A 177B Model on a 3060: The 32 GB Number Nobody Measured

September 8, 2026 · Updated: October 5, 2026 · by Mark Bartlett

Download this guide as PDF

> **Quick Answer:** Stock, my RTX 3090 ran Qwen3.8-Flash-Next no faster than a 3060 with the same RAM, because nothing was on the card. One llama.cpp flag bought 34 to 40 percent. The 32 GB machine the video's container cap stood in for does a third of that, streaming 25 MiB per token off the SSD. Here is where your box lands.

📚 **More on this topic:** Every SSD-Streaming MoE Engine · Qwen 3.6-35B MoE on One Card · What Can You Run on 12GB VRAM · InsiderLLM Benchmarks

You have a 12 GB or a 24 GB card, 32 or 64 GB of RAM, and a video telling you that a 177-billion-parameter open-weight model runs on a used 3060. Qwen3.8-Flash-Next — Qwen's open-weight preview of the Qwen 4 architecture, 125B of experts plus a 51B n-gram lookup table, 6B active per token, 512 experts per layer — is the model, and Codacus's video is the claim under test. His 3060 sits in a Ryzen 5 5600X box with 64 GB of DDR4, it decoded at 22 to 24 tok/s and prefilled at about 115, and when he capped a container at 32 GB it still showed 22. Those figures are from a video, flags and RAM speed not on record. I ran the same quant on an RTX 3090 with 62 GB and on an RTX 3060 with 32 GB, both on stock llama.cpp v0.4.0, one day per rig, and the 32 GB number is the one that matters — it is the machine most people asking actually own.

Should you run it

24 GB card and 64 GB of RAM: yes. Measured. With `-ncmoe 29` the 3090 decodes at **22.6 tok/s** at zero depth and **24.5** with 4,096 tokens of context in place, and a 2,343-token prompt prefills at 137 tok/s. That is the top of the video's range, on the more expensive card, after one flag — stock, the same card does 16.9.

12 GB card and 64 GB of RAM: yes, at something between 11 and 17. Inferred, not measured. I did not have a 3060 with 64 GB on the bench. The 17 is the 3090's stock number with the whole expert set cached in RAM, and on this model a 3060 with the set cached has no reason to be slower at stock, because stock puts nothing on either card. The 11 is what the 3060 did with seven expert layers on the card and the set not cached. A 64 GB 3060 lands between those, and closer to the top with `-ncmoe 41`. The video's 22 to 24 on a 64 GB 3060 sits above that range;

his own description calls the 24.4 “about 50 percent over stock on the same card,” which puts his stock near 16, inside it.

12 GB card and 32 GB of RAM: it runs, at 8.7 to 11.5 tok/s, and that is the honest number. Measured. Stock, the 3060 box decodes at **8.7** with a 2,343-token prompt prefilling at 29 to 41 tok/s, and every generated token pulls 25 MiB off the NVMe. Seven layers on the card takes decode to 11.1 to 11.5 and settled prefill to 70. It is usable for chat and slow for anything that feeds it long context. I could not find this measured anywhere, and the video’s 32 GB container is not this machine; the section below explains why.

The one flag: -ncmoe

Stock llama.cpp with `--cpu-moe` puts the dense layers, the shared experts and the output head on the card and every routed expert in system RAM. On the 3090 that is **5,368 MiB of 24,576**. Nineteen gigabytes of the most expensive memory in the box sat empty while DDR4 fed a 45 GiB expert set to the GPU — one token at a time. That is why a stock 3090 does no better than a stock 3060 on this model: the card is not doing the work.

`-ncmoe N` keeps the experts of the first N layers on the CPU and lets the rest onto the card. The model has 48 layers, so stock is `-ncmoe 48` and every step down moves one layer’s experts, about 960 MiB at this quant, onto the GPU. I stepped 48, 40, 34, 29 and 26 on the 3090, three llama-bench invocations each, tg128 at depth 0 and depth 4,096.

-ncmoe	expert layers on GPU	peak VRAM	tg128 d=0	tg128 d=4096	gain d=0	per layer
48	0	5,368 MiB	16.85 [16.84..16.85]	17.54 [17.50..17.57]	—	—
40	8	13,068	17.84 [17.54..18.18]	19.78 [19.53..19.96]	+0.99 (+6%)	+0.12
34	14	18,840	20.01 [19.65..20.55]	22.05 [21.89..22.14]	+3.16 (+19%)	+0.23
29	19	23,652	22.60 [22.04..23.29]	24.48 [24.32..24.71]	+5.75 (+34%)	+0.30
26	22	OOM				

The stock row is reps 2 and 3; rep 1 ran on a cold page cache and is in the method section. At 26 the loader asks CUDA for a 24,991 MiB buffer and cudaMalloc refuses. **Twenty-nine is the last value that fits, at 23,652 MiB, and it is the setting to use on a 24 GB card with 64 GB of RAM.**

The per-layer gain is not flat — it rises from 0.12 tok/s per layer over the first eight layers to 0.30 over the last five, and the reason is arithmetic rather than magic: per-token time falls by a roughly constant 0.8 ms for every layer that leaves the CPU, and a constant saving in milliseconds is a growing gain in tokens per second. The practical rule is the simple one. **Fill the card and stop.** There is no point below the OOM line where the curve flattens.

At `-ncmoe 29` the server's 2,343-token prompt prefilled at 137 tok/s and decoded at 20.8, reading 0.33 MiB per token from disk. Decode from the server sits a couple of tokens under llama-bench on both rigs — the HTTP path and sampling, not the model.

RAM is the purchase

The 3060 box is an i7-7700 with 31 GB of DDR4-2133 and a budget NVMe, which is the machine a used 3060 usually ships in. The expert set at this quant is 45.3 GiB. It does not fit — and the question was what happens when it does not.

-ncmoe	expert layers on GPU	peak VRAM	tg128 d=0	tg128 d=4096	gain d=4096	per layer
48	0	4,929 MiB	8.66 [8.55..8.73]	10.92 [10.89..10.97]	—	—
44	4	8,779	9.22 [9.07..9.34]	11.65 [11.61..11.68]	+0.73 (+7%)	+0.18
41	7	11,665	9.64 [9.51..9.71]	12.22 [12.21..12.23]	+1.30 (+12%)	+0.19
38	10	OOM				

Read the depth-4,096 column as the decode number here. At depth 0 the three repeats inside every invocation climb — 7.2, 9.0, 9.9 tok/s in one stock run — because each one starts on whatever the last one left in a cache that cannot hold the set. At depth 4,096 a prefill runs first and the repeats agree to 0.2 tok/s. **Forty-one is the last value that fits** on 11,909 usable MiB, with 244 MiB to spare.

What the SSD is doing is the finding. I read the disk counters at request start, first token and last token, with `iostat` sampling every second alongside, on three stock server reps back to back:

rep	prefill	decode	disk read during prefill	disk read during decode	per decoded token
1	31.2 tok/s	8.71 tok/s	56.8 GB	3.6 GB at 247 MiB/s	28.2 MiB
2	31.8	8.72	51.7 GB	3.2 GB at 221 MiB/s	25.2 MiB

rep	prefill	decode	disk read during prefill	disk read during decode	per decoded token
3	41.1	8.90	27.1 GB	3.2 GB at 221 MiB/s	24.7 MiB

Update, 2026-10-05: a re-run on the same box read 29.1, 39.0 and 39.3 tok/s, so the stock prefill range is now 29 to 41 tok/s; the low end depends on what the previous request left in the page cache. The 10-05 reference record.

The server's resident set sits at 29 to 30 GB, the page cache at 30.5 GB, and every decoded token pulls 25 to 28 MiB from the NVMe at 220 to 250 MiB/s, sustained, for the whole run. Each 2,343-token prompt reads 27 to 57 GB — more than the entire expert set, once, because the batch touches every expert in every layer and the cache evicts the front while the back streams in. **The page cache does not hold at 32 GB.** I expected rep 3 to settle once the hot experts had been touched a few times. It did not; the set is one and a half times the RAM, and there is no steady state to reach.

Moving seven layers to the card helps twice: the GPU does those layers, and the host-side set shrinks to about 38.7 GiB, so the cache misses less. Per-token disk traffic drops to 4.7 to 6.9 MiB, decode goes to 11.1 to 11.5, and prefill on a settled cache doubles to 70 tok/s. It still reads 5 to 6 GB per prompt.

Now the video. A container capped at 32 GB is not a 32 GB machine. The cap limits what the process can hold resident; the page cache belongs to the host kernel, and if the host has 64 GB the expert set is sitting in it, warm, on the other side of the cap. The container's 22 tok/s is a 64 GB machine's number with a resident-set limit — the cache did the work the cap could not see. A real 32 GB box, where the cache is the RAM, says 8.7. Two confounds keep that from being a refutation: my 3060 box is a four-core i7-7700 on DDR4-2133 against his six-core 5600X on DDR4 of unstated speed, and his flags beyond the thread count are not on record. It is a different measurement of a different thing, and the different thing is what a 32 GB owner will get.

Which leads to a prediction I have not tested and am labelling as such: **a 3060 with 64 GB of RAM beats a 3090 with 32 GB on this model.** The 3090 with 32 GB streams the set from disk exactly as the 3060 did, and nineteen layers on the card do not make up for 25 MiB per token off an SSD. The RAM is the purchase — the card is the flag.

–load-mode none

The loader prints a recommendation when it sees CPU-overridden tensors under mmap: use `--load-mode none` for better performance. I pre-registered the sweep on it — the loader said so — and abandoned it after two reps.

With `none` the 45 GiB expert set goes into the process's own memory instead of the page cache, and the 27 GB n-gram table stays lazily mapped on top of it. On 62 GB that is a 48 GB resident bench process, 12 GB available, 2 GB into swap, and the NVMe reading at 2.0 GB/s during decode. Depth-0 decode was 15.5, a touch under mmap. Depth-4,096 came out bimodal — 10.3, 16.9, 12.5 in one run, 8.4, 8.9, 16.8 in the next — which is what a box paging looks like. **On consumer RAM, mmap is the right mode.** The recommendation is written for a machine with headroom the set does not leave you.

Threads

The video's one takeaway setting is thread count: 12 threads at 13.6 tok/s and swinging, 6 threads, one per real core, at 24.4 and stable. On the stock path, decode only, 3 reps each on the six-core 3090 box: 3 threads 11.7 tok/s, 6 threads 16.1, 12 threads 15.6. Six over twelve is 3 percent, inside the ±1.5 tok/s rep noise. Only 3 threads is clearly worse, 27 percent down. His 24.4 is the figure his fork's README gives for the expert cache with the MTP draft head, so the swing he saw lives on that path, which I did not run. On stock it does not show up. On his path, I did not look.

What's next

His llama.cpp fork carries an expert cache that keeps hot experts in VRAM and streams the rest, plus the MTP draft head, and its README claims 24.4 tok/s on a 3060 with both. That is the next test, against this baseline, on these same two rigs. Not a promise, and not this week.

Limits

One quant, Unsloth's UD-IQ3_XXS, 81.96 GB in three shards. Two rigs, one day each. llama.cpp v0.4.0 only, mmap only after the `none` attempt above, DDR4 on both boxes. The 3090-vs-3060 ratio is directional across these two machines: DDR4-2667 against 2133, six cores against four, 62 GB against 31, and an expert set cached on one and streamed on the other. Change any of those and the ratio moves. The Codacus figures are quoted from a video, not reproduced under his settings.

Method and full results

Model. unsloth/Qwen3.8-Flash-Next-GGUF, `UD-IQ3_XXS`, three shards, 10,946,624 + 49,567,921,344 + 32,382,955,968 bytes. sha256

`268f81fdedf3149a538f252308927a4d5d1f6e062c178568a51e3b519744f8a8`, `cfe600b236b88c7fad1613a5ca5e83b9f2beb63cbd44c32b2be50a44747c695f`, `f1912ba34c79427d2295a58dcb2b732b5931af5bef7a373c60557a57d9ee7250`, verified against the Hugging Face listing on both rigs. Loader metadata: arch `qwen4exp`, 176.94B parameters, 3.0625 bits per weight, 48 layers, 512 experts, 10 used, n_embd 2560; `per_layer_token_embd.weight` 27,466 MiB with lazy read.

Engine. llama.cpp v0.4.0, commit `5266f24da75dc449bd56cbed7addb9c8e4a6a73e`, CUDA 12.8, sm_86, `GGML_NATIVE=OFF`, built on the 3090 box and shipped to the 3060 box as a portable bundle. No fork, no expert cache, no MTP.

Rigs. Miu: RTX 3090 24 GB (24,123 MiB usable), i7-8086K 6c/12t, 62 GB DDR4-2667, Samsung 980 PRO NVMe, Ubuntu 24.04, display manager down, memory-fault soak paused for every run. Rushuna: RTX 3060 12 GB (11,909 MiB usable), i7-7700 4c/8t, 31 GB DDR4-2133, SSSTC CA6 256 GB NVMe, Ubuntu 24.04, driver 580.173.02, no display.

Placement at stock, from a verbose load on the 3090: CUDA0 holds the dense layers, shared experts and output, 3,816 MiB of weights plus 132 MiB KV at 4,096 context and 634 MiB compute; the host holds 144 routed-expert tensors, 48 down projections at 450 MiB and 96 gate/up at 256 to 343 MiB, about 45.3 GiB; the n-gram table is mmapped lazily. `--cpu-moe` on the server and `-ncmoe 48` on llama-bench are the same override on a 48-layer model.

Stock, 3090, 2026-09-07. Server: `llama-server -ngl 99 --cpu-moe -fa on -c 4096 -np 1 --no-warmup`. Load 15.1 s with file pages evicted, 3.9 s warm; RSS 33.2 GB at load, 45.1 GB after three reps; card 5,274 MiB. 2,343-token prompt, `n_predict` 128, temperature 0:

rep	prefill	decode
1, experts cold	65.6 tok/s	16.26 tok/s
2	125.6	15.23
3	132.0	17.49

llama-bench `-ngl 99 -ncmoe 48 -fa 1 -p 512 -n 128 -d 0,4096 -r 3`, 6 threads, 3 invocations: pp512 158.6 / 220.6 / 215.9, tg128 16.68 / 17.02 / 17.63, pp512 at d=4096 159.2 / 208.6 / 204.2, tg128 at d=4096 16.58 / 17.84 / 16.36. Thread sweep `-p 0 -n 128 -t 3,6,12 -r 3`, 3 invocations: 3 threads 11.66 / 11.56 / 11.83; 6 threads 16.00 / 17.65 / 14.77; 12 threads 16.15 / 14.24 / 16.45.

Sweep, both rigs, 2026-09-08. `llama-bench -ngl 99 -ncmoe N -fa 1 -p 0 -n 128 -d 0,4096 -r 3 -t T -lm mmap -o json`, T = 6 on the 3090 box and 4 on the 3060 box, three

invocations per setting, peak VRAM sampled from nvidia-smi every 0.5 s. Mean over the three invocations of llama-bench's own three-repeat average; brackets are min..max over invocations. The 3090's stock row as run, including the cold-cache rep 1 left by the aborted `none` attempt: 15.78 [13.65..16.85] at d=0, 16.84 [15.46..17.57] at d=4096; the table above uses reps 2 and 3. OOM confirmation was a verbose `llama-server` load at the failing value: 3090, `-ncmoe 26`, `allocating 24991.16 MiB on device 0: cudaMalloc failed: out of memory`; 3060, `-ncmoe 38`, `allocating 13441.16 MiB` on 11,909.

Server at the best fit.
`llama-server -ngl 99 -ncmoe N -fa on -c 4096 -t T --load-mode mmap -np 1 --no-warmup`, same prompt, streamed so the prefill/decode boundary is timestamped, `/proc/diskstats` read at request start, first token and end. 3090 at 29, one rep: load 6.5 s, RSS 13.6 GB at load and 28.2 GB after, card 23,550 MiB; prefill 137.2, decode 20.84; 2,490 MiB read during prefill, 42 MiB during decode. 3060 at 41, three reps: load 22.1 s, card 11,561 MiB; prefill 41.9 / 69.5 / 70.5, decode 11.11 / 11.26 / 11.45; disk during prefill 36,091 / 5,852 / 5,082 MiB, during decode 879 / 596 / 674 MiB. 3060 stock, three reps, as tabled above; `iostat -dxt nvme0n1 1` alongside: 251 one-second samples over 1 MB/s read, mean 596 MB/s, peak 1.47 GB/s during prefill.

`--load-mode none`, 3090, two reps at stock. tg128 d=0 15.45 ± 1.13 and 15.57 ± 0.66; d=4096 13.24 ± 3.37 and 11.34 ± 4.74; peak VRAM 5,480 MiB. Sampled during the third rep before it was killed: bench VmRSS 48,355,092 kB; `free -m` available 12,426 of 64,232 MiB, swap used 2,061 MiB; `vmstat` si/so 1,108 / 3,940 KB/s, bi 2,089,432 KB/s; GPU utilisation 1 percent.

Raw llama-bench JSON, server logs, iostat logs, the runner and every number above are in the benchmarks record.

A weekly email with every new guide and measured benchmark.

Subscribe — free, no spam

Source: https://insiderllm.com/guides/qwen3-8-flash-next-rtx-3090-3060-32gb/

Free guides for running AI locally