How to Fix Slow Qwen 3.6 27B on RTX 3090 (10-80 tok/s)

May 1, 2026 · by Mark Bartlett

Download this guide as PDF

> **Quick Answer:** If you're seeing 10-19 tok/s on a 3090 with Qwen 3.6-27B, work this checklist in order: confirm full GPU offload (-ngl 99), reboot to clear VRAM fragmentation, switch to UD-Q4_K_XL (avoid IQ4_NL), apply the fixed Qwen 3.6 chat template, check for power-limited cards, set realistic context-length expectations (89 tok/s fresh drops to 30 at 128K), pick the right backend (llama.cpp 30-40, ik_llama.cpp 31-39 at 128K, vLLM with MTP can hit 80+), and only then look at SSM CPU bottlenecks. The first six fix 90% of slow setups.

📚 **More on this topic:** DFlash on RTX 3090: both Qwens benched · Qwen 3.6 Complete Guide · llama.cpp vs Ollama vs vLLM · VRAM Requirements

You spun up Qwen 3.6-27B on your RTX 3090, expecting the 35-80 tok/s you read about on r/LocalLLaMA, and you're sitting at 12. Maybe 18 on a good run. The model works, the output is fine, but something is wrong with the speed.

This is a real problem with real fixes. The r/LocalLLaMA can't-replicate thread ran 64 comments deep and surfaced the actual causes. Most are config issues that take a minute to fix. A couple are architectural. One is a backend choice with real tradeoffs. Work the list in order.

Image: Qwen 3.6 27B slow on RTX 3090 diagnostic flowchart

Quick reference

Symptom	Likely cause	Fix
10-15 tok/s, partial offload	Layers on CPU	`-ngl 99` explicit
15-20 tok/s, full offload	Wrong quant or template	UD-Q4_K_XL + fixed template
25-30 tok/s sustained	Backend tradeoff	Try ik_llama.cpp
30-40 tok/s, drops at 100K+	Normal prefill→decode shift	Working as expected
Want 60+ tok/s	Need vLLM + MTP spec decoding	Worth the complexity?
Want 2x on top of all that	DFlash speculative decoding	See DFlash bench

1. Verify full GPU offload

The first thing to check, every time. Look at the loader output.

```
load_tensors: offloaded 65/65 layers to GPU
```

If the first number is anything less than the second, you have layers on CPU and you'll never hit the speeds you want. Add `-ngl 99` (or `--n-gpu-layers 99`) explicitly. The default in some llama.cpp builds is 0 if you didn't specify, which silently puts everything on CPU. Multiple users in the thread reported "I'm at full GPU offload, why so slow" and the answer was that they weren't.

```
./build/bin/llama-server \
  --model qwen3.6-27b-UD-Q4_K_XL.gguf \
  -ngl 99 \
  --flash-attn on \
  ...
```

`-fa on` is also worth confirming. Flash attention shaves a measurable amount off generation time on the 3090 and is on by default in recent builds, but old configs sometimes carry it as off.

2. Reboot to clear VRAM fragmentation

Stupid. Real. After a long uptime with multiple model loads and unloads, NVIDIA's driver can leave VRAM fragmented enough that a 17 GB Q4 model won't allocate cleanly. The model still loads, but layers spill or contiguous memory pressure slows the kernels.

Three confirmations of this in the thread. One user reported a 20% speed jump after a reboot with no other changes. If you're getting weird numbers and `nvidia-smi` shows your VRAM allocated but the model loaded "successfully," reboot first and rerun. Costs you 60 seconds.

3. Quant choice matters more than people think

Not every Q4 is the same. The Unsloth Dynamic quants are calibrated against an importance matrix and they're noticeably better quality at the same speed than the bartowski or default GGUFs. Use **UD-Q4_K_XL** as the default. UD-Q4_K_M is the smaller variant if you need the headroom.

What to avoid: **IQ4_NL** is the worst Q4 variant for Qwen 3.6-27B according to multiple thread comments. The non-linear quant scheme that works on Llama-class dense models loses ground on the hybrid Gated DeltaNet attention path Qwen uses. Slightly slower, noticeably worse output. Skip it.

If you're at 16 GB or below and need a smaller quant, Q3_K_M is the fallback. Below that, output quality drops fast on the 27B.

4. The chat template is broken in many builds

This one bit a lot of people. Qwen 3.6 changed the chat template from 3.5, and several llama.cpp distribution builds shipped with the wrong embedded template. Symptoms: gibberish, missing tokens, the model talks to itself, refuses to stop generating, or throughput numbers are technically fine but the output is garbage.

Fix: pull the current Jinja template from the Qwen 3.6-27B model card and pass it explicitly with `--jinja --chat-template-file qwen3.6.jinja`. Or grab a known-good one from the Unsloth GGUF repo's tokenizer config. Don't trust the template baked into a GGUF older than April 25.

If you're using LM Studio, check the model card's "Prompt Template" field and override if it doesn't match Qwen's official format. Bad templates can also wedge the thinking-mode tags, so reasoning output disappears or leaks into the user-visible response.

5. Power limit on the 3090

Some 3090 cards ship with 320W or 350W power caps below the reference 350W spec, and aftermarket boards on shared power supplies sometimes get throttled by the user. Run:

```
nvidia-smi -q -d POWER | grep -i "power limit"
```

Look for `Current Power Limit`. If it's well under 350W, you're leaving speed on the table. The dense 27B is compute-bound at decode time on the 3090, and a 320W cap is roughly an 8-12% throughput hit. Not catastrophic, but if you're chasing the difference between 30 and 35 tok/s, this matters.

To raise it (with appropriate cooling and PSU):

```
sudo nvidia-smi -pl 350
```

Don't go over the rated TDP for your specific card. Founders Edition is 350W. Some AIB cards are higher. Check the spec.

6. Sustained vs initial throughput

This one isn't a bug. People misread the numbers.

Qwen 3.6-27B on a fresh empty context will hit ~89 tok/s on a 3090 for the first few hundred tokens. That number drops to ~30 tok/s sustained at 128K context. **That's the prefill-to-decode shift, not a regression.** Decode at long context is dominated by KV cache reads, and a 24GB card's memory bandwidth is the wall.

When the Reddit posters quote "80 tok/s on 3090," they're often quoting a fresh-context burst speed. When you measure your real workload at 32K of conversation history, you should expect 30-40, not 80. Set your expectations to "30-40 tok/s sustained at working context lengths" and you'll stop chasing a phantom problem.

If you really need higher sustained speed at long context, that's a vLLM-plus-speculative-decoding question, which is step 7.

7. Backend choice: llama.cpp, ik_llama.cpp, or vLLM

This is where the gap between "slow" and "fast" stops being a config issue and becomes a tradeoff.

llama.cpp. UD-Q4_K_XL on a 3090 lands at 30-40 tok/s sustained for typical chat. Multiple confirmations in the thread. This is the right baseline. If you're below it, the problem is in steps 1-6.

ik_llama.cpp. A llama.cpp fork with custom kernels for hybrid SSM models. Running `sokann/Qwen3.6-27B-GGUF-5.076bpw` (a 5-bit quant tuned for the fork) at full GPU offload, users in the thread report 31-39 tok/s decode at 128K context. About the same as llama.cpp at 4-bit, but at 5-bit quality. If you're VRAM-comfortable and want better output without the speed hit, this is the path.

vLLM with MTP speculative decoding. This is the 80+ tok/s territory. One thread member documented 82 tok/s sustained on a 3090 at 125K context using TurboQuant 3-bit NC weights, MTP draft, and `cudagraph_mode=PIECEWISE`. Critical detail: do not use `cudagraph_mode=FULL` with MTP. It produces garbled output. The setup is finicky and the quant is more aggressive, so quality drops a notch from UD-Q4_K_XL. The depth comparison is in llama.cpp vs Ollama vs vLLM.

The honest read: most people running into "slow" issues on llama.cpp don't actually need vLLM. They need their template fixed.

8. The SSM hybrid CPU quirk

The deepest cause, and the one most users do not have. Qwen 3.6 is a hybrid SSM architecture. Three Gated DeltaNet layers per Gated Attention layer. The DeltaNet recurrence step uses a small CPU-side compute buffer (around 552 MiB labeled `CUDA_Host buffer` in the loader output). On older CPUs without AVX-VNNI or AVX-512 (i9-9900K, i7-6700K, anything pre-2019 Intel), this bookkeeping path becomes a real bottleneck on a fast GPU.

You'll see `graph splits = 2` in the llama.cpp startup log and a small per-token CPU cost that matters when the GPU is doing 30+ tok/s. Newer CPUs (i7-12700K and up, Ryzen 5000+) close this gap and the SSM cost falls into the noise.

The original Reddit OP's Claude analysis of this bottleneck was directionally right but overstated. The CPU cost **is** real for SSM hybrid models, but it isn't the dominant factor for most users. If you're on a modern CPU and you're slow, the issue is in steps 1-7. If you're on a 2018-era CPU and stuck at low-20s with everything else clean, this is finally where you've bottomed out architecturally.

There's no software fix. CPU upgrade is the answer. If that's not on the table, switch to vLLM, which handles the hybrid path differently and is less sensitive to CPU vintage.

After all that, want another 2x?

If your 30-40 tok/s baseline is now solid and you want more, DFlash speculative decoding is the next step. I built and ran the bench on my own 3090 on April 30. The numbers came in at 2.56x mean speedup for Qwen 3.6-27B Q4_K_M against autoregressive on the same card. That moves a healthy 32 tok/s baseline to a healthy 84 tok/s. NVIDIA only, sm_86+, single GPU, batch=1.

DFlash is worth doing only after the checklist above. If your baseline is broken, DFlash is going to 2x a broken baseline. Get the simple stuff right first, then optimize.

The honest summary

Most “slow Qwen 3.6 on 3090” reports come down to one of three things:

1. **Layers on CPU because `-ngl` wasn’t set.** One flag fix.
2. **Bad quant or bad template.** Switch to UD-Q4_K_XL and update the Jinja template.
3. **Wrong expectations.** 80 tok/s is a fresh-context burst, not your sustained speed at 32K.

Steps 4-7 catch the next tier. Step 8 is a real architectural ceiling but it’s the minority case. Work the list top to bottom and you’ll find your bottleneck within 30 minutes.

The community thread is what made this list possible. Sixty-four comments of frustrated developers helping each other debug, and the answers shook out into the order above. If you find a new failure mode that isn’t covered here, that’s the place to post it.

Related guides

- DFlash on RTX 3090: I Benched Both Qwens
- Best Way to Get 2x Token Output on RTX 3090: Qwen 3.6 + DFlash
- Qwen 3.6 Complete Guide: 27B Dense, 35B-A3B MoE, and Which to Use
- llama.cpp vs Ollama vs vLLM: When to Use Each
- VRAM Requirements for Local LLMs

Get notified when we publish new guides.

Subscribe — free, no spam