


# How to Get 2.5x Faster Qwen on RTX 3090 (Free)

April 30, 2026 · Updated: May 5, 2026 · by Mark Bartlett

[Download this guide as PDF](#)

**Quick Answer:** On my RTX 3090 24GB, the DFlash + DDTree bench (HumanEval, GSM8K, Math500 from bench\_llm.py) gives Qwen 3.5-27B Q4\_K\_M a mean speedup of 2.59x against autoregressive, and Qwen 3.6-27B Q4\_K\_M a mean of 2.56x. The Luce README headlines 3.43x for Qwen 3.5 on HumanEval and 1.98x mean for Qwen 3.6. My 3.5 lands below the headline; my 3.6 lands above it. The community is hitting the same gap – the r/LocalLLaMA can't-replicate thread is the same conversation. The speedup is real and worth building if you're on 24GB and running 27B for batch coding or math. The README's top number is currently a ceiling, not a target.

 **More on this topic:** [Best Way to 2x Token Output on RTX 3090](#) · [Qwen 3.6 Complete Guide](#) · [Best Way to Run Qwen 3.6-35B MoE Locally](#) · [Speculative Decoding Explained](#)

I built DFlash on my own RTX 3090 and ran the bench. Both Qwens, same harness, no shortcuts. The mean speedups: **2.59x for Qwen 3.5-27B Q4\_K\_M**, **2.56x for Qwen 3.6-27B Q4\_K\_M**. The Luce DFlash README headlines 3.43x on Qwen 3.5 HumanEval and a 1.98x mean for Qwen 3.6. My 3.5 is below the README headline. My 3.6 is above the README mean.

That's the article. The speedup is real. The gap to the headline is also real. Below are the numbers, the build steps that worked on my box, and the three honest reasons my 3.5 lags and my 3.6 leads.

Image: DFlash speedup chart for Qwen 3.5 and Qwen 3.6 on RTX 3090

---

## The numbers

---

Both runs used `python3 scripts/bench_llm.py` with defaults: HumanEval, GSM8K, and Math500 in sequence, batch=1, greedy decoding. Target weights are Q4\_K\_M GGUFs. Drafts are the matching `z-lab/Qwen3.5-27B-DFlash` and `z-lab/Qwen3.6-27B-DFlash`. Hardware is one RTX 3090 24GB on a Linux box, CUDA 12, sm\_86.

## Qwen 3.5-27B + DFlash

Bench	Autoregressive	DFlash	Acceptance length	Speedup
HumanEval	33.65 tok/s	<b>92.74 tok/s</b>	7.66	<b>2.76x</b>
GSM8K	33.66 tok/s	<b>83.33 tok/s</b>	6.75	<b>2.48x</b>
Math500	33.69 tok/s	<b>85.08 tok/s</b>	6.91	<b>2.53x</b>
<b>Mean</b>	<b>33.67 tok/s</b>	<b>87.05 tok/s</b>	<b>7.11</b>	<b>2.59x</b>

## Qwen 3.6-27B + DFlash

Bench	Autoregressive	DFlash	Acceptance length	Speedup
HumanEval	32.95 tok/s	<b>92.62 tok/s</b>	7.48	<b>2.81x</b>
GSM8K	32.86 tok/s	<b>73.90 tok/s</b>	5.97	<b>2.25x</b>
Math500	32.84 tok/s	<b>85.86 tok/s</b>	6.94	<b>2.61x</b>
<b>Mean</b>	<b>32.88 tok/s</b>	<b>84.13 tok/s</b>	<b>6.80</b>	<b>2.56x</b>

## Side-by-side, mine vs README

Bench	My 3.5	My 3.6	README 3.5	README 3.6
HumanEval	2.76x	2.81x	<b>3.43x</b>	2.24x
GSM8K	2.48x	2.25x	2.55x	1.71x
Math500	2.53x	2.61x	2.93x	1.99x
<b>Mean</b>	<b>2.59x</b>	<b>2.56x</b>	<b>2.97x</b>	<b>1.98x</b>

README numbers from the [Luce DFlash repo](#) and the [z-lab Qwen3.6-27B-DFlash card](#) (UD-Q4\_K\_XL, n\_gen=256). My 3.5 trails the README on every bench. My 3.6 beats the README on every bench. Both observations matter and I'll cover them in turn.

## Setup that actually worked

Five commands end-to-end. This is the compressed version of what landed on my disk after the dead ends.

```

git clone https://github.com/Luce-Org/lucebox-hub.git
cd lucebox-hub/dflash

# Build (~3 min on Linux + CUDA 12, sm_86 = RTX 3090)
cmake -B build -S . -DCMAKE_BUILD_TYPE=Release -DCMAKE_CUDA_ARCHITECTURES=86
cmake --build build -j2          # NOT -j -- full parallel OOMs even on 64GB RAM

# Symlink the matching draft for the run you're testing
ln -sf /models/Qwen3.5-27B-DFlash models/draft # path is hardcoded

# Run the suite (HumanEval + GSM8K + Math500, ~15-20 min total)
python3 scripts/bench_llm.py

```

Two real gotchas.

The `-j2` thing isn't a suggestion. On a 64GB box with default `-j`, the link stage of the custom CUDA kernels OOMs partway through. I hit it twice on different machines before I committed to `-j2` for everyone. If you're on 128GB+ you can probably go higher, but `-j2` is the safe default.

The other thing is the TLS quirk. The Baltimore CyberTrust root certificate expired in May 2025, and a couple of upstream model URLs still chain through it on some distros. If `wget` or `curl` errors out on cert validation while pulling weights, run `sudo dpkg-reconfigure ca-certificates` or edit `/etc/ca-certificates.conf` to disable the expired root with the `!mozilla/Baltimore_CyberTrust_Root.crt` prefix, then `sudo update-ca-certificates`. One-liner if your distro has it: `sudo update-ca-certificates --fresh`. This is a hosting-side artifact, not a DFlash issue, but it's the most likely thing to bite you on a fresh Linux install.

Run the bench twice for each model to be honest with yourself about variance. DDTree at temp=1.0 is non-deterministic and single runs swing a few percent.

---

## Why my 3.5 lags the README

The README headline for Qwen 3.5-27B is 3.43x on HumanEval. I got 2.76x. That's a real gap and worth explaining without trashing the project.

Three things I'd check, in order.

The first is that `bench_llm.py` defaults are not what produced the headline number. The script ships with what it ships with, and the README's 3.43x line could reflect a separate harness, a

different `n_gen`, a different prompt budget, or a tuned tree configuration. The README doesn't claim those defaults are tuned to peak. I'm reporting what `bench_llm.py` does out of the box, which is the right reproducible target for an audience that's going to pull the repo and run the same script.

The second is memory pressure. A single 3090 with 24GB has the Q4\_K\_M target (~16GB), the BF16 draft (~3.5GB), the captured-feature ring buffer, and KV cache all sharing the same pool. The DDTree verify stage spikes peak memory; bigger boxes (A40 48GB, RTX 6000 Ada) have more headroom and may be hitting fewer evictions or running with tree budgets I can't afford.

The third is plain variance. DDTree at temp=1.0 is non-deterministic by design. Single-bench runs swing a couple of percent. My 2.76x is one run with the published seed; a different seed gives a different number. The README's 3.43x might have been the lucky upper end of its own variance band rather than a steady-state number.

None of these are bugs. They're the gap between a paper-grade existence proof and what a hobbyist on a single 3090 will see at the prompt.

---

## Why my 3.6 beats the README

---

The flip side is the more interesting story. On Qwen 3.6-27B my mean is 2.56x; the README mean is 1.98x. Every bench is up. HumanEval went from 2.24x in the README to 2.81x on my run. GSM8K from 1.71x to 2.25x.

The simplest explanation is draft maturity. The Qwen 3.6-27B-DFlash draft was still training when the README was published on April 26. The HuggingFace card carries the "still training" note explicitly. I ran my bench on April 30 against a draft that's had four more days of optimizer steps. Acceptance length on HumanEval climbed from 5.94 in the README to 7.48 on my run. The draft is predicting longer correct prefixes than it did three days ago. Higher AL flows directly into higher speedup with the same DDTree budget.

That matches what should happen as a draft trains: the predictor gets better at the target's distribution, the verifier accepts more of the tree per pass, and the same kernel cost amortizes over more committed tokens. There's no magic here.

Frame it the right way. The 3.6 README numbers are a snapshot of an in-flight draft. They're going to keep climbing until the draft converges. The 3.5 README numbers are a snapshot of a fully-trained draft running on tuned settings, which sets a target that the out-of-the-box harness doesn't quite hit on a stock 3090. Both observations are useful. Neither is a complaint about the

project. Luce shipped a real GGUF port of block-diffusion speculative decoding to consumer hardware, which is the hard part.

---

## The community is seeing the same gap

---

The morning I finished my run, an OP on r/LocalLLaMA posted "[Can't replicate Reddit numbers with Qwen 27B on a 3090Ti](#)". The opening line: "I feel like I'm going insane. I see people here posting 30 - 100+ tok/s (100+ being with speculative decoding) on a 3090 with Qwen 3.6 27B. I'm trying to replicate this but my performance numbers are nowhere near that."

Their thread isn't strictly about DFlash, but it captures the same gap. Replies from other 3090 owners running stock LM Studio reported 38 tok/s on Qwen 3.6-27B Q4\_K\_M with default settings. One commenter on a 4090 reported 25-40 tok/s with llama.cpp Q4 GGUF and 50+ tok/s with vLLM MTP. A 2x3090 setup was the only one approaching the very-high numbers people quote.

That's the pattern. The published numbers are usually achievable, but they require a specific stack and tuning posture that the average reader doesn't share. My 2.59x and 2.56x are the numbers a hobbyist with `bench_llm.py` defaults gets on a single 3090. The 3.43x is what the README team got with their settings. Both are true. Calibrate accordingly.

---

## What about MTP?

---

A different speculative-decoding strategy is landing in mainline llama.cpp. [PR #22673](#) by `am17an` adds MTP (Multi-Token Prediction) for Qwen 3.6 and a few other models. The Reddit announcement called it "in beta" but the GitHub status is draft.

MTP bakes extra layers into the same checkpoint. Those layers predict tokens N+1, N+2, N+3 in parallel during a single forward pass; the predictions are treated as draft tokens. No separate draft model. PR tests Qwen 3.6-27B dense, Qwen 3.6-35B-A3B MoE, and Qwen 3.5-0.8B as a draft target. (The Reddit OP said "Qwen 3.5 MTP"; actual testing is Qwen 3.6.)

On a single RTX 3090, against the same Qwen 3.6-27B I benched above:

Approach	Speedup	Memory overhead	Backend
			CUDA via Luce fork

Approach	Speedup	Memory overhead	Backend
DFlash + DDTree (this article)	2.56x mean, 2.81x HumanEval	~3.5 GiB BF16 draft	
MTP (PR #22673)	~1.85x, 76% acceptance	~2.49 GiB MTP layer	<b>CUDA + Vulkan + Metal mainline (May 5)</b>

**Update – May 5.** Vulkan and Metal kernels both work now (am17an confirmed in the PR thread today). A dual-R9700 user got ~1.85x on Qwen 3.6-27B and +40-55% on Qwen 3.6-35B-A3B; an M1 Ultra hit ~1.5x on the 27B Q8\_0. cturan posted a 3090+3060 bench at 22.97 → 42.45 t/s, independently matching the PR's 1.85x. The PR is still draft – ggeranov asked for `handle_mtp_for_ubatch` to be extracted from `llama_context` first, so structural rework before merge. Google also released paired Gemma 4 MTP layers (`gemma-4-31B-it-assistant` and friends) today, likely a separate PR.

DFlash is ahead on raw speedup today. MTP wins on ergonomics: single GGUF, no separate draft repo, lands in mainline llama.cpp once merged. Two costs to know – MTP prefill runs roughly 0.51x with the layer enabled, which matters on long prompts, and acceptance varies 54-91% depending on workload (similar to DFlash).

They could compose. MTP draft tokens fed into DDTree verification is the obvious experiment, but nobody's built it yet.

I haven't reproduced MTP on Miu – that's the next bench. When PR #22673 merges, I'll run the same `bench_llm.py` suite I used for DFlash and post the head-to-head numbers.

---

## What this means for you

---

**Should you build DFlash today?** Yes if you're on 24GB VRAM and you run Qwen 3.5-27B or Qwen 3.6-27B for batch coding, math, or reasoning workloads. Mean 2.5x on real hardware is real. The first three minutes of your build will be the cmake stage, and `-j2` will save you the second pass. Whether you see 2.5x or 3x is bench-dependent and seed-dependent; the order of magnitude is solid.

**Are the README headline numbers achievable?** Probably, with tuning beyond `bench_llm.py` defaults. I haven't replicated 3.43x on my 3090 and the wider community thread suggests nobody on stock settings is hitting it yet either. If the Luce team posts a tuned bench script or a tree-config preset, that's the day to retry. Until then, treat 3.43x as a ceiling and 2.5-2.8x as the working number.

**What gets better from here?** Three things to watch. The Qwen 3.6 draft is still training, so my 2.56x mean will likely climb on each fresh checkpoint pull. The llama.cpp CUDA path is getting NVFP4 acceleration on Blackwell (see [our FP4 guide](#)), and the same kernel work tends to lift the dp4a paths on Ampere a few percent. KV cache quantization (TQ3\_0 is on by default in DFlash builds) frees headroom that lets you push DDTree budgets higher.

If you're sizing hardware for 27B-class local work specifically, the [VRAM requirements guide](#) is the reference. A used RTX 3090 at \$700-850 is still the right buy, and DFlash is one more reason that holds.

---

## Run the bench yourself

---

The repo is [Luce-Org/lucebox-hub](#). The drafts are [Qwen3.5-27B-DFlash](#) and [Qwen3.6-27B-DFlash](#). Targets are any current [Unsloth Qwen3.5-27B-GGUF](#) or [Qwen3.6-27B-GGUF Q4\\_K\\_M](#). The bench script is `scripts/bench_llm.py`.

If your numbers come back wildly different from mine, either way, post them. The dataset of real-world DFlash runs on real hardware is what closes the gap between README and prompt. My run is one row.

Get notified when we publish new guides.

[Subscribe – free, no spam](#)

---

Source: <https://insiderllm.com/guides/dflash-rtx-3090-bench-both-qwens/>

Free guides for running AI locally