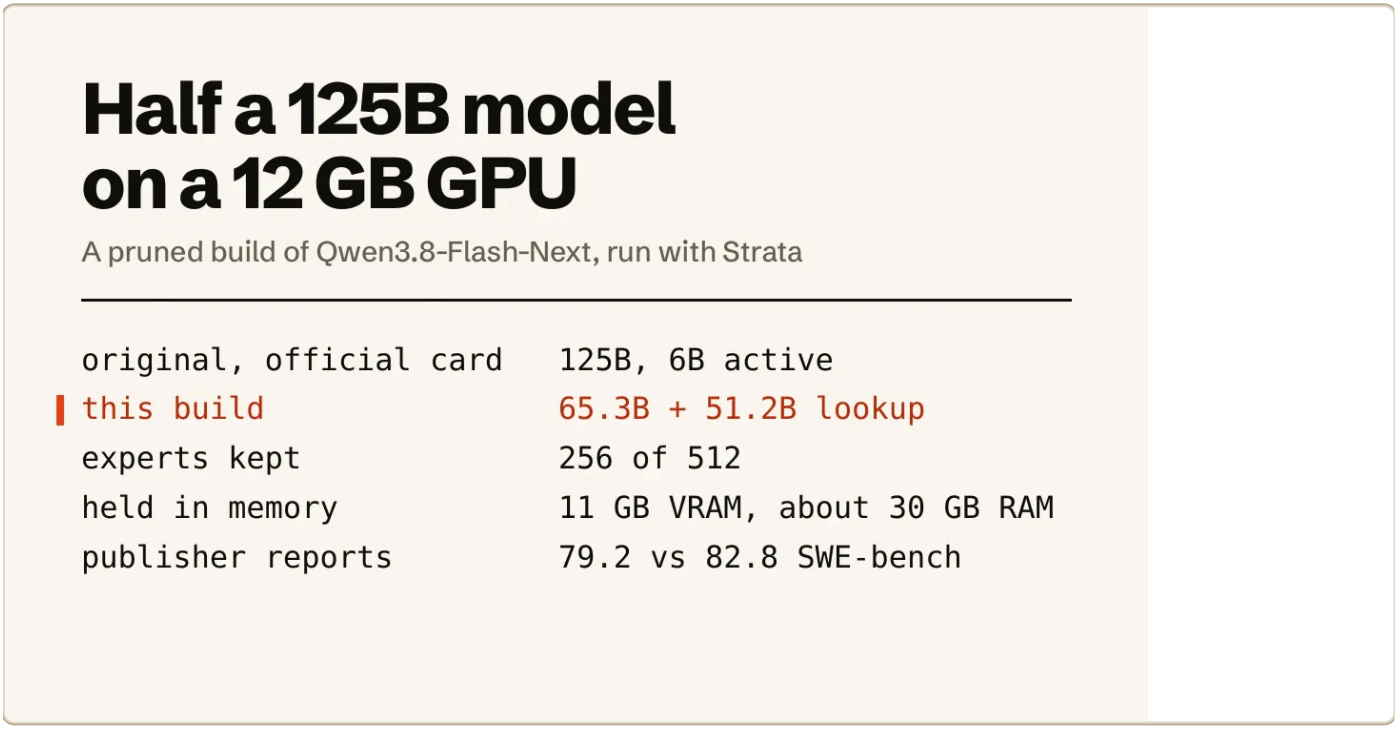
Half a 125-billion-parameter coding model, running on a 12 GB laptop GPU

On 1 November a pruned, quantised coding model ran on my laptop's 12 GB GPU, split across VRAM, RAM and the CPU. What it is, the speeds I can and cannot vouch for, and what the publisher's quality numbers do and do not tell me.



Published
1 November 2026

Reading time
4 minutes

About
Mullet

Tags
AI, Linux

More on: Mullet

On 1 November a coding model ran on my laptop: an RTX 4080 Laptop GPU with 12 GB of VRAM, and 62 GB of RAM. The file is a pruned and quantised build of a 125-billion-parameter coding model: about 65 billion parameters in the part held in memory, plus a 51-billion-parameter lookup table that is not resident, at an average of about 4 bits per weight.

What it is

The original model is Qwen/Qwen3.8-Flash-Next. Its official card says 125B parameters with 6B activated per token, plus a 51B n-gram embedding and 4B for multi-token prediction.

What I run is ISTA-DASLab's "Qwen3.8-Flash-Next GSQ-RCO Coder" GGUF, served by Strata, a third-party local inference engine (the publisher's description). The file is a pruned and quantised build of the original, not a fine-tune. The publisher calls it "a capability-targeted compression" and "an experimental release". Half of the routed experts are removed (256 of 512 in each of 48 layers,

with 10 active per token). A method called RCO chose which ones, by comparing the pruned and unpruned model on calibration data. The card names code, agentic and vision data for that, and says the reduction targets code, agentic tool use, vision and spatial reasoning. The name carries GSQ, a quantisation method, and the card says the remaining weights are quantised to 3.5 bits per weight.

I added up the tensor shapes in the two files:

- Shard 1 is 29.6 GB, has to be in memory, and holds 65.31 billion parameters: the transformer.
- Shard 2 is 28.8 GB, need not be in memory, and holds 51.20 billion parameters in one tensor, an n-gram embedding lookup table.
- Together that is 116.5 billion (arithmetic; Hugging Face lists 117B).

So the transformer is about 65 billion against the original's 125 billion, roughly half (arithmetic: 65.31 divided by 125 is about 0.52). The publisher gives no active-parameter count for the pruned build, so I give none. The lookup table can stay on disk, which is how a 58 GB file runs on a machine with 62 GB of RAM.

The quant label

The file is labelled IQ1_M by its publisher, and it is the only Coder size I have installed in Strata. The stored types are a mix. In the tensor table about 67.6 B parameters are IQ4_NL, 16.8 B are IQ2_S, 14.3 B are IQ3_XXS, 8.4 B are IQ3_S and 2.3 B are Q6_K, plus smaller amounts. The file averages 4.01 bits per weight (arithmetic: 58.4 GB for 116.5 B parameters). The publisher says the retained transformer weights are stored at 3.5 bits and the n-gram table at 4.5. In practice it is mostly 2 to 4.5 bit types.

How it fits

Strata loads the experts into RAM and locks part of them in a cache in VRAM for the GPU. Its own start-up message says about 30 GB goes into RAM. Across the logs I found it reported:

- An expert cache in VRAM of 1,930 to 2,584 experts, taking 3.7 to 4.95 GiB.
- A cache hit rate between about 41% and 82%, depending on the request. Its log also splits out the share of experts read over PCIe, which was between about 0.6% and 23%. I have not worked out where the rest of the misses go.
- About 11 GiB of VRAM in use (nvidia-smi showed 11,364 MiB of 12,282 when I checked) and 33 GiB resident for the engine's process. The 29.6 GB shard must be resident, which fits the RAM figure. The process was also using over 300% CPU when I looked, so the CPU does real work too. I have not broken either figure down.

How fast

I have no trustworthy speed figure for an idle machine. An earlier reading gave 5.4 seconds for a 10,120-token prompt and 55.9 seconds for a 105,069-token prompt, about 1,870 and 1,880 tokens a second (arithmetic), but I did not keep that output. When I repeated the same small script later on 1 November, with another job running on the GPU, a 10,119-token prompt took 25.8 seconds (about 390 tokens a second) and a 105,066-token prompt took 80.7 seconds (about 1,300). So prompt speed depends heavily on what else the machine is doing.

For generation I only have Strata's own log lines from the 32k run: 15 requests at 34 to 67 tokens a second, ten of them between 41 and 55.

Strata warns that the first start can take one to three minutes and that the PC can be slow or stop responding while it loads. With the files already cached, the one restart I timed on 1 November, after killing the server on purpose, had it answering again in 32 seconds, including the time the watchdog took to notice. Strata's built-in tuner said "already fastest" twice.

Strata says its draft layer, a helper that speeds up answers, makes them about twice as fast. I have not done a clean with-and-without comparison, so I can neither confirm nor rule out the 2x.

What I know about quality

These are third-party numbers, the publisher's own. At xhigh reasoning effort it reports SWE-bench Verified 79.20 for this build against 82.80 for the unpruned BF16 model, which it puts at 95.7% retained, and LiveCodeBench v6 87.43 for both, after an update on 17 October 2026. I have not reproduced any of that.

It also says multi-step agentic tasks tolerate the reduction less well than single-problem code generation, and that quality outside code, agentic, vision and spatial tasks may be worse. Mullet's steps are multi-step agentic tasks, which is the case it flags.

My own test is a sample of one: a single step on 1 November, a new module with nine tests, that passed first time with gates G1 to G6 green (the reviewer gate was off). That tells me very little about its quality.

What it costs

The machine, not an API. There is no per-token bill, but there is 58 GB of disk, most of my RAM and GPU while it runs, and a slow first start.

The code is at https://github.com/tomgould/mullet-public (https://github.com/tomgould/mullet-public).