Wicked Fast Gemma 4 vs Qwen 3.6 on RTX 3090: 3.10x Tested

May 8, 2026 · by Mark Bartlett

Download this guide as PDF

> **Quick Answer:** Wicked fast on a single RTX 3090. Gemma 4 26B-A4B Q4_K_XL: 128 tok/s mean. Qwen 3.6-27B Q4_K_M on the same workstation, same llama.cpp build, same bench: 41 tok/s. Speedup: 3.10x. Both fit in ~17 GB VRAM. MoE math doing real work — Gemma 4 activates 8 of 128 experts per token. The catch: Gemma 4 forces a reasoning trace by default; fix is `--jinja` plus `--chat-template-kwargs '{"enable_thinking":false}'`. Want speed on a 3090, Gemma 4. Want simpler ergonomics, Qwen 3.6.

📚 **More on this topic:** DFlash vs MTP on RTX 3090 (May 6 head-to-head) · DFlash on RTX 3090 (April 30 firsthand bench) · Qwen 3.6 Complete Guide · Gemma 4 Local AI Guide · Run Qwen 3.6-35B MoE Locally · Run 31B Models on a Laptop with LARQL

I ran Gemma 4 26B-A4B and Qwen 3.6-27B on the same RTX 3090, same llama.cpp build, same bench harness, back-to-back. Gemma 4 was **3.10x faster on decode**: 128.08 tok/s mean against Qwen's 41.27 tok/s. Both fit in roughly the same VRAM. Same Q4 quant tier. The numbers below are firsthand from Miu, my workstation 3090.

The MoE math is doing real work, and the architectural details below show why this gap should hold up. The thinking-mode quirk in the middle of the article is the one thing that'll bite you if you swap Gemma 4 in cold.

Why this comparison matters today

Today is the day Gemma 4 went viral on r/LocalLLaMA. A score-16 thread showed Gemma 4 26B at **600 tok/s on a single RTX 5090**. A score-12 thread reported MTP for Gemma 4 in llama.cpp delivers another ~40% speedup on top. A score-8 thread confirmed NVFP4 GGUFs are shipping for Blackwell. The Gemma 4 family has the local-AI community's full attention.

But 600 tok/s is on RTX 5090. What does Gemma 4 26B-A4B actually do on a single RTX 3090 — the consumer-hardware floor that most readers actually have? That's what this article answers. And it pairs the answer with a head-to-head against the model 3090 owners have been running for the past month: Qwen 3.6-27B.

Same hardware, same llama.cpp mainline build, same bench script. Apples-to-apples on what most readers can actually run.

The setup

Hardware. Miu — single RTX 3090 24GB on Linux (Xubuntu), CUDA 12, sm_86. No dual-GPU configuration. Same workstation that ran the April 30 DFlash bench and the May 6 DFlash vs MTP head-to-head.

Build. llama.cpp mainline at commit `f9cd456ea` (b9079, 2026-05-08 17:46 CET). Built fresh today with CUDA support and static linking (`BUILD_SHARED_LIBS=OFF`) to avoid library conflicts on a workstation that has multiple llama.cpp variants installed:

```
cmake -B build -DGGML_CUDA=ON -DBUILD_SHARED_LIBS=OFF
cmake --build build --config Release -j 3
```

`-j 3` keeps the build polite — full parallelism on a 24-thread Ryzen will pin every core and the workstation gets unusable. `-j 3` finishes in a few minutes and you can keep working.

Why mainline. Gemma 4 needed several llama.cpp commits to work properly. `git log --since="2026-04-15" --grep gemma` lists five commits worth knowing about: model type detection (`fcc750875`), tensor-parallel MoE AllReduce fix (`fd6ae4ca1`), NVFP4 tensor support (`f772f6e43`), common parser edge cases (`4414c04b9`), and a recent commit that ignores non-language tensors (`a736e6c0a`). The May 6 am17an MTP fork branch was missing some of these. b9079 is the floor I'd trust today.

Models.

Model	Quant	Disk	GPU	Architecture
Gemma 4 26B-A4B	Q4_K_XL (Unsloth, Apache 2.0)	15.83 GiB	~17 GB	30 layers, 128 experts, 8 active per token
Qwen 3.6-27B	Q4_K_M	~16 GB	~16 GB	Standard dense transformer

The Gemma 4 architecture has two more details that matter for the speed story. Hybrid attention: 5 of every 6 layers use sliding-window attention (1024-token window), 1 in 6 uses full

attention. Hybrid GQA: most layers run 8 KV heads, the full-attention layers run 2 KV heads (8x KV-cache compression on those layers). Both reduce memory pressure during decode.

Bench script. am17an's gist — the community standard since PR #22673 testing started. Same harness as the May 6 DFlash vs MTP article. Nine prompts: `code_python`, `code_cpp`, `explain_concept`, `summarize`, `qa_factual`, `translation`, `creative_short`, `stepwise_math`, `long_code_review`. `n_predict=192`, `temperature=0.0`, `seed=42`, `cache_prompt=False`.

Server flags (Gemma 4):

```
-fa on -c 10000 -np 1 -ngl 99 --no-mmap --no-cache-prompt \
  --jinja --chat-template-kwargs '{"enable_thinking":false}' \
  --port 8090
```

The `--jinja` and `--chat-template-kwargs '{"enable_thinking":false}'` combination is load-bearing — see The thinking-mode quirk below. Without it, Gemma 4 outputs go to the `reasoning_content` field instead of `content` and the bench script measures wrong.

Server flags (Qwen 3.6):

```
-fa on -c 10000 -np 1 -ngl 99 --no-mmap --no-cache-prompt --port 8090
```

No `--jinja`, no template kwargs. Qwen 3.6 lets the prompt decide.

Run order. Both runs back-to-back on the same workstation, same day, no reboot in between. Server stopped between runs to free VRAM cleanly. No background load, no other models warm in memory. Whatever variance exists is bench-harness variance, not workstation drift.

The numbers

Same harness, same prompts, both runs same day on Miu.

Prompt	Qwen 3.6-27B (tok/s)	Gemma 4 26B-A4B (tok/s)	Speedup
code_python	41.6	129.7	3.12x

Prompt	Qwen 3.6-27B (tok/s)	Gemma 4 26B-A4B (tok/s)	Speedup
code_cpp	41.3	124.6	3.02x
explain_concept	41.6	129.9	3.12x
summarize	42.1	129.7	3.08x
qa_factual	41.5	129.8	3.13x
translation	39.9	125.5	3.15x
creative_short	41.2	130.8	3.17x
stepwise_math	41.4	129.5	3.13x
long_code_review	40.8	123.2	3.02x
MEAN	**41.27**	**128.08**	**3.10x**

Aggregate over 1,581 predicted tokens: Qwen wall-time 40.69s, Gemma 4 wall-time 13.26s, ratio 3.07x.

Range: Qwen 39.9-42.1 (5.5% spread). Gemma 4 123.2-130.8 (6.2% spread). Both extraordinarily consistent across prompt types — no scenario where one model unexpectedly tanked or surged. Translation and `long_code_review` ran slightly slower for both, which matches what you'd expect from prompt structure (more state to track), but the gap held at ~3x throughout.

The Qwen numbers track within rounding distance of the May 6 DFlash vs MTP article's autoregressive baseline (~42 tok/s on the same hardware). That's the apples-to-apples sanity check — same model, same quant, same workstation, same harness, same baseline week to week.

The architectural story

Why is Gemma 4 26B-A4B 3x faster despite roughly the same total parameter count?

MoE math. Qwen 3.6-27B is dense — every token uses all 27B params for the forward pass. Gemma 4 26B-A4B activates **8 of 128 experts per token**, plus shared layers. Effective active-params-per-token is far less than 26B. The total VRAM footprint is similar (~16-17 GB) because all experts must live in memory; only a fraction get used per token.

For the RTX 3090 specifically, this is a memory-bandwidth story, not a compute story. Decode on a 3090 is bandwidth-bound — the GPU finishes the matrix multiply faster than it can load

weights. Less weight per token directly translates to more tokens per second. That's the core architectural lever Gemma 4 26B-A4B is pulling.

The GGUF metadata reveals additional Gemma 4 quirks worth flagging:

- **30 layers total**
- **Hybrid attention pattern** — 5 of every 6 layers use sliding-window attention (1024-token window); 1 in 6 uses full attention
- **Hybrid GQA pattern** — most layers run 8 KV heads; full-attention layers run 2 KV heads (8x compression on those layers)
- **128 experts, 8 active per token**
- **2816 embedding dimension, 16 attention heads**
- **256K trained context window** (we tested at 10K)

The "A4B" branding refers to active-billion parameters in some dense-equivalent computational sense, not strict active-params-per-token. Total parameters: 25.23B. Whatever internal metric Google is using, the practical effect on a 3090 is real and measurable in the table above.

For a deeper write-up on the trade-offs of running 26-35B MoE models locally, see Run Qwen 3.6-35B MoE Locally. For the alternative approach of decoupling attention from FFN entirely, see LARQL.

The thinking-mode quirk

Both models have thinking modes. They behave differently, and Gemma 4's default will surprise you the first time.

Gemma 4 26B-A4B forces a reasoning trace by default. Even with `--reasoning-budget 0` or `--chat-template-kwargs '{"thinking":false}'`, the output goes to the `reasoning_content` field instead of `content`. The bench script reads `content`, so the speed measurement comes back as zero. The fix:

```
--jinja --chat-template-kwargs '{"enable_thinking":false}'
```

The `--jinja` flag is load-bearing — without it, the chat-template kwargs don't reach the template engine. The load log will still print `thinking = 1` (cosmetic bug), but the output correctly populates `content` and skips the reasoning trace. This is a documented issue in llama.cpp discussions.

Qwen 3.6-27B lets the prompt decide. It emits empty `<think></think>` tags, then proceeds straight to the answer. No flag tweaks needed.

That difference matters for production deployment. A team swapping Gemma 4 in for Qwen 3.6 will hit empty `content` fields on day one and spend an hour figuring out why. If you ship Gemma 4 anywhere user-visible, run with the `--jinja` + `enable_thinking=false` combo until you actively want reasoning enabled.

What's not measured here

This bench measures decode speed on short prompts. It does not measure:

- **Output quality.** Gemma 4's answers feel coherent in ad-hoc testing, but this article doesn't bench quality. Use the Gemma 4 guide and the Qwen 3.6 guide for benchmark comparisons on reasoning, coding, and instruction-following.
- **Long-context behavior.** Both models support far more than the 10K context I ran. Gemma 4's hybrid SWA pattern and Qwen 3.6's GQA both have known scaling characteristics; neither was stressed here.
- **Prompt eval / prefill speed.** am17an's gist focuses on decode. Long-prompt prefill is a different bench.
- **Multimodal behavior.** Gemma 4 is image-text-to-text per the GGUF metadata. Untested in this run.
- **Speculative decoding.** No draft models loaded for either side. Gemma 4 has paired MTP layers (Google released them May 5) but mainline llama.cpp didn't have the Gemma-side MTP path at b9079. The May 6 DFlash vs MTP article covers what spec decode adds for Qwen 3.6 on the same hardware.

Worth noting: today's r/LocalLLaMA thread on MTP for Gemma 4 in llama.cpp suggests Gemma 4 + MTP should hit roughly 180 tok/s on this hardware once the PR merges and stabilizes. Future article.

Practical recommendation

If you're on a single RTX 3090 right now, today, May 2026:

Want speed and willing to manage Gemma 4's thinking-mode quirk: Gemma 4 26B-A4B Q4_K_XL. 3x faster decode for the same VRAM footprint, similar Q4 quality tier, similar disk size.

Want simpler ergonomics: Qwen 3.6-27B Q4_K_M. Predictable behavior, well-documented in the existing community thread, no flag dance. Slower but rock-solid.

Want highest quality, willing to bench yourself: try both for your specific task. This article doesn't tell you which produces better outputs — only how fast they produce them. The right answer is task-dependent and depends on your prompt patterns.

Both run cleanly on a single 3090 with mainline llama.cpp at b9079. Pick based on your workload and your tolerance for the thinking-mode flag dance.

What's next

When the Gemma 4 MTP path lands in mainline llama.cpp, I'll re-run this bench against the merged build to see what the +40% speedup actually looks like on a 3090. The PR is downstream of the Qwen-focused PR #22673 but uses Gemma 4's KV-cache-sharing architecture and dynamic draft length heuristics, so it'll likely land separately.

The DFlash + MTP composition experiment from the May 6 article still hasn't been built. Once the Gemma 4 MTP path is in, a three-way Gemma 4 + MTP vs Qwen 3.6 + DFlash vs Qwen 3.6 + MTP becomes the natural follow-up. Newsletter Issue 9 will summarize once those numbers exist.

If you've got numbers from your own RTX 3090 running either model, post them in the r/LocalLLaMA Gemma 4 thread — that's where the community is converging right now.

Methodology footnotes

Build details. Single RTX 3090 24GB on Linux (Xubuntu), CUDA 12, sm_86. llama.cpp mainline at commit `f9cd456ea` (b9079, 2026-05-08 17:46 CET). Built with `cmake -B build -DGGML_CUDA=ON -DBUILD_SHARED_LIBS=OFF` then `cmake --build build --config Release -j 3`.

Gemma 4 weights. unsloth/gemma-4-26B-A4B-it-GGUF, Q4_K_XL quant, Apache 2.0, ~15.83 GiB on disk.

Qwen 3.6 weights. Standard Q4_K_M GGUF, ~16 GiB on disk.

Bench script. am17an's gist — same harness as the May 6 DFlash vs MTP article. Nine prompts, `n_predict=192`, `temperature=0.0`, `seed=42`, `cache_prompt=False`.

Server flags (Gemma 4). `-fa on -c 10000 -np 1 -ngl 99 --no-mmap --no-cache-prompt --jinja --chat-template-kwargs '{"enable_thinking":false}' --port 8090`.

Server flags (Qwen 3.6). `-fa on -c 10000 -np 1 -ngl 99 --no-mmap --no-cache-prompt --port 8090`.

Run order. Gemma 4 first, server stopped, Qwen 3.6 second. Both runs same workstation, same day, no reboot. No background load.

Cross-check. The Qwen 3.6 baseline (~42 tok/s) matches the autoregressive baseline measured in the May 6 DFlash vs MTP article on the same workstation, confirming the bench harness is consistent across runs.
