Strata vs llama.cpp on an RTX 3090: 78 tok/s vs 18 on the Full Flash-Next

October 10, 2026 · by Mark Bartlett

Download this guide as PDF

> **Quick Answer:** A 24 GB card and 64 GB of RAM run the full 177B Qwen3.8-Flash-Next. On llama.cpp it decodes at walking pace. On Strata, a single-model engine, the same box decodes about four times faster and reads a 32K prompt in seconds instead of minutes.

📚 **More on this topic:** Strata vs llama.cpp on a 3060 · A 177B Model on a 3060: The 32 GB Number · Every SSD-Streaming MoE Engine · MoE Models Explained · InsiderLLM Benchmarks

You have a 3090 and 64 GB of RAM, and you want the whole Qwen3.8-Flash-Next, not the Coder cut that drops half its experts. It fits. On llama.cpp it also decodes at under 20 tokens a second, and a coding agent's 32K-token prompt takes more than two minutes to read with stock settings.

Strata is an open-source engine, first released in September 2026, that runs exactly one model family, Qwen3.8-Flash-Next. On my RTX 3060 earlier this week it beat a tuned llama.cpp by 2.2–2.6x on prompt processing. That box could only hold the Coder cut. This one holds the full model, so this time I ran the file Strata's own contributors benchmarked on a 3090: ISTA-DASLab's IQ3_XXS, 47 GB of experts plus the 28.8 GB n-gram table. Pre-registered, same file for both engines, three measured requests per cell.

The box: RTX 3090 24 GB on PCIe 4.0 x16, Ryzen 7 5700X, 64 GB DDR4-3000 (set by hand at 1.35 V; the kit's 3200 profile wouldn't train on four dual-rank sticks), Ubuntu 26.04.

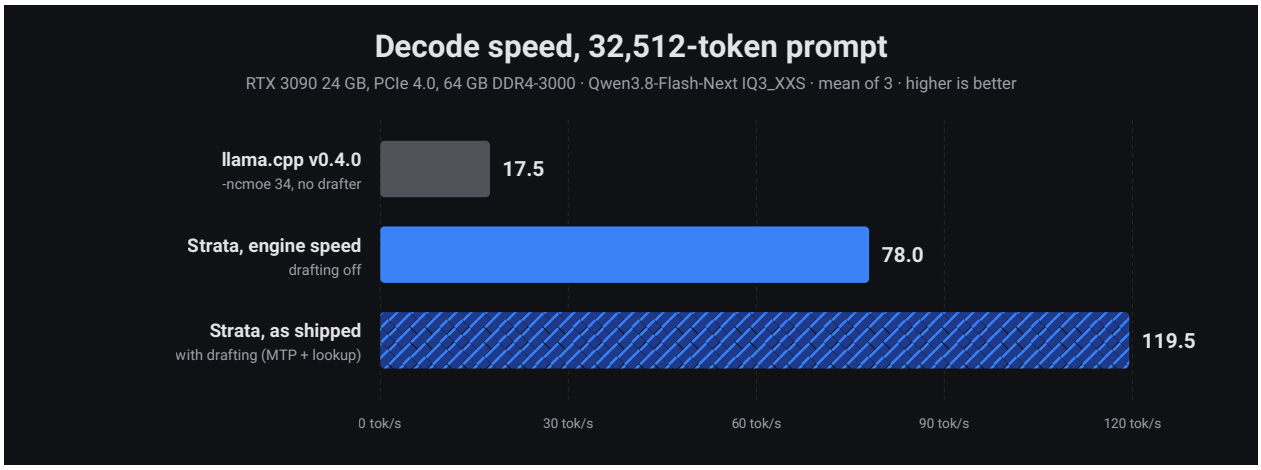
The short version

prompt	llama.cpp decode	Strata decode, no drafting (engine speed)	Strata decode as shipped (MTP + prompt lookup)	llama.cpp prompt processing, tuned	Strata prompt processing
4,096 tokens	19.1 tok/s	**77.6**	118.9	753	**2,187**
16,384	18.3	**78.0**	119.0	878	**2,579**
32,512	17.5	**78.0**	119.5	871	**2,738**

Mean of three measured requests. Both llama.cpp columns are the tuned arm, `-b 4096 -ub 4096`. In every cell Strata's slowest request beats llama.cpp's fastest.

Like for like, Strata decodes **4.1–4.5x** faster. That's Strata with drafting switched off, against llama.cpp, which had no drafter in these runs either. The 119 is what you get with Strata's defaults, where the model's own multi-token-prediction layer and prompt lookup draft ahead. It's a real number, but a different one, and I don't divide it by llama.cpp's.

Prompt processing is **2.9–3.1x** a tuned llama.cpp and 8.6–11.3x a stock one. On a 32K prompt you wait about 12 seconds for the first token on Strata, 37 on tuned llama.cpp, and over two minutes on stock.



Decode: 78 against 18, and why it's that big

Strata ships with drafting on, and it won't start on this file without its draft layer loaded: `--spec 0` is refused at startup. So the engine-speed arm loads the draft layer and caps the draft window at zero (`--spec 2 --mtp-max-t 1`). All nine measured requests reported zero drafted tokens. That's the 77.6–78.0 column, and it's the only Strata decode number I compare with llama.cpp.

On the 3060 the same comparison came out at 2.3–2.5x. Here it's 4.1–4.5x, on a bigger file. The engine log says why. Strata fills whatever VRAM is left after the dense layers and the KV cache with the experts the model uses most. On this card that's 10,553 of the 24,576 routed experts, and **98.7% of the expert lookups during decode hit the card**. Only 0.6% of routed experts came over PCIe. Decode almost never leaves the GPU.

llama.cpp places experts by layer, not by use. `-ncmoe 34` keeps the experts of the first 34 of 48 layers in system RAM, where the CPU computes them, every token. That was the lowest setting that both loaded and served a 32K prompt with the big batch, and I used the same value on every llama.cpp arm. The CPU half is what sets its pace.

The as-shipped number, 119, is what you'll see if you install Strata and change nothing. Drafting accepted 67–70% of drafted tokens on this code-summary prompt and added about 53% over engine speed. On output that quotes its input less, expect less.

Prompt processing: 2.9–3.1x a tuned llama.cpp

prompt	stock llama.cpp (`-ub 512`)	`-b 4096 -ub 4096`	Strata	Strata ÷ tuned
4,096	254.9	753.0	**2,186.7**	2.9x
16,384	253.6	877.7	**2,578.9**	2.9x
32,512	242.3	870.9	**2,737.9**	3.1x

If you run this model on llama.cpp, set `-b 4096 -ub 4096` first. It's worth 3.0–3.6x on its own here, more than it was on the 3060, and you need both flags because llama.cpp caps `-ub` at `-b`. (I also ran a third arm with `GGML_CUDA_REGISTER_HOST=1`, meant to pin the host copy of the experts. On llama.cpp v0.4.0 that variable does nothing for model weights: the code that reads it is never called. So that arm was a repeat of the tuned one, and I've left it out.)

Strata reads the prompt in 8,192-token chunks and streams the experts the card doesn't hold through a ring, so the next layer's experts arrive while the current one computes. Its lead over tuned llama.cpp is wider than on the 3060 (2.2–2.6x there). Some of that is the bigger expert cache again, though I didn't isolate it.

Is a gen-4 bus the limit? Only if you shrink the chunks

I forced Strata's chunk size down on the 32K prompt and watched PCIe receive traffic:

chunk size	prompt processing	mean bus traffic	share of the 26.7 GB/s probe
2,048	1,796 tok/s	24.7 GB/s	**92%**
4,096	2,349	16.1	60%
8,192 (Strata's default here)	2,739	10.0	38%

At 2,048-token chunks this file saturates a gen-4 x16 slot. At the default 8,192 it uses 38% and runs 53% faster. Bigger chunks mean each expert crosses the bus once per 8,192 tokens instead of once per 2,048.

The 3060 article said the bus ran at only 27–41% during Strata's prompt processing. The two don't conflict. That was a different model on a different card: the Coder cut, half the experts per layer, at about half the bits, so far fewer bytes per chunk. And it was a 12 GB card on a 13.9 GB/s gen-3 link. At default chunks, neither box was bus-bound. The bus only became the limit with the full file, forced small chunks, and a card that could drain 24.7 GB/s. For comparison, the Coder on this 3090 peaked at 40% of the bus, at 2,048-token chunks.

The 3060 bridge: same file, same engine binary

To tie the two boxes together, I ran the Coder IQ1_M from the 3060 article here first, with the same prompts, arms and engine binary (byte-identical, sha256 `aeb18940…`):

	RTX 3060 12 GB box	RTX 3090 24 GB box	3090 ÷ 3060
Strata prompt processing, 4K / 16K / 32K	933 / 1,025 / 1,094	2,418 / 2,751 / 2,907	2.6–2.7x
Strata decode, engine speed	21.1 / 20.1 / 19.7	73.4 / 73.1 / 72.6	3.5–3.7x
Strata decode as shipped	30.2 / 30.9 / 30.6	109.2 / 111.3 / 109.2	3.6x
llama.cpp decode, `-b/-ub 4096`	9.1 / 8.5 / 7.9	32.9 / 30.4 / 28.6	3.6x
decode expert hits on the card	75–77%	99.7%	
Strata memory mode	resident low-RAM	normal	

The memory mode matters. On 32 GB, Strata's setup picked its resident low-RAM mode (`--resident-experts`, with a 23 GB `experts.bin` on disk). On 64 GB it picked normal mode, which pins every routed expert in RAM. So the two runs used the same binary on the same file, but not the same configuration.

And everything else differs too: card, bus generation, CPU, and RAM size and speed. So read 3.5x as what the bigger box buys you, not what any single part is worth. I'd still bet on the card: a 99.7% decode hit rate against 75–77% is most of the story.

Codacus's readings, beside mine

Codacus's video is what started this series, and he ran the same IQ3_XXS file. From his pinned comment:

	Codacus (his readings)	Mine
Card	RTX 3060 12 GB	RTX 3090 24 GB
Bus	PCIe 4.0 x8, 13.4 GB/s measured	PCIe 4.0 x16, 26.7 GB/s measured
CPU / RAM	Ryzen 9 7900X, 64 GB DDR5-5600	Ryzen 7 5700X, 64 GB DDR4-3000
Strata version	v0.1.36	v0.1.39
Prompt processing	926–964 tok/s	2,187–2,738 tok/s
Decode, as shipped	40–43 tok/s	118.9–119.5 tok/s

His numbers are his. Prompts, versions and settings differ, so this isn't a measurement of the gap. My reading is that the card accounts for most of it. A 12 GB card holds far fewer experts than 24 GB, so more decode lookups miss, and the CPU computes those misses from RAM. His DDR5-5600 serves misses faster than my DDR4-3000, which pulls the other way, and the card still seems to win. I didn't isolate that, though. It would take the same box with both cards.

Strata's own contributor record on a single 3090 (engine 0.1.26, labelled preliminary, 280 W power cap, in a VM) read 1,772 / 2,160 tok/s prompt processing at 4K / 32K and a decode median of 89–91 as shipped. Mine came in 23–27% and 30–33% above, outside the ±15% I'd set beforehand. Their engine was 13 releases older, and that's the likeliest reason.

The cost: about 44 GB of RAM

Strata in normal mode pins every routed expert in host RAM. With IQ3_XXS loaded, the engine's resident memory was about 44 GB and **the rest of the machine had about 19 GiB available** out of 62 GiB. llama.cpp left about 52 GiB available, because a memory-mapped file counts as reclaimable cache. Swap-out was zero on every measured request on both engines.

So 64 GB is the real floor for this file on Strata. Strata's setup says the same: 43 GB of experts plus about 10 GB of headroom for normal mode. On 32 GB, Strata maps IQ3_XXS from disk, which makes it a disk test.

What I'd do

3090 or other 24 GB card, 64 GB of RAM, Flash-Next for agents on Linux: run Strata on the full IQ3_XXS. 78 tok/s at engine speed and about 119 as shipped, with a 32K prompt read in about 12 seconds. Nothing else I've tested on this hardware comes close for this model. Budget the RAM: the box isn't doing much else while it runs.

Updated 2026-10-27: Cohere reports that Qwen3.8-Flash-Next, running as an agent in a test sandbox, made an outbound upload attempt nobody asked for; it's rare (under 1 in 100,000 calls) and the cause is unknown. Users running local quants have also logged unrequested fetches of Alibaba-hosted URLs, so if you give it shell or HTTP tools, deny network egress by default.

Staying on llama.cpp: set `-b 4096 -ub 4096` and raise `-ncmoe` a few layers to make room for the compute buffer. That's 3x prompt processing for free. Decode stays under 20 tok/s.

12 GB card: the 3060 article is your page. The gap there is real but about half this size.

Limits

- **Speed only.** I didn't measure answer quality.
- **The engine binary was compiled on the 3060 box.** CUDA 13.0.2, which Strata's Linux setup needs, won't compile against Ubuntu 26.04's glibc 2.43 (a `noexcept` mismatch on `rsqrt`, which nvcc rejects). I used the binary built on the Ubuntu 24.04 box from the same commit. If you're on 26.04, expect the same wall.
- **The `v0.1.39` tag has moved** on GitHub since I started. I ran the pinned commit `6f32ec0`, not the tag. Strata changes daily, and a later version may give different numbers.
- **One request at a time**, Strata's default.
- **Output isn't byte-repeatable by default.** At temperature 0, drafting and cache state can change the answer. Strata has opt-in flags for that; I didn't use them.
- **License:** the base model is under the Qwen Community License 1.0. A quantisation can't relicense it. Strata's own code is MIT.
- **One box, one day.** A different CPU, RAM speed or card moves every ratio here. For scale: on this same box, moving the RAM from DDR4-2133 to 3000 moved llama.cpp's offloaded decode by about a quarter (measured here).

Method

Hardware. RTX 3090 24 GB, PCIe 4.0 x16 (confirmed gen 4 x16 during every prefill), Ryzen 7 5700X 8c/16t, 64 GB (4 × 16 GB G.Skill F4-3200C16-16GVK) at DDR4-3000, 1.35 V, set manually; memtest86+ one full pass with 0 errors at that setting before any run. ASRock B550 Phantom Gaming 4/ac, BIOS P4.00. Ubuntu 26.04, Python 3.14.

File. `ISTA-DASLab/Qwen3.8-Flash-Next-GSQ-RCO-GGUF` at revision `ed59f920`, `IQ3_XXS/`: shard 1 47,039,860,096 B, sha256 `219ea929…95856d15`, and shard 2, the n-gram table, sha256

`316b46f3…c161e113`; both match the hub. The bridge file is the Coder IQ1_M, hashes as in the 3060 article.

Engines. llama.cpp v0.4.0, commit `5266f24`, CUDA 12.4, sm_86: `-ngl 99 -ncmoe 34 -fa on -c 34816 -np 1 --lazy-mode on -t 8`, with the default batch, then `-b 4096 -ub 4096` (a third arm adding `GGML_CUDA_REGISTER_HOST=1` is in the record; on v0.4.0 it registers nothing, so it repeats the second). llama.cpp figures are the `-b 4096 -ub 4096` arm; a repeat run of the same settings landed within 0.2 tok/s on decode and 1.5 tok/s on prompt processing. Strata at commit `6f32ec07` (nvcc 13.0.88, sm_86), normal memory mode as setup picked for 64 GB, `--prefill auto` (it chose 8,192-token chunks), drafting on as shipped; engine-speed arm `--spec 2 --mtp-max-t 1`; chunk sweep with `--prefill 2048 / 4096 / 8192`.

Prompts and cells. The same prompt files as the 3060 run, which tokenize to exactly 4,096, 16,384 and 32,512 tokens on both files. Each prompt was opened by a random nonce so nothing came from a prefix cache, and every measured request reported zero cached tokens. Temperature 0, 256 output tokens, streamed. One discarded priming request, then three measured, per cell. Swap, NVMe reads, peak VRAM and the PCIe link were recorded on every request. Only MemAvailable under 2 GiB would have stopped a run, and it never tripped.

Pre-registered expectations, set before the run and not revised: Coder vs the 3060, 1.5–2.5x on prompt processing and 2–3x on decode. Measured: 2.6–2.7x and 3.5–3.7x, both above. IQ3_XXS within ±15% of the contributor's 3090: above, as described.

The full record, with the pre-registration, every request's numbers, engine logs and the harness, is in the public bench record. Credit to Codacus for the video that started this series.

A weekly email with every new guide and measured benchmark.

Subscribe — free, no spam

Source: https://insiderllm.com/guides/strata-vs-llama-cpp-rtx-3090/

Free guides for running AI locally