Best Way to Get 2x Token Output on RTX 3090: Qwen 3.6 + DFlash

April 27, 2026 · Updated: May 1, 2026 · by Mark Bartlett

Download this guide as PDF

> **Quick Answer:** Two huge wins for RTX 3090 owners running Qwen 3.6-27B. DFlash speeds up decode — I reproduced a ~2.5x mean speedup on my own 3090, pushing Q4_K_M into 80+ tok/s territory. PFlash (announced May 1) cuts the long prefill wait by ~10x at 128K context — 24.8 s TTFT vs 248.4 s vanilla llama.cpp. The two compose in one C++/CUDA process: PFlash trims the prompt to the spans that matter, DFlash + DDTree decodes the answer. Linux, NVIDIA sm_80+, single 24GB card. A fundamentally different 27B experience on hardware most local-AI readers already own.

📚 **More on this topic:** Qwen 3.6 Complete Guide · Best Local Coding Models · Speculative Decoding Explained · VRAM Requirements

Your RTX 3090 is leaving roughly half its throughput on the table when you run Qwen 3.6-27B. The autoregressive ceiling on a single 3090 with the Q4_K_M GGUF sits around 35 tok/s. With Luce DFlash plus DDTree wired into the same llama.cpp graph, the published numbers double it. 78 tok/s on HumanEval. 70 tok/s on Math500. Mean 1.98x across the standard suite, single-user, batch=1, greedy decoding.

This is the first GGUF port of block-diffusion speculative decoding I've seen for consumer hardware. The paper-grade speedups for Qwen 3.5-27B were the bigger headline — 3.43x on HumanEval, 207 tok/s peak with a 128K context — but those numbers are an existence proof. The story for most readers is the Qwen 3.6-27B port that landed April 26, because that's the model people are actually running.

Image: DFlash speculative decoding speedup chart for Qwen 3.6-27B on RTX 3090

PFlash update — 10x prefill speedup (May 2026)

The same Luce-Org team that shipped DFlash announced **PFlash** on May 1. DFlash speeds up decode — the token-by-token generation after the prompt is processed. PFlash speeds up the other half: **prefill**, the long pause before the first token comes back when you hand a 64K or 128K prompt to a 27B model.

Headline claim, from the r/LocalLLaMA announcement and the lucebox-hub repo (MIT, same monorepo as DFlash): **24.8 s TTFT vs ~248.4 s vanilla llama.cpp at 128K on Qwen3.6-27B Q4_K_M, RTX 3090 — 10.0x cold**. At 64K it's 13.5 s vs 134.95 s, also 10.0x. Warmed-cache llama.cpp settles to 169.3 s at 128K, so warmed-vs-warmed is smaller. NIAH single-needle retrieval preserved at every context tested — and the team flags that NIAH single-needle is structurally easy for an attention-based selector, with RULER and multi-needle audits pending.

The good news for 3090 owners: **PFlash and DFlash compose in the same process**. PFlash compresses the prompt to the spans that matter; DFlash + DDTree decodes the answer at the ~74 tok/s the existing benchmarks already cover. Same 24GB card, same C++/CUDA stack.

Newly announced May 1, not yet reproduced firsthand on Miu. Reproduction notes will land in the DFlash bench reproduction piece.

The Headline Numbers

Same hardware, same harness, same model, two decode strategies. RTX 3090, Qwen 3.6-27B Q4_K_M (Unsloth Dynamic), batch=1, greedy.

Benchmark	Autoregressive	DFlash + DDTree	Speedup	Acceptance Length
HumanEval	34.90 tok/s	**78.16 tok/s**	**2.24x**	5.94
Math500	35.13 tok/s	**69.77 tok/s**	**1.99x**	5.15
GSM8K	34.89 tok/s	**59.65 tok/s**	**1.71x**	4.43
Mean	**34.97 tok/s**	**69.19 tok/s**	**1.98x**	**5.17**

Numbers from the Luce DFlash README as of April 27, 2026.

For reference, Qwen 3.5-27B on the same rig hits 129.52 tok/s on HumanEval (3.43x), 110.51 on Math500 (2.93x), 96.15 on GSM8K (2.55x). The 3.5 draft is fully trained. The 3.6 draft is still training. The gap between them is what closes over the next few weeks.

That 1.98x mean for 3.6 is also 2.8x faster than running Qwen 3.5-27B AWQ on SGLang on the same 3090, per the README's comparison table. I have not independently re-run that comparison. The headline tok/s numbers are reproducible against the public bench scripts.

Why This Works

Standard speculative decoding runs a small draft model that proposes a chain of tokens, then the big target model verifies them in one pass and commits the longest accepted prefix. Eagle-style chains hit ceilings around 4-5 accepted tokens per step on most workloads.

DFlash flips two things. First, the draft is block-diffusion: instead of generating tokens one at a time, the draft receives `[last_target_token, MASK x 15]` plus the last 5 captured target hidden states and denoises all 16 mask positions in a single forward pass. Every position conditions on the same captured context rather than its own noisy predictions. That is structurally different from a chain draft. The acceptance lengths in the README — 8.33 on Qwen 3.5 with the matched draft — show the difference.

DDTree is the verification side. Instead of a single 16-token chain, it builds a best-first tree of up to 22 nodes spanning the top-K branches at each draft position. One target forward pass verifies the entire tree using a causal mask derived from parent pointers. Budget=22 is "where draft accuracy plateaus" per the README; pushing higher costs more verify work without commensurate gains.

The cleverness on the implementation side is state management. Before each verify, the target's recurrent state — SSM intermediate, conv window, KV cache — gets snapshotted and restored after acceptance. Three custom CUDA kernels (`ggml_gated_delta_net_tree_persist` , `ggml_ssm_conv_tree` , plus a sliding 4096-slot `target_feat` ring) keep the bookkeeping cheap enough that the speedup survives all the way out to 128K context. Without that ring buffer, holding captured features for 128K would burn 6.6GB by itself.

If speculative decoding is unfamiliar territory, my primer is here. The DFlash and DDTree papers are arXiv:2602.06036 and arXiv:2604.12989 respectively.

Hardware Reality

What you need:

- NVIDIA GPU at sm_86 or newer. RTX 3090, A10, A40, RTX 4090, RTX 5090, Jetson AGX Thor (sm_110, needs CUDA 13+). The 3090 is the reference card in the README.
- 24GB VRAM. The Q4_K_M target is around 16GB, the BF16 draft is 3.46GB, the rest is KV cache and the captured-feature ring buffer.
- CUDA 12+. CMake 3.18+. Linux. The custom kernels are not portable to Metal or ROCm.

- About 80GB free disk for the build, model weights, and cache. The target GGUF alone is ~16GB.

What does not work today:

- **No Mac.** Apple Silicon users running Qwen 3.6 should stay on MLX or llama.cpp Metal. DFlash's three custom CUDA kernels do not have Metal equivalents.
- **No AMD.** Same story for ROCm. RX 7900 XTX and MI300X are out of scope.
- **No multi-GPU.** Single device, single user. This is a chat-grade local-AI tool, not a serving backend.

If you're sizing a card around this specifically, see the VRAM requirements guide. A used RTX 3090 at $700-850 is still the right purchase for 27B-class local work, and DFlash is one more reason that holds.

Setup

These commands follow the README's quick-start verbatim. I haven't run them locally yet — your mileage on huggingface gating may vary.

Clone and build:

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

cmake -B build -S . -DCMAKE_BUILD_TYPE=Release
cmake --build build --target test_dflash -j
```

Download models (Qwen 3.5 path — the matched draft is fully trained):

```
huggingface-cli download unsloth/Qwen3.5-27B-GGUF \
  Qwen3.5-27B-Q4_K_M.gguf --local-dir models/

huggingface-cli download z-lab/Qwen3.5-27B-DFlash \
  model.safetensors --local-dir models/draft/
```

The z-lab repos are gated. Set `HF_TOKEN` in your environment before downloading. For the Qwen 3.6 path, swap `Qwen3.5` for `Qwen3.6` in both lines and accept the gating terms on the z-lab/Qwen3.6-27B-DFlash page first.

Run a one-shot generation:

```
python3 scripts/run.py --prompt "def fibonacci(n):"
```

Multi-turn chat:

```
python3 examples/chat.py
```

OpenAI-compatible HTTP server (for plugging into Aider, Continue, etc.):

```
python3 -m venv .venv
.venv/bin/pip install fastapi uvicorn transformers jinja2
.venv/bin/python scripts/server.py --port 8000 --daemon
```

Reproduce the benchmark numbers:

```
python3 scripts/bench_he.py --n-gen 256 --ddtree-budget 22
python3 scripts/bench_llm.py
```

For the long-context mode (up to 256K with KV in TQ3), the README documents an env-var-driven invocation that switches the prefill strategy. That's a power-user path; defaults are fine for chat and code work.

Honest Limits

A few things worth knowing before committing a weekend to this.

Greedy only. Temperature and top-p are accepted by the OpenAI server but ignored in the verify path. If you depend on sampling for variety — creative writing, RP, anything where determinism is

a downside — DFlash is not your tool today. The contributing section flags "temperature / top-k sampling in the verify path" as a good first PR. Until that lands, treat this as a tool for code, math, and reasoning workloads where greedy is fine.

Batch=1, single user. This is a single-stream consumer-GPU tool. If you're running a multi-user serving stack, you want vLLM or SGLang. DFlash trades batching headroom for single-stream peak speed.

Q4_K_M costs accept rate. The DFlash paper reports its acceptance rates against a BF16 target. Running against Q4_K_M loses about 30 points of per-position acceptance. The published 5.17 mean acceptance length on Qwen 3.6 is what survives that quantization tax. If you can spare the VRAM for Q5_K_M or Q6_K, expect higher numbers.

One model pair. The DFlash port targets Qwen3.5-27B and Qwen3.6-27B specifically. The bookkeeping kernels assume the qwen35 architecture (which Qwen 3.6 inherits — same hidden dim, same layer count). Other models — Llama, Mistral, Gemma, DeepSeek — would each need their own draft training and graph rewriting.

The Qwen 3.6 draft is still training. Today's 1.98x mean is interim. The matched-draft AL is 5.05 on the snapshot z-lab pushed April 26. The mismatched 3.5-on-3.6 fallback hits 4.74 — about 22% drop from cross-generation drift. As the matched draft finishes training, expect the AL to push toward the 8.33 the 3.5 draft hits today. That math suggests a 3x mean speedup on Qwen 3.6 is plausible by mid-May. I'm not betting on it. I am noting it.

Linux only. The custom CUDA kernels assume a Linux build environment. Windows + WSL2 will probably work; nobody has reported it yet.

Where DFlash Fits

If you're running Qwen 3.6-27B for coding or reasoning on a single 3090 today, DFlash is the highest-impact tool I've seen this month. The default llama.cpp path runs around 35 tok/s on this hardware. DFlash gets you to 60-78 tok/s depending on workload. That is the difference between "feels like a code assistant" and "feels like a chat assistant." It's chat-grade speed on a single consumer GPU without spending $6,000 on a multi-3090 rig.

If you're on Mac, this isn't your tool — see the Mac local AI guide for the MLX route. If you're on AMD, watch the space; an MXFP4-aligned ROCm port is plausible but not announced.

If you're not on Qwen 3.6 at all — running Llama 4, Mistral Large, or DeepSeek V3.2 — DFlash isn't applicable yet. The single-pair limitation is real. The path forward in the README mentions

full llama.cpp integration with `llama-speculative-dflash.cpp` and `llama-cli` / `llama-server` wiring, plus more model pairs. Watch the repo.

What's Next

The Luce team's roadmap, per the README:

- Temperature and top-k sampling in the verify path
- Full llama.cpp integration as a new architecture (`llama-speculative-dflash.cpp`), with `llama-cli` and `llama-server` wiring
- More model pairs beyond Qwen 27B
- Blackwell sweep (RTX 5090, RTX PRO 6000 Blackwell, B200) — the FP4 tensor cores plus DFlash's tight verify pass should compound

The interesting downstream question is what happens when the NVFP4 path that just landed in llama.cpp meets DFlash on a 5090. Smaller weights mean faster prefill and verify. Block-diffusion drafts cut the dependency chain. On paper, those compose. Nobody has run that benchmark yet.

Bottom Line

The simplest version of this article: if you have a 3090 and you run Qwen 3.6-27B, install DFlash, follow the quick-start, and you'll get roughly 2x the tokens per second on the workloads people actually run. The numbers are published, the code is MIT, the matching draft is going to keep improving. Single-user, greedy, NVIDIA-only. That's the deal.

For Qwen 3.5-27B users, the speedup is bigger today (3.43x on HumanEval) because the draft is fully trained. If your workflow tolerates 3.5, run that.

Project home: github.com/Luce-Org/lucebox-hub. Project blog: lucebox.com. DFlash paper: arXiv:2602.06036. DDTree paper: arXiv:2604.12989.

Related Guides

- Qwen 3.6 Complete Guide: 27B Dense, 35B-A3B MoE

- Best Local Coding Models Ranked (2026)
- Speculative Decoding Explained
- VRAM Requirements for Local LLMs
- Best Local LLMs for Mac in 2026 (if you're on Apple Silicon, DFlash isn't your path — start here)
- FP4 Just Landed in llama.cpp: NVFP4 vs MXFP4 Explained
- llama.cpp vs Ollama vs vLLM

Sources: Luce DFlash README, z-lab/Qwen3.5-27B-DFlash, z-lab/Qwen3.6-27B-DFlash, DFlash paper (arXiv:2602.06036), DDTree paper (arXiv:2604.12989), lucebox.com.

Get notified when we publish new guides.

Subscribe — free, no spam
