Wicked Fast Qwen 3.6 27B: 60 tok/s with MTP on RTX 3090 (2026)

May 19, 2026 · by Mark Bartlett

Download this guide as PDF

> **Quick Answer:** On my RTX 3090, Qwen 3.6 27B Q4_K_M generates 60 tok/s with MTP on the latest llama.cpp branch — 1.86x wall-clock speedup over baseline across nine mixed prompts. That's up from 1.50x mean on May 6, after 185 commits of polish on am17an's mtp-clean branch. The PR #22673 flag renamed from `--spec-type mtp` to `--spec-type draft-mtp` — Unsloth's docs and DataCamp's tutorial both still reference the old name. n=3 wins on total wall time; n=2 wins on per-prompt throughput for 8 of 9 prompts. Pick by workload, not by blanket recommendation.

📚 **More on this topic:** DFlash vs MTP on RTX 3090 (May 6) · Qwen 3.6 Complete Guide · DFlash 2x Token Output · Speculative Decoding Explained

On my RTX 3090 + RTX 3060 12GB workstation, Qwen 3.6 27B Q4_K_M just hit 60 tok/s with MTP on the latest llama.cpp branch — roughly 1.6x faster than the same setup without MTP on mean per-prompt throughput, and 1.86x faster on wall-clock time across nine mixed prompts. PR #22673 is still draft, but 185 commits of polish since May 6 have moved the speedup needle from 1.50x mean to today's number.

Two headline numbers, both real. n=3 gives the better wall-clock total. n=2 gives the better per-prompt throughput on 8 of 9 prompts. Picking between them depends on workload, not on Unsloth's blanket recommendation. Below: the numbers, what changed in 13 days, and what to actually run today.

The bench

Hardware. Miu — single RTX 3090 24GB paired with an RTX 3060 12GB for the draft offload, Linux, CUDA 12, sm_86. Same workstation as the May 6 head-to-head against DFlash.

Model. RDson/Qwen3.6-27B-MTP-Q4_K_M-GGUF — 15.35 GiB target with the MTP layer baked into the same file. No separate draft weights to manage.

Bench harness. am17an's gist — nine mixed prompts at `n_predict=192`, `temp=0.0`, `seed=42`. Same harness as cturan's dual-GPU run, the Strix Halo reproduction, the M1 Ultra run, and my own May 6 bench. Numbers compose with the community thread.

llama.cpp build. am17an/llama.cpp at the `mtp-clean` branch, commit `2dff7ff8f` (May 19, 2026). Built with CUDA: `cmake -B build -DGGML_CUDA=ON -DCMAKE_BUILD_TYPE=Release`.

Server flags (common). `-ngl 99 -c 10000 --no-mmap --no-cache-prompt -fa on -np 1`. Baseline runs no spec flags. MTP runs add `--spec-type draft-mtp --spec-draft-n-max 3` (or `2`).

Three benches, three numbers

Same hardware, same model, same prompts. Three configurations.

Configuration	Mean tok/s	Wall (s)	Speedup vs baseline	Acceptance
Baseline (no MTP)	38.3	50.64	1.00x	n/a
MTP n=3	61.4	27.20	**1.86x wall** / 1.60x mean	71.1%
MTP n=2	63.8	34.44	1.47x wall / **1.67x mean**	80.5%

Two speedup metrics because they measure different things. Mean tok/s averages each prompt's individual generation rate — useful for comparing how fast tokens stream once generation starts. Wall-clock total time is what users actually feel: how long from prompt submitted to all responses complete across the full bench. For most readers building an agent or chat workflow, wall-clock is the metric that matters. Mean tok/s is the one Unsloth and DataCamp quote.

n=3's 1.86x wall-clock speedup is the practical headline. Same RTX 3090, same Qwen 3.6 27B Q4_K_M, same prompts. Add `--spec-type draft-mtp --spec-draft-n-max 3` and the bench finishes in 27.2s instead of 50.64s. That's a real, reproducible, today-available number — provided you build from the mtp-clean branch.

The surprise: n=2 isn't always the sweet spot

Unsloth's official Qwen 3.6 docs say to set `--spec-draft-n-max 2` for most setups. On my RTX 3090, per-prompt throughput agrees with them — n=2 wins 8 of 9 prompts on tok/s. But total wall time tells a different story.

Prompt	n=3 tok/s	n=2 tok/s	Winner
code_python	64.0	67.9	n=2
code_cpp	62.7	65.7	n=2
explain_concept	58.8	63.3	n=2
summarize	64.4	65.5	n=2
qa_factual	64.9	65.9	n=2
translation	55.1	60.8	n=2
creative_short	56.0	63.1	n=2
stepwise_math	68.6	58.1	**n=3**
long_code_review	57.8	64.3	n=2

n=3 wins exactly one prompt on per-prompt throughput: `stepwise_math`, at 68.6 vs 58.1 tok/s. That prompt's acceptance rate on n=3 was 80.8% — when the MTP layer's predictions match the target reliably, larger draft batches generate more accepted tokens per cycle, and the per-cycle gain dominates. n=2 is the safer default at 80% mean acceptance, but its smaller batch size means each verified cycle produces fewer tokens.

The wall-clock story is more dramatic. n=3 totals 27.2s; n=2 totals 34.44s. That entire 7.24s gap traces to one prompt: `long_code_review`. n=3 finished it in 5.77s; n=2 in 13.26s — a 7.49s delta on a single prompt. Every other prompt n=2 wins by a small margin.

The mechanism: prefill. PR #22673 documents that MTP slows prompt processing. On this run, n=2 spent ~10s on prefill for `long_code_review`'s ~700-token system prompt before generation even started. n=3 spent ~2.5s on the same prompt's prefill. n=2's larger prefill overhead is the wall-clock killer on long inputs.

The pattern: n=2 generates faster but starts slower. On short prompts that's a win. On long contexts the prefill penalty erases the per-token advantage and then some.

Practical takeaway:

- **Coding agents and short-prompt workflows** → `--spec-draft-n-max 2`. Higher acceptance, more consistent per-prompt throughput, what Unsloth recommends.
- **Long-form generation (analysis, creative writing, long code reviews)** → `--spec-draft-n-max 3`. Larger batches pay off when the model has runway to commit to a direction.
- Pick based on what you're running, not based on a blanket "n=2 is best" line. Both work. Workload determines which wins.

What changed since May 6

On May 6 I benched the same hardware against the same model on PR #22673 at commit `267f8afe8` and hit 1.50x mean MTP speedup. Today the same setup hits 1.86x wall-clock speedup. What moved the needle in 13 days:

- **185 commits on the `mtp-clean` branch.** am17an force-pushed continuously from May 4 onward. The branch is structurally rewritten in places — `handle_mtp_for_ubatch` lifted out of `llama_context` per ggerganov's review feedback.
- **GDN partial rollback shipped across all backends.** CUDA, Vulkan, and Metal all support it now. This is the core acceptance-rate improvement — when a draft batch contains a partial rejection, the rest of the batch's accepted tokens get kept instead of the whole batch being discarded.
- **Multi-backend support stabilized.** Metal's `keep_intermediates=true` path landed for unified-memory devices. Vulkan partial rollback works. CUDA was already working but got polish.
- **n-gram + MTP compatibility fixes.** The commit message `spec : fix compatibility with n-gram and add TODOs` lands a fix for the prior interaction issue.
- **The flag renamed.** `--spec-type mtp` is now `--spec-type draft-mtp`. The change unifies MTP under the `draft-*` taxonomy alongside `draft-eagle3` and `draft-simple`. **Documentation hasn't caught up:** Unsloth's docs and DataCamp's complete tutorial both still reference the old `mtp` name. Neither works on current mtp-clean builds.

Worth flagging: ggerganov, the llama.cpp maintainer, is now actively engaged. There's a `gg-mtp-rebase` branch on am17an's fork and a `gg/spec-parallel` branch in the main repo. Merge appears closer than "still draft" suggests. Best estimate: within weeks, not months.

For the May 6 baseline numbers and the head-to-head against DFlash on identical hardware, see the May 6 piece. DFlash still leads on mean speedup (2.56x vs MTP's 1.60x mean today), but the gap is narrower than 13 days ago and MTP's mainline path is more accessible.

The honest framing

Unsloth describes their Qwen 3.6 MTP GGUFs as "no longer experimental." DataCamp has published a complete tutorial that assumes the PR is the path. Both framings are technically correct — if you're building from am17an's `mtp-clean` branch and you ignore the flag-name drift.

What neither says clearly: PR #22673 has not merged. Master llama.cpp doesn't have MTP. Prebuilt llama.cpp release binaries don't have MTP. Anyone running `apt install llama-cpp` or downloading a release .zip gets nothing.

For most readers that's a fine state of affairs. Build from the PR branch today, the bench numbers are real, the speedup is reproducible. But "MTP merged into llama.cpp" hasn't actually happened yet. Watch for it — likely soon — but not today.

How to actually run MTP today (May 2026)

Five steps. Tested on Miu the day this article published.

1. Clone am17an's fork:

```
git clone https://github.com/am17an/llama.cpp.git
cd llama.cpp
git checkout mtp-clean
```

2. Build with CUDA:

```
cmake -B build -DGGML_CUDA=ON -DCMAKE_BUILD_TYPE=Release
cmake --build build --config Release --target llama-server -j
```

3. Download an MTP GGUF. Three options I'd consider:

- RDson/Qwen3.6-27B-MTP-Q4_K_M-GGUF — what I benched, 15.35 GiB, single Q4_K_M quant.
- unsloth/Qwen3.6-27B-MTP-GGUF — Dynamic 2.0 variants across multiple quants. Pick what fits your VRAM.
- havenoammo/Qwen3.6-27B-MTP-UD-GGUF — UD XL quants for users who want maximum quality fit.

All three include the MTP layer baked into the GGUF. No separate draft model to manage.

4. Run llama-server with MTP enabled:

```
./build/bin/llama-server \
  -m /path/to/Qwen3.6-27B-MTP-Q4_K_M.gguf \
  --host 127.0.0.1 --port 8080 \
  -ngl 99 -c 10000 -fa on -np 1 \
  --spec-type draft-mtp --spec-draft-n-max 3
```

Note the flag: `--spec-type draft-mtp`, not `--spec-type mtp`. If your build predates May 6 the flag is `mtp`. After the rename, it's `draft-mtp`. The Unsloth docs and DataCamp tutorial both still use the old name — they won't work on current mtp-clean builds without substitution.

5. For coding agents or short-prompt workflows:

```
--spec-type draft-mtp --spec-draft-n-max 2
```

Use 2 instead of 3. Per the table above, n=2 wins on per-prompt throughput for code generation and short responses. Pick by workload.

What this means for your local AI setup

For 24GB cards (RTX 3090, RTX 3090 Ti, RTX 4090): MTP is a clear win. Qwen 3.6 27B Q4_K_M at 60 tok/s is faster than most people can read in real time. Local coding agents that previously felt sluggish at ~40 tok/s become smooth. Long-context workflows benefit even more on n=3. See the Qwen 3.6 setup guide for the model overview and the coding-models roundup for context.

For 12-16GB cards: MTP costs roughly 2.5 GiB extra VRAM beyond the base model — the MTP layer plus its KV cache. On a 16GB card running 14B Q4 with a generous context, that overhead may not fit. Test before committing. If you're hitting VRAM pressure, the 35B-A3B MoE path trades model quality for headroom in a way MTP doesn't.

For Mac (M4 Max, M5 Max): This article is CUDA-only. Apple Silicon has its own MTP path through MLX — separate work, separate bench. Treat the numbers above as not portable.

Honest trade-offs

Three things MTP costs you alongside the speedup.

Prefill is slower. PR #22673's own benchmarks show ~0.51x prefill at long prompts on the same hardware. Time-to-first-token goes up. Generation speed goes from low-40s to low-60s, but you wait longer to see the first token, especially with large input contexts. For chat with short prompts the cost is invisible. For RAG with 8K+ context the cost is real.

VRAM overhead. ~2.5 GiB for the MTP layer plus its KV cache. At 27B Q4_K_M on a 24GB card, that's comfortable. On a 16GB card running 14B at any reasonable context, it may not fit. Run with `--no-mmap` and watch nvidia-smi during model load to see your actual footprint.

Build complexity. You're building llama-server from a draft PR branch. No prebuilt binaries on the llama.cpp releases page have MTP. CUDA toolchain required. If you've never built llama.cpp from source, expect a couple of hours on first try — see the Qwen 3.6 troubleshooting piece for the common errors. After that, rebuilds are fast.

What's next

PR #22673 will likely merge into llama.cpp master within weeks given ggerganov's active involvement and the rate of commits on `gg-mtp-rebase`. When that happens:

- Prebuilt binaries on the llama.cpp releases page will include MTP. The clone-and-build step goes away for most users.
- The flag name may shift again as the speculative-decoding subsystem unifies. Watch for it. Documentation will lag.
- Performance numbers may shift slightly post-rework. Whether the 1.86x wall-clock holds, exceeds, or regresses is what the post-merge bench will tell us.

I'll re-bench after merge and publish the follow-up. The newsletter catches the announcement when it lands.

Bench data, build details, and the per-prompt acceptance rates from this run are reproducible via am17an's gist. If you've got numbers from your own hardware, post them on the PR #22673 thread — that's where the maintainers are watching.

Get notified when we publish new guides.

Subscribe — free, no spam
