Bonsai 2 vs Qwen3.8-27B on an RTX 3090: Speed Doubled, Scope Broke

September 24, 2026 · by Mark Bartlett

Download this guide as PDF

> **Quick Answer:** A 27B that loads from a 7.2 GB file and decodes 1.8x faster than the Q4 it was compressed from is real, and I measured it. So is the bill: on my 47-item routing split it went from 19 right to 12, and every lost item is the same field. Here is the speed table, the per-field table, and why PrismML's 98% and my 12 of 47 can both be true.

📚 **More on this topic:** Qwen 3.8-27B vs 3.6-27B on a 3090 · What Can You Run on 8GB VRAM · What Can You Run on 12GB VRAM · Jev Mode on a 3090 · InsiderLLM Benchmarks

If you own an 8 GB or 12 GB card, a 27B model has been the thing just out of reach — Q4 of a 27B is 17 GB. You can stream experts over PCIe for a MoE, but a dense 27B does not offload gracefully, and the answer for two years has been "buy a 24 GB card." Bonsai 2 27B is PrismML's ternary quantization of Alibaba's Qwen3.8-27B: every weight stored as −1, 0 or +1 with a small scale per group, so the same 27 billion parameters land in a 7.2 GB file. It came out on September 17, and PrismML's number is that it keeps 98.2% of the full-precision model's benchmark score.

I ran it on my RTX 3090 against Qwen3.8-27B at the Q4 most people actually download, same box, same build, same hour, and then put both through the 47-item routing split this site has been scoring since August. The speed half is as good as the pitch. The quality half is not — and the interesting part is where it isn't.

Should you use it

If your card is 12 GB and you want a 27B for chat: try it. Code is where PrismML's table claims parity, and I have not measured it. It peaked at 8,118 MiB with 8k of context on a 3090, so it fits a 12 GB card with room. On an 8 GB card that number does not fit; it will need a short context, and I have not measured how short. It decodes at 77 tok/s here against 42 for the Q4, and it prefills a hair faster too.

If you are routing on it, or asking it to read whether a message refers back to earlier conversation: measure first. On my split it kept the two fields that pick a tool and a mode and lost the one that picks a scope, badly, in one consistent direction. That is a narrow failure — and exactly the kind a "98%" aggregate hides.

Anything with a 24 GB card and no VRAM pressure: the Q4 is still the safer file. The speed is nice. It is not worth a field of judgement if you do not need the memory.

Speed: 1.8x, and it holds at depth

llama-bench, `-ngl 99 -fa 1 -p 512 -n 128 -d 0,4096,8192 -r 5`, the harness every row on this site uses. Three reps per model, models alternating, a discarded priming run before each measured run so nothing loads cold. Mean of three, min..max in brackets.

	Qwen3.8-27B UD-Q4_K_XL	Bonsai 2 27B PQ2_0	ratio
tg128 d=0	42.13 [42.11..42.15]	77.62 [77.57..77.67]	**1.84x**
tg128 d=4096	41.60 [41.59..41.62]	75.93 [75.90..75.96]	**1.83x**
tg128 d=8192	41.11 [41.11..41.11]	74.38 [74.36..74.40]	**1.81x**
pp512 d=0	1,415 [1,412..1,419]	1,476 [1,474..1,478]	1.04x
pp512 d=4096	1,358 [1,357..1,361]	1,413 [1,412..1,415]	1.04x
pp512 d=8192	1,300 [1,299..1,300]	1,351 [1,350..1,351]	1.04x
peak VRAM	17,666 MiB	**8,118 MiB**	0.46x
board power under load	402 W	400 W	

Decode is bandwidth-bound on a single card, so a file that moves 2.5x fewer bytes per token (17.9 GB against 7.2) should be faster, and it is. But 1.84x is well under 2.5x — the gap is the cost of unpacking trits into something the tensor cores can multiply. Prefill is compute-bound and barely moves. Both sat on the 420 W power cap during prefill — the ternary model runs no cooler, it just finishes sooner. The repeat spreads are 0.01 to 0.10 tok/s on decode, which is why I am comfortable quoting the second decimal.

One footnote on the build, because it matters for anyone comparing to other rows here. Bonsai 2's file format, `PQ2_0`, does not load on mainline llama.cpp; it needs PrismML's fork, which sits on upstream b10709. So both models ran under the fork, not under this site's v0.4.0 pin. I checked the fork against the pin on the same box and the same Qwen3.8 file the day before: decode within 1% at every depth, prefill 2.2 to 2.4% slower. Read the prefill column above as about 2.3% under what the pin would give — the ratio between the two models is unaffected.

What it costs: 12 of 47, and all of it in one field

The task is the intent classifier from my own agent: three fields, four labels each, exact match on all three, scored against the frozen 47-item split the Jev-mode piece and the skill-compilation series before it established. Same system prompt, same request byte for byte, temperature 0, thinking off, both models served by the same fork. Two passes each; both models gave identical answers on all 47 items both times.

	Qwen3.8-27B Q4_K_XL	Bonsai 2 PQ2_0
exact, all three fields	**19 / 47**	**12 / 47**
tool correct	28	28
mode correct	24	**27**
scope correct	28	**18**
valid JSON	47 / 47	47 / 47
mean wall per item	0.74 s	0.53 s

Look at the middle rows before the top one. That is where the story is. On tool, the field that decides whether the agent searches the web, hits the document store, or just answers, the two are tied at 28. On mode, Bonsai is ahead: 27 to 24, right on five items where the Q4 is wrong against two the other way. If the split scored only those two fields, exact match on tool and mode, the ternary model would win — 26 to 23.

Then scope — 28 to 18, and the whole seven-item gap. Bonsai's errors are not scattered. Twenty-three of its wrong scopes are the same call: `facts` where the answer was `session` (12 times) or `all` (11 times). The Q4's scope errors are the family's usual one, `session` read as `all`, nine times. So the ternary model kept the routing judgement and lost the field that depends on noticing whether a message points back at something said earlier — the one field where the model has to remember, not decide. That was already the hardest field on this split for every model that has run it. Bonsai fails it in a new direction, and it fails it consistently.

Item by item: both right on 10, the Q4 alone on 9, Bonsai alone on 2, neither on 26. Those 26 are mostly the same block no model or adapter has moved since August — I have stopped expecting them to. The full per-item table shows every disagreement with the wrong field marked.

For scale: one item on this split is 2.1 points. The Q4 of Qwen3.8 lands at 19 here; the Qwen3.6-27B Q4_K_M the series started on scored 23. The seven-item gap is well outside anything the series has ever called noise.

Two numbers that are both true

PrismML's 98.2% is an average over 14 benchmarks, run with EvalScope and vLLM on an H100, in thinking mode, against Qwen3.8-27B at FP16 — 84.78 against 86.32, per the model card. Math and code sit at parity in their table; knowledge takes the biggest hit.

Mine is one task, 47 items, thinking off, on a consumer card, against the Q4 that consumer card can actually hold. Nothing in my setup can reproduce theirs: FP16 of a 27B is 54 GB and does not fit a 3090, and a routing prompt with thinking off is a different animal from AIME with thinking on. Both numbers can stand — they are answers to different questions. What mine adds is the shape of the loss on a task where the model has to read a conversation rather than solve a problem — a shape a 14-benchmark mean cannot show you, because it averages the field that broke in with the thirteen others that did not.

I said before any download what would count: within one item of the Q4 and at least as fast, confirmed; three or more items below, refuted, whatever the speed. Seven below is refuted. The gate was written down first so I could not move it after.

The bits-per-weight note

PrismML says 1.76 bits per weight and 5.9 GB. llama-bench, loading the file I ran, prints 2.13. Both are right; they describe two files. The 1.76 figure is the `PTQ1_0` packing, 5.95 GB over 26.9 billion parameters. The `PQ2_0` file, which the card recommends for GPUs and which every number above was taken on, is 7.2 GB, and 7.2 GB over the same parameters is 2.14. Two-bit slots cost a quarter more bytes for a layout the CUDA kernels can chew on.

For sizing your card the honest figure is the file you will load: **7.2 GB and 2.14 bpw, 8,118 MiB peak with an 8k sweep** on this build. "A 27B in 5.9 GB" is true of a packing you are told not to use on a GPU.

What would change the verdict

Two things, and I would run either. The same 47 items in thinking mode: PrismML's number is a thinking-mode number, and it is possible the scope field comes back when the model is allowed to reason before it answers. Or a different task. This split is routing on a conversation; a code or math split is where the vendor's table says parity, and a result there would be a different result, not a contradiction of this one. Either would go in the dataset beside these rows.

What would not change it is a faster fork or a bigger card — the speed is not in question.

Method and limits

Box. Tamanna: RTX 3090 24 GB, Ryzen 7 5700X, 32 GB DDR4, PCIe 4.0 x16, headless, open frame, Ubuntu 26.04, driver 580.178.04. The card sat at gen 4 in every under-load sample.

Build. PrismML-Eng/llama.cpp at tag `prism-b10709-9a9394a`, commit `9a9394a`, built for sm_86 with the same cmake line and gcc-13 / CUDA 12.4 toolchain as my v0.4.0 pin. Fork-vs-pin check in the record.

Files. `Ternary-Bonsai-2-27B-PQ2_0.gguf`, 7,206,168,928 bytes, sha256 `3907dc16…`, verified against the hub's LFS object on download. `Qwen3.8-27B-UD-Q4_K_XL.gguf`, 17,923,394,624 bytes, sha256 `bee238bb…`, the file the 3.8-vs-3.6 piece measured, re-verified against that record; Unsloth has since replaced the file under the same name, so a fresh download will not hash to this.

Quality run. `llama-server -ngl 99 -fa on -c 4096 --jinja` from the fork, one model at a time; `POST /v1/chat/completions`, temperature 0, `max_tokens` 100, `enable_thinking: false`; parsed and scored by the ten-seeds scorer unchanged. Both models: 47 of 47 valid JSON, zero thinking leaks, identical output on both passes.

Limits. One card, one build that is not the pin, one 47-item task with thinking off. Bonsai 2's vision tower was not loaded. The `PTQ1_0` file was not run. Qwen3.8-27B at FP16, the claim's real baseline, cannot run on this hardware. Everything I measured is in the repo at docs/bench-results/tamanna-bonsai-2-vs-qwen38-27b-2026-09-24/, and the six speed rows are in the dataset at v1.8.0, tagged with the fork build rather than the pin.