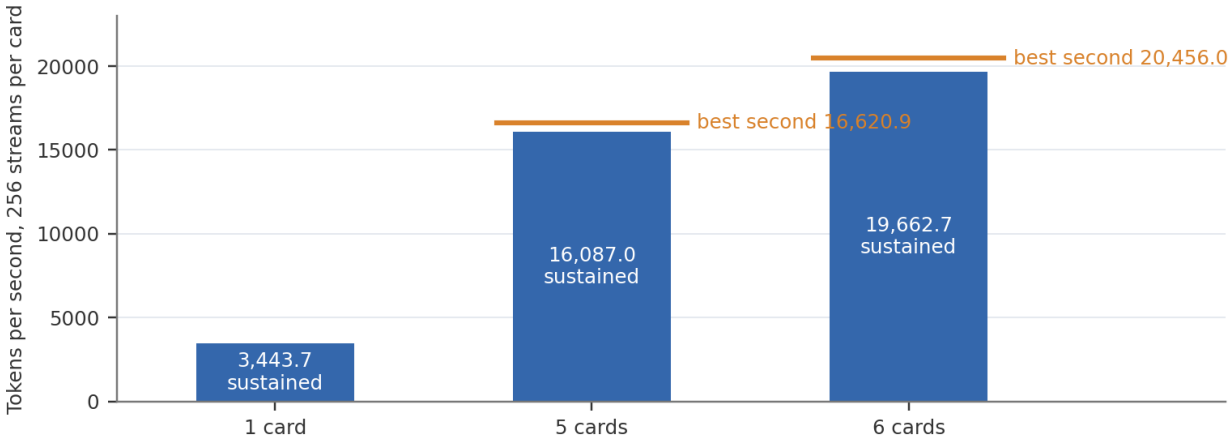


Figure 3. Sustained tokens per second at 256 streams per card against the number of cards, with the best second marked for five and six cards. Values are in Tables 3 and 4 and in Section 5.

8. One Prompt, 1,536 Sections, 1,477 Pages

We also used the aggregate for something concrete. A separate orchestrator turns one prompt, "Write me a complete software engineering handbook. Every topic, with examples.", into one document. It asks the model for the book's parts, then each part's chapters, then each chapter's section titles, one request per node of the outline and all of a level at once, and finally writes every section in its own request, spread evenly over the cards with at most 256 in flight per card.

Each section request carries only the book title, its part, chapter and section title, and an instruction to write about 250 words; no section sees another. Thinking was disabled, temperature 0, and each section capped at 520 output tokens. A browser page merged all streams into one scrolling feed.

Job	Cards	Plan	Sections	Plan time s	Tokens written	Write time s	Average while writing tok/s	Whole job s	Words	Pages at 300 words
Recorded take	6	1 -> 8 -> 96	1,536	6.44	629,303	37.17	16,931.2	43.61	443,160	1,477

Table 5. The recorded fan-out job on six cards. Pages are words divided by 300.

The take made 1,641 requests, 105 of them planning. It had zero request errors and zero empty sections; the mean section was 288.5 words.

The average while writing, 16,931.2, is 86% of the sustained benchmark figure of 19,662.7. The shortfall is the start, where 1,536 prompts must be processed first, and the tail, where sections finish at different times.

The handbook has 8 parts, 96 chapters and 1,536 sections, about 3.1 million characters and 1,402 fenced code blocks. We publish it with this paper, unedited, as a PDF. Typeset at A5 it runs to 1,489 pages; the 1,477 in the title is the word count divided by 300. The recording of the take is published with it.

It is not a good book, and the reason is how it was made. 46 section titles occur more than once. 113 of its heading-delimited blocks (7%) end without closing punctuation, which is what stopping at the 520-token limit looks like. Sections were written without sight of one another, so repetition across chapters is expected and consistency accidental. Nothing was checked against anything, including the code in its 1,402 blocks. The handbook shows the rate at which text can be produced when work divides into independent requests, and says nothing about the quality of the text.

9. Two Measurement Faults We Made and Caught

The first was an open-file limit. The first six-card take ran the orchestrator with the default limit of 1,024 open files, while 1,536 sockets were needed. 520 sections failed all three connection attempts (1,560 errors) and came back empty, yet the job "finished", reporting 423,225 tokens in 28.83 seconds (14,678.9 tokens per second) and 988 pages. We discarded it, raised the limit and re-ran; the recorded take is the re-run.

The second was a peak timed in the browser. The feed page computed its own "best single second" from the arrival times of chunks and showed 22,352 tokens per second for the recorded take. The highest best second measured from the servers' own counters was 20,456.0, so the page was 9% high. Delivery to a browser is bursty: a stall followed by a catch-up packs more than a second of tokens into a one-second span. We removed that line from the recording and the page, and quote only server-counted peaks. The live rate shown in the recording is timed the same way: while every section was in flight it read between 18,539 and 21,828, around a server-counted rate near 19,700, so it is an indicator and not a measurement.

10. What These Numbers Will Not Carry

The evidence is one day, one model build and one engine build. The six-card sustained figure is three runs, with a spread of 0.47%; the five-card figure is three runs at 1,280 streams and one at the other levels; each ladder level is one 40-second window and each tuning arm one 25-second window. We give no intervals.

Short prompts (median 48 tokens) and capped 512-token answers are the favourable case for throughput; longer prompts were not measured. With 512-token answers at 256 streams a request lasts about 39 seconds, so the multi-card windows contain almost no request turnover. The sustained figure is a full-occupancy decode rate, not a long-run service rate with arrivals and departures. No output quality was scored, for the benchmark or the handbook. Power is GPU power only, so tokens per joule exclude the CPU, memory, fans and conversion losses. The 6.8% gap between Max-Q and Workstation cards mixes the power limit with the host. The clock-ceiling comparison is one run with the ceiling and two without, minutes apart. The client addressed each server directly; whether six cards behind one endpoint would deliver the same total is untested.

What survives is the shape of the ladder, the 256-stream ceiling on this build, the drafting crossover between 32 and 64 streams, near-linear scale-out across cards, and a best second about 4% above sustained. No absolute figure survives as a property of the model. Still unknown are why the per-stream cost rises past 256 streams and whether FP8 or BF16 weights would give a higher total on these cards.

11. What We Do Now

We quote a throughput figure as three numbers with the stream count: best second, sustained, and per stream. Peaks come from server counters only. We run 256 streams per card for this model and build. Drafting stays on for endpoints that serve a few users and off for batch work. The load generator and orchestrator now raise their own open-file limit and report empty responses as failures.

The measurement had a cost: the replicas displaced our resident model on the four-GPU node for 36 minutes in the morning and again for the six-card run, which also stopped several services on the two-GPU node for about 12 minutes. During the five-card run the four-GPU node's CPU reached 94.6 C, against a 94 C critical threshold, with frequency boost on and six stale processes from unrelated work each holding a core; with those gone the six-card run peaked at 77.2 C.

12. Methods and Artefacts

Engine: vLLM, public nightly build 0.33.0rc1.dev320 in its public container image, one server per GPU. Model: nvidia/Qwen3.8-27B-NVFP4. Options were text only, 32,768 context, up to 512 sequences and memory fraction 0.90 (0.97 where stated). Throughput performance mode was used for the tuning arms and the multi-card runs; the streaming interval was 4 tokens for the six-card runs and the fan-out jobs. For the measurement only, the two-GPU node's CPU governor was set to performance and the four-GPU node's CPU frequency boost enabled.

Our load generator, synchroniser, orchestrator and feed page are single-file programs written for this work. Statistics are repeats and one least-squares line. The handbook (unedited, as a PDF) and the recording of the take are published with this paper; the handbook's defects are given in Section 8. Retained are the per-level result records with raw counter samples, the engine logs, the orchestrator's statistics, the recorded frames and the handbook exactly as generated.

PureTensor operates a sovereign AI infrastructure fleet (on-premises GPU compute, storage, and serving) run day-to-day by autonomous agents under human direction. This paper is PT-R-2026-073. All measurements were taken on 13 November 2026, between 08:04 and 11:12 UTC; the raw artefacts are retained.