GTX 1650 vs RTX 3060 on a 35B MoE: What the Card Buys

September 11, 2026 · by Mark Bartlett

Download this guide as PDF

> **Quick Answer:** A $60-class 4 GB GTX 1650 runs Qwen3.6-35B-A3B at 20 tok/s on 32 GB of DDR4, every expert layer but two in system RAM, nothing touching the SSD. Two creators said 17 on a 6 GB 1060. The RTX 3060 in the same slot does 28 at the same setting and 39 tuned. Here is what the card buys.

📚 **More on this topic:** What Can You Run on 4GB VRAM · Best Way to Run Qwen 3.6 35B MoE · A 177B Model on a 3060 · InsiderLLM Benchmarks

You have an old gaming card, the kind that sells used for the price of a takeaway — and 32 GB of RAM in the box it came in. Here is what that gets you on Qwen3.6-35B-A3B: **20 tok/s**, with the dense layers on the card, 38 of the model's 40 expert layers sitting in DDR4-2133, and the SSD reading nothing at all while it generates. That is usable chat. The cost is prefill — 66 tok/s on a 2,343-token prompt, so a long paste takes half a minute before the first word appears.

I measured it on a GTX 1650 4 GB, then pulled the card, put an RTX 3060 12 GB in the same slot, and ran the same file with the same flags on the same build, the same afternoon. The RAM and the CPU did not change. Only the card did — and this is what the card bought.

Should you run it

4 GB card, 32 GB RAM: yes, at 20. Measured. Stock llama.cpp with every routed expert on the host, `-ncmoe 40`, decodes at **20.2 tok/s**. Two expert layers fit on the card beside the dense weights, and they take it to **20.6**. A third layer loads its weights and then fails to allocate a 283 MiB compute buffer — that is the ceiling on 3,714 usable MiB.

6 GB card, 32 GB RAM: about 17, on two videos. Not measured here. Codacus ran this model on a GTX 1060 6 GB with 24 GB of DDR4 and reported 17 tok/s; CloudAICode reported 17 on a 1060 6 GB too. Both are from videos — flags and RAM speed not on record. The 4 GB number above sits over both, and I would not read much into a 4 GB card beating a 6 GB one until someone runs a 1060 through this harness; RAM speed and thread count on those boxes are the likelier explanation.

12 GB card, 32 GB RAM: 28 stock, 39 tuned. Measured today, v0.4.0. The 3060 at `-ncmoe 40` decodes at **28.1 tok/s**, and at `-ncmoe 24`, sixteen expert layers on the card, **38.6**. Prefill is 242 tok/s stock and 365 tuned. That is the number the 35B guide has carried since July — reproduced on a new build.

What the card buys

Start at full offload, where the comparison is cleanest. At `-ncmoe 40` neither card holds an expert. The GPU has the attention weights, the one shared expert that fires on every token, the output head, and the KV cache — about 2.7 GB on either card. Everything else, the 256 routed experts per layer of which eight fire per token beside the one shared expert, is in system RAM, and the RAM is the same DDR4-2133 on both runs.

card, `-ncmoe 40`	peak VRAM	tg128 d=0	tg128 d=4096	server prefill	server decode
GTX 1650 4 GB	2,701 MiB	20.22 [20.19..20.24]	19.76 [19.74..19.77]	65.0 tok/s	19.7
RTX 3060 12 GB	2,765 MiB	28.12 [28.00..28.22]	27.99 [27.93..28.02]	242.1	27.5

The 3060 is **39 percent faster at decode and 3.7x faster at prefill** with nothing on it but the dense parts. I expected less. The story I had told myself, and had half-written into this site, was that at full offload the number is the RAM's number and the card is a passenger. It is not — attention, the shared expert and the head are enough work per token for a five-year-old budget card to be the slower of two, and the 1650's build carries llama.cpp's own warning that TU117 has no tensor cores, so its sm_75 path is not the fast one. Prefill is where that shows — 65 against 242 is the card doing arithmetic on 2,343 tokens at once.

Then let each card take what it can.

card, best fit	`-ncmoe`	expert layers on GPU	peak VRAM	tg128 d=0	tg128 d=4096	server prefill	server decode
GTX 1650 4 GB	38	2	3,665 MiB	20.63 [20.62..20.65]	20.15 [20.12..20.17]	66.0	20.2
RTX 3060 12 GB	24	16	10,259	38.55 [38.10..38.86]	38.39 [38.34..38.44]	365.0	37.7

The gap opens to **87 percent**, and the reason is simply how many layers each card can take — sixteen against two. On the 1650 each of its two layers was worth 0.21 tok/s; on the 3060 the sixteen layers average 0.65 apiece over the whole run, and I did not step the 3060 one layer at a time on this build, so whether its first two layers bought what the 1650's did is not something today's numbers say.

So the rule, and it is the one that decides which part to buy. **RAM sets the floor: with the file in the page cache, a 4 GB card gets you 20. The card sets the ceiling: every expert layer it can hold is a step up from that floor, and a 12 GB card holds eight times as many as a 4 GB one.**

The two video claims, measured

"17 tok/s on a 6 GB 1060." Both cards here beat it, and I am stating that without triumph — a 1650 is a newer card than a 1060 and neither box in those videos is this box. What I can say is that the floor on this model with 32 GB of DDR4-2133 is 20, not 17, and that a 4 GB card is enough to reach it.

"Switching `--load-mode none` is worth three and a half tokens a second." That is CloudAICode's figure for pulling the whole file into process memory instead of mapping it. I ran the pair at the 1650's best fit, `-ncmoe 38`, three llama-bench invocations each and three server reps each:

`-ncmoe 38`	tg128 d=0	tg128 d=4096	server prefill	server decode	load	disk read during decode
mmap	20.63 [20.62..20.65]	20.15 [20.12..20.17]	66.0	20.17	4.7 s	0
none	20.70 [20.68..20.72]	20.22 [20.21..20.24]	70.5	20.25	27.7 s	0

Decode moves **0.3 percent**, consistent in sign across all six invocations and inside the 0.05 tok/s per-sample spread. Prefill moves **6.8 percent** — that is real. Load goes from under five seconds to 28, because `none` reads the 22 GB file into anonymous memory every time the server starts. Three and a half tokens a second did not appear.

Now set that beside Monday. On the same box with an 82 GB Flash-Next file, `none` was not an option at all, and on the 3090 box with 62 GB it swapped, read the NVMe at 2 GB/s during decode, and produced bimodal samples. So the rule has two halves. **When the file fits in RAM, `none` is a prefill setting worth a few percent and a longer load. When it does not, `none` is a**

liability, and mmap is the only mode that runs. The video's own description says the advice reverses when the model outgrows system memory. It does.

Zero disk reads, all day

Every decode today, on both cards and in both load modes, read nothing from the SSD. The disk counters moved by 0 MiB between first token and last on every server rep, and `iostat` across the stock 1650 run averaged 0.9 kB/s with a 76 kB/s peak. Monday, on this same box, the same counters showed **25 to 28 MiB per decoded token at 220 to 250 MiB/s**, sustained. The difference is one number — 22 GB fits in a 32 GB page cache with room to spare, and 82 GB is one and a half times the RAM. Same NVMe, same slot. If your file is smaller than your RAM, the SSD is not in the path and you can stop thinking about it; if it is larger, the SSD is the path.

Build note

Every run today is llama.cpp v0.4.0 at commit `5266f24`, the build this site's dataset pinned on Monday. The 3060 at `-ncmoe 40` came in at 28.12 against the July b10088 row's 28.1, and at `-ncmoe 24` at 38.55 against 38.9 — so the second machine confirms the pin gate within 1 percent.

Limits

One model, one quant, one box, one day. DDR4-2133 dual channel on both runs, which is on the slow side; DDR5 would raise the floor — by how much, I have not measured. The 1650 ran on an sm_75 build, and llama.cpp printed a suggestion at every load to compile with the Pascal code path and force MMQ for Turing cards without tensor cores. I did not, and the 1650's numbers may move if someone does — an open variable, not a footnote. The two 1060 figures are quoted from videos, not reproduced. The 3060 was not stepped one layer at a time on this build, so the per-layer figure for it is an average over sixteen.

Method and full results

Model. `Qwen3.6-35B-A3B-UD-Q4_K_M.gguf` from unsloth/Qwen3.6-35B-A3B-GGUF, 22,134,528,992 bytes, sha256 `ac0e2c1189e055faa36eff361580e79c5bd6f8e76bffb4ce547f167d53e31a61`, verified on this box after the copy and identical to the file behind the July rows and the September 7 pin gate. 40 layers, 256 routed experts per layer, eight routed plus one shared per token.

Engine. llama.cpp v0.4.0, commit `5266f24da75dc449bd56cbed7addb9c8e4a6a73e`, rebuilt with `CMAKE_CUDA_ARCHITECTURES="75;86"` so one bundle serves both cards, CUDA 12.8, shared libraries, Release, `GGML_NATIVE=OFF`. `cuobjdump` shows sm_75 and sm_86 in `libggml-cuda.so`. On the 1650 llama.cpp prints: "suboptimal performance due to a lack of tensor cores … consider compiling with CMAKE_CUDA_ARCHITECTURES=61-virtual;80-virtual and DGGML_CUDA_FORCE_MMQ to force the use of the Pascal code for Turing." Not done.

Box. i7-7700 4c/8t, 31 GB usable DDR4-2133 dual channel, SSSTC CA6 256 GB NVMe, Ubuntu 24.04, driver 580.173.02, headless, PCIe 3.0 x16 in slot 01:00.0 for both cards. GTX 1650: TU117, 3,714 MiB usable. RTX 3060: 11,909 MiB usable. The 3060 went in after a reboot at 12:07 PDT; the file was read once into page cache before its run so no invocation paid the page-in.

Harness. `llama-bench -ngl 99 -ncmoe N -fa 1 -p 0 -n 128 -d 0,4096 -r 3 -t 4 -lm mmap -o json`, three invocations per setting; the row is the mean of the three invocation means, brackets min..max. Peak VRAM from nvidia-smi sampled every 0.5 s. Server: `llama-server -ngl 99 -ncmoe N -fa on -c 4096 -t 4 --load-mode <mode> -np 1 --no-warmup`, 2,343-token prompt, `n_predict` 128, temperature 0, streamed so the first token timestamps the prefill/decode boundary, `/proc/diskstats` read at request start, first token and end, `iostat -dxt nvme0n1 1` alongside.

1650 sweep, mmap. `-ncmoe 40`: 2,701 MiB, d=0 20.22 / 20.24 / 20.19, d=4096 19.74 / 19.76 / 19.77. `-ncmoe 39`: 3,167 MiB, 20.44 / 20.44 / 20.41 and 19.95 / 19.92 / 19.98. `-ncmoe 38`: 3,665 MiB, 20.65 / 20.62 / 20.62 and 20.12 / 20.15 / 20.17. `-ncmoe 37`: weights load, then `allocating 283.00 MiB on device 0: cudaMalloc failed: out of memory` in `graph_reserve: failed to allocate compute buffers`. Server at 40, three reps: prefill 65.2 / 64.9 / 64.9, decode 19.66 / 19.73 / 19.76, RSS 21.4 GB, cached 23.6 GB. Server at 38: prefill 66.4 / 66.0 / 66.0, decode 20.17 / 20.16 / 20.19.

1650, `--load-mode none` at 38. llama-bench d=0 20.72 / 20.69 / 20.68, d=4096 20.24 / 20.21 / 20.21, peak 3,699 MiB. Server: load 27.7 s, RSS 18.6 GB; prefill 70.8 / 70.5 / 70.5; decode 20.24 / 20.26 / 20.25; 1 to 10 MiB read during each prefill, 0 during decode.

3060, mmap. `-ncmoe 40`: 2,765 MiB, d=0 28.15 / 28.22 / 28.00, d=4096 28.00 / 28.02 / 27.93; server prefill 242.1, decode 27.51. `-ncmoe 24`: 10,259 MiB, d=0 38.10 / 38.70 / 38.86, d=4096 38.34 / 38.38 / 38.44; server prefill 365.0, decode 37.74. July b10088 rows for reference, `-p 512 -r 3`, harness default threads: `-ncmoe 40` 28.1 at d=0; `-ncmoe 24` 38.9 at d=0 and 38.5 at d=4096, pp512 413 and 394.

Raw llama-bench JSON, server and iostat logs, the runner and the failure log are in the benchmarks record.
