Ornith 1.5 35B vs Qwen 3.6 on RTX 3090: Speed Tested

August 28, 2026 · by Mark Bartlett

Download this guide as PDF

> **Quick Answer:** Ornith 1.5's 35B MoE generates about 9% faster than Qwen 3.6's on one RTX 3090, and loses prompt processing by 3%. Both land within 14 MiB on VRAM. Here's what that 9% is really a gap between, and why the viral 155 tok/s may not be a llama.cpp number.

📚 **More on this topic:** The 35B MoE on a 3090 · Qwen 3.8 27B vs 3.6 on RTX 3090 · MoE Models Explained · Best Local Coding Models · DFlash vs MTP on RTX 3090 · VRAM Requirements

Ornith 1.5 landed on August 19: DeepReinforce, MIT licence, three sizes, and a 35B-A3B in the middle that fits a 24GB card. If you already run Qwen 3.6-35B-A3B, the middle one is the interesting one, because it is the same shape as the model you have: 35B total, 3B active, 256 experts, 8 of them per token.

So the question is a download question. The Q4_K_M is 21.86 GB. Is it worth the bandwidth and the evening?

I benched it against Qwen 3.6-35B-A3B on one RTX 3090, both files in one session, A-B-B-A. Generation, prompt processing and peak VRAM. Here's what came back, what the viral numbers people keep quoting turn out to be attached to, and one place where this run made a finding of my own smaller than I'd written it.

Image: Dot plot comparing Ornith 1.5-35B-A3B and Qwen 3.6-35B-A3B generation speed on an RTX 3090 at three context depths, with slot-to-slot spread drawn to scale

The run

Miu, my workstation: one RTX 3090 24GB, i7-8086K, 62GB DDR4-2667, Xubuntu 24.04, display manager stopped so the card idles at 33 MiB before each load. llama.cpp b10088, commit `67b9b0e`, CUDA sm_86, the build every paired sweep on this site is pinned to. Command was `llama-bench -ngl 99 -fa 1 -p 512 -n 128 -d 0,4096,8192 -r 5`, byte-identical to the string on the Qwen 3.8 page. Order was A-B-B-A across four slots — Ornith, Qwen, Qwen, Ornith — with the page cache dropped between each, so both models sit in the same session and cross-session drift is absent rather than controlled for.

Provenance, briefly, because one half of it was missing. Every slot was bracketed: evict, confirm 0.00% page residency, hash the file against its upstream digest, launch, hash again after. Eight hashes, eight matches. Ornith's digest is bartowski's LFS oid; Qwen's was **not on record** — the 35B rows on this site predate the `model.sha256` field — so I recovered it from the `huggingface_hub` download metadata and verified both files before writing the harness. That is bookkeeping catching up, not a flourish.

Thermally the four slots are symmetric. Start temps 45, 49, 50, 50 °C, peaks 67 to 70, all four pinned at 1950 MHz and 416–418 W against the same software power cap, and each model has one slot on each side of the warm-up ramp.

Generation: Ornith wins by about 9%

Mean of two slots per model, per depth.

Context depth	Ornith 1.5-35B-A3B	Qwen 3.6-35B-A3B	Gap	Gap %
Empty (d=0)	**172.44** tok/s	158.18 tok/s	+14.26	+9.01%
4K (d=4096)	**170.69** tok/s	157.18 tok/s	+13.51	+8.60%
8K (d=8192)	**166.59** tok/s	153.47 tok/s	+13.13	+8.55%

It holds at every depth, and it holds by a wide margin over the run's own noise. The largest gap between a model's two slots anywhere in the sweep is 0.725 tok/s. The between-model gaps are 18 to 37 times that, depending on the depth. This is resolvable in a way most single-digit-percent benchmark claims are not.

Worth noting the control: Qwen 3.6 came back at 158.18 / 157.18 / 153.47 here, against the **157.66 tok/s** this site already publishes for the same file on the same card from July. Same rig, different month, different session, under 1% apart. The incumbent behaved.

Prompt processing goes the other way

Everyone reaches for the decode number. Prefill is the half that decides whether a long paste feels instant or feels like waiting, and here it reverses the result.

Context depth	Ornith 1.5-35B-A3B	Qwen 3.6-35B-A3B	Gap	Gap %
Empty (d=0)	3,602.94 t/s	**3,694.22** t/s	−91.28	−2.53%
4K (d=4096)	3,387.42 t/s	**3,493.14** t/s	−105.72	−3.12%

Context depth	Ornith 1.5-35B-A3B	Qwen 3.6-35B-A3B	Gap	Gap %
8K (d=8192)	3,263.34 t/s	**3,362.46** t/s	−99.13	−3.04%

Qwen is faster at ingesting, by 2.5% to 3.1%, at all three depths. That clears both the slot-to-slot spread and the within-run standard deviation everywhere, so it is a real direction and not a shrug, which is worth saying plainly, because the equivalent Qwen 3.8 run could not resolve prompt processing at all and I published it as unresolvable.

Three percent on prefill against nine on decode is not a hard trade for most people. At 3,300 t/s you are ingesting an 8K paste in about two and a half seconds either way. But if your workload is paste-heavy and reply-short — code review, log triage, summarising — the two numbers point in opposite directions and the decode win is the smaller half of your day.

What that 9% is a gap between

Here's the gotcha, and it is the reason this page does not say "Ornith is 9% faster than Qwen."

The two files were not quantized by the same person to the same recipe, and there is no version of this run where they could have been. Ornith is bartowski's imatrix Q4_K_M, calibrated over 573 chunks. Qwen is Unsloth's Dynamic UD-Q4_K_M, calibrated over 76. Unsloth has not shipped an Ornith 1.5 build — they stopped at 1.0 — and the community reproductions of their method are explicitly not Unsloth releases, so the recipe cannot be held constant in either direction.

And the mismatch has a direction, which is the part that matters:

	Ornith 1.5-35B-A3B	Qwen 3.6-35B-A3B
File on disk	21,864,081,056 bytes	22,134,528,992 bytes
Blocks	41	40
Tensor bytes	21.85 GB	22.12 GB
Parameters	35.51 B	34.66 B
Bits per weight	4.92	5.11
imatrix chunks	573	76

Qwen's file is **1.24% larger while carrying one fewer block and fewer parameters**. Averaged across the weights it is 3.7% denser. Batch-one generation on a 3090 is bandwidth-bound on the weights it reads per token, so more bytes per weight is a plausible partial contributor to Qwen

decoding slower. Not a proven one, and nothing here isolates it. But it pushes the same way as the result, and I would rather hand you that than let a clean 9% stand unqualified.

Read the number as a gap between **these two files**. An unknown share of it is recipe rather than model, and there is no build in existence that would let me split them.

What is controlled is more than usual. llama.cpp loads both under the same `qwen35moe` architecture, with the same 256 experts and 8 active, the same 16 heads over 2 KV heads, the same 262,144 context, the same hybrid attention layout, and the same `general.file_type`. Structurally these two are near-twins. The weights differ, and one carries an extra block.

The noise floor, and a number I have to correct

On the August 14 sweep the same-model repeats came back within 0.00 to 0.02 tok/s of each other, and I have leaned on that. It did not repeat. Here the repeat spreads are 0.365 to 0.725 tok/s — one to two orders of magnitude looser.

That does not put a scratch on the conclusion. A 14 tok/s gap measured against a 0.7 tok/s noise floor is a safer read than a sub-one-percent gap measured against 0.02. But the old line was about a specific run on a specific pair of files, not a standing property of this rig, and repeating it here would be quoting myself out of date.

The second half is more interesting, and it is a finding rather than a caveat. **Ornith is roughly three to four times less repeatable than Qwen, at every depth.**

Depth	Ornith slot-to-slot spread	Qwen slot-to-slot spread	Ratio
d = 0	0.535 tok/s	0.135 tok/s	3.96×
d = 4096	0.365 tok/s	0.125 tok/s	2.92×
d = 8192	0.725 tok/s	0.172 tok/s	4.22×

Be precise about which noise that is. It is between slots: the same file, benched twice, half an hour apart, with an eviction and a re-hash in between. Within a single llama-bench run the two models are close, and at empty context Ornith is actually the quieter of the two. So it is not that Ornith's kernels jitter more; it is that Ornith lands in a slightly different place each time it is loaded and run. Both spreads are small in absolute terms, and neither would change a purchase. But if you are benching Ornith yourself, one run is worth less than one run of Qwen, and you should repeat it.

VRAM: 14 MiB apart, and what that does to a finding of mine

	Peak VRAM	Repeats
Ornith 1.5-35B-A3B	21,662 MiB	A1 and A2 identical
Qwen 3.6-35B-A3B	21,676 MiB	B1 and B2 identical
Difference	**14 MiB**	**0.06%**

Four slots, two values, no variance. Both models leave roughly 2.4 GiB spare on a headless 24GB card with 8K already in the cache. On the VRAM question there is nothing to decide: if one fits, the other fits.

Now the part that costs me something.

Ornith's file carries 41 blocks; Qwen 3.6's carries 40. The extra one is a multi-token-prediction block, `nextn_predict_layers = 1` in the GGUF metadata, absent from Qwen's entirely. That is exactly the structural difference this site measured back on August 14, when Qwen 3.8-27B shipped an MTP block that 3.6-27B did not, and the whole generation-over-generation VRAM delta came out at **254 MiB**. I used that number to correct a circulating claim that an MTP block costs ~2.5 GiB resident, and the correction was right.

What I then had to resist writing was the tidy version: an MTP block costs about 254 MiB. This run says no. Same structural difference, different model pair, and the entire delta — MTP block and every other difference between the two files put together — is 14 MiB, with the block-carrying model on the lower side.

So I opened both files and added the blocks up, which is what I should have done in August instead of reasoning from deltas.

	Ornith 1.5-35B (bartowski)	Qwen 3.8-27B (Unsloth)
MTP block weight tensors	Q4_0, every one	Q5_K, `nextn.eh_proj` at Q6_K
Average over the block	4.52 bits per weight	5.38 bits per weight
MTP block on disk	454.9 MiB	272.6 MiB
A normal block in the same file	459.2 MiB	238.1 MiB

bartowski's card says it outright: "the MTP layers are stored at Q4_0 in the imatrix quants (except for the Q8_0 quant), since imatrix calibration does not exercise them." A block the calibration never touches gets stored cheaply. Unsloth made the opposite call on 3.8.

Look at the last row before you draw anything from the third. **An MTP block is a full-size block.** The tensors that make it an MTP block are 4.5 MiB here and 41.1 MiB in the 3.8 file. Everything else in it is an ordinary block, and on this MoE that means a complete set of 256 experts, which is why Ornith's lands within 1% of a normal one. It costs about one more block, at whatever precision the quantizer picked.

Which leaves 254 MiB and 14 MiB as what they always were: net deltas between two whole files, upper bounds on the block rather than measurements of it. The statement that survives both is **a resident, unused MTP block is cheap, and how cheap is a property of the quant, not of MTP.** That is narrower than what I nearly wrote, and narrower is the point.

Where the 155 tok/s actually comes from

Four numbers went round the forums after the launch, and they are why the search traffic exists: 155 tok/s on an RTX 4090, 53 tok/s on a $329 RTX 3060, 170,000 tokens of context, and a 79% SWE-bench Verified scored locally. Seven YouTube channels covered Ornith 1.5 in nine days and most of them are auditing that list.

Trace it back and the decode figure is a **single social post about a single RTX 4090**, claiming 158 tok/s decode, 5,800 t/s prefill, 79% SWE-bench Verified and 250K context with a q8_0 KV cache. The 3060 figure is a different account, 53 tok/s with 900 t/s prefill. Both are one person on their own hardware. The vendor's own announcement publishes benchmark scores and no throughput at all.

Two things about that are worth your attention.

The number drifted in retelling. The post says 158. The videos say 155. Nobody has done anything dishonest; it is just what happens when a figure travels without its source attached.

Nothing in the circulated version names an engine or a quant. That is the gap. And the 24GB quickstart that travels alongside these claims points at **vLLM** for a 3090 or 4090, reserving llama.cpp for the 12GB tier. If the 155 came off a vLLM server, comparing it to a llama.cpp number is comparing two different programs.

I am not refuting it. I did not run a 4090 and I did not run vLLM, so I have no standing to. What I can tell you is that it is not an extraordinary claim: my 3090 does **172.44 tok/s** on this model at empty context under llama.cpp, and a 4090 has more bandwidth than a 3090. If anything the surprise is how ordinary 155 looks. The caveat runs the other way too — my figure is a synthetic `tg128` decode at up to 8K, and the 4090 claim is at 250K with a quantized KV cache. Those are different workloads and the long-context one will be slower. Both numbers are honest. They answer different questions.

What everyone else has measured

I went looking for a 3090 comparison of these two models before benching, and there isn't one. Closest is a Token Race run that put Ornith 1.5-35B Q4_K_M against Qwen 3.6-35B UD-IQ4_NL in llama.cpp on a Ryzen 7 with an RTX 4070 Ti Super — a different card, a different Qwen quant, and eleven real prompts from 30 to 77,376 tokens rather than a synthetic sweep.

It is worth reading, because it lands on the same shape from a different direction: Ornith decoding faster, Qwen winning prompt ingestion by a wide margin (1,309 against 900 t/s on their mixed-context workload). Different hardware, different quant, different harness, same two directions. Their prefill gap is far larger than my 3%, which is what you would expect once context depth and KV quantization enter the picture, and their absolute decode figures are much lower for the same reason.

The written coverage is quality-only. The comparisons you'll find of Ornith 1.5 against Qwen 3.6 are the vendor's published benchmark table: Terminal-Bench, SWE-bench Verified and Pro, GPQA Diamond, MCP-Atlas, reprinted without local throughput. Useful, and a different question from this one.

What this does not tell you

Nothing on this page says Ornith is a better model. I measured throughput, on one card, at one quant, with a synthetic benchmark. Whether it writes better code than Qwen 3.6 is untouched here, and every capability comparison in circulation traces back to DeepReinforce's own table, which nobody outside the lab has reproduced.

The other four limits, stated flat:

- **One quant pairing.** bartowski Q4_K_M against Unsloth UD-Q4_K_M. A different pairing could move the 9%.
- **One card.** A 3090 at 1950 MHz under a software power cap. Nothing here transfers to a 4090, a 3060, or Apple silicon.
- **Synthetic depths.** 0, 4K and 8K with a default f16 KV cache. Neither model was pushed toward its 262K ceiling, and the 3% prefill gap is measured where prefill is cheapest.
- **No speculative decoding.** Ornith's MTP block was resident and unused for the entire sweep. llama.cpp can drive it with `--spec-type draft-mtp`, and that run is not this run.

Bottom line

If you have 24GB and you run Qwen 3.6-35B-A3B today, Ornith 1.5-35B-A3B is worth the 21.86 GB. Not because 9% will change your afternoon. It won't; that's about a quarter-second across a thousand-token reply. It's because it costs you the same VRAM, drops into the same llama.cpp invocation, and gives you a second MIT-licensed model of the same shape to check work against. The download is the whole cost.

If your day is mostly pasting large context in and reading short answers back, stay where you are. Qwen ingests faster, the decode win is on the half of the workload you spend less time in, and 3% each way is not a reason to move.

And if you came here because of the 155 tok/s: it is one person's RTX 4090, the engine behind it was never stated, and the guide circulating with it recommends vLLM. My 3090 does 172 under llama.cpp. Ornith 1.5 is genuinely quick. It just isn't quick for the reasons the forum post gave you.

Related guides

- Best Way to Run Qwen 3.6 35B MoE Locally — the incumbent, measured on this same card
- Qwen 3.8 27B vs 3.6 on RTX 3090 — where the 254 MiB came from
- MoE Models Explained — why 35B-A3B reads like 3B and stores like 35B
- DFlash vs MTP on RTX 3090 — what the MTP block does when you actually switch it on
- Best Local Coding Models — where both of these sit against the field