FP4 Just Landed in llama.cpp: NVFP4 vs MXFP4 Explained (2026)

April 25, 2026 · by Mark Bartlett

Download this guide as PDF

> **Quick Answer:** FP4 inference is now in the GGUF ecosystem. NVFP4 (Nvidia's block-scaled FP4, GGML_TYPE_NVFP4 = 40) merged into llama.cpp through a series of PRs from late March through April 2026, with the Blackwell-native path reposted as PR #22196 on April 21. MXFP4 (the OCP open-standard variant) is in ik_llama.cpp, with the gguf-py constants merged back in November 2025 and kernels filling in since. Blackwell hardware (RTX 5090, RTX PRO 6000 Blackwell, B200, GB200) gets real tensor-core acceleration. Older cards run FP4 but only see the memory savings. Quality vs Q4_K_M is still being tested by the community — early days. Worth experimenting with on Qwen 3.6, DeepSeek V4-Flash, and large MoEs.

📚 **More on this topic:** llama.cpp vs Ollama vs vLLM · LLM Quantization Explained · Model Formats: GGUF, GPTQ, AWQ, EXL2 · Qwen 3.6 Complete Guide · RTX 5090 Local AI Benchmarks

FP4 in the GGUF ecosystem has been a “soon” story for over a year. As of April 25, 2026, it’s a “now” story. NVFP4 merged into llama.cpp in pieces from late March through April. MXFP4 is in ik_llama.cpp. Both formats are open. Both work today. The Blackwell-native path gives RTX 5090 and RTX PRO 6000 Blackwell users real hardware acceleration. Older cards run the same files but only collect the memory savings.

This is a release piece. The PRs landed in the last few days. Community benchmarks are arriving in a trickle. Most of what follows is what’s verifiable from the PRs and model uploads. The “is FP4 better than Q4_K_M?” question is genuinely open right now.

Image: NVFP4 and MXFP4 block-scaling diagram with Blackwell tensor core

---

What just landed

Two forks of llama.cpp picked up FP4 in parallel.

llama.cpp: NVFP4 (GGML_TYPE_NVFP4 = 40)

NVFP4 support arrived in llama.cpp through a sequence of pull requests from contributor `michaelw9999` and a few others, mostly between late March and late April 2026:

PR	What it added	Status
#20644	NVFP4 dp4a kernel (CUDA)	Merged Mar 26
#21074	Generic NVFP4 MMQ kernel	Merged Apr 1
#21095	Convert script for compressed-tensors NVFP4	Open
#21227	SYCL backend NVFP4 mul_mat	Merged Apr 1
#21455, #21539	Vulkan NVFP4 support	Merged Apr 10 / Apr 14
#21896	Blackwell-native NVFP4 (closed)	Closed Apr 21
#22196	Repost of #21896, Blackwell-native NVFP4	Open, Apr 21

The headline is the type ID itself. `GGML_TYPE_NVFP4 = 40` is now part of the ggml type table. That's what makes a quantization format real — convert scripts can target it, kernels dispatch to it, and downstream tools (Ollama, LM Studio, llama-server) inherit support.

Practically, three things changed at once:

- **Conversion**: you can convert a compressed-tensors NVFP4 model to GGUF. Most of the early uploads use Nvidia's `nvidia-modelopt` quantizer.
- **Inference kernels**: CUDA, SYCL, and Vulkan all have NVFP4 paths now.
- **Blackwell-native acceleration**: PR #22196 wires NVFP4 to Blackwell's FP4 tensor cores. That's the speed story.

ik_llama.cpp: MXFP4

`ik_llama.cpp` is Iwan Kawrakow's fork of llama.cpp focused on quantization research and unusual model formats. MXFP4 support there is older — PR #1007 added the gguf-py MXFP4 constants on November 24, 2025 — with kernels and surrounding plumbing filling in over the months since. The fork already had GPT-OSS MXFP4 support before mainline llama.cpp had any FP4 path at all.

If you're running MoE models on `ik_llama.cpp` for the fused-MoE optimizations, MXFP4 is now an option for storing those experts.

NVFP4 vs MXFP4: not the same thing

Both formats are 4-bit floats with block scaling. That's where the similarity ends.

NVFP4 is Nvidia's variant. Block size of 16. Two-level scaling: a per-block scale stored in FP8 (E4M3) plus a per-tensor scale in FP32. Designed against Blackwell's FP4 tensor cores. The two-level scale buys accuracy back in cases where a single shared exponent across a 32-element block doesn't capture enough of the dynamic range. Nvidia's bet is that the tensor-core hardware can absorb the cost of the second scale.

MXFP4 is the OCP Microscaling Formats standard. Block size of 32. Single-level scaling: one shared E8M0 exponent per block. Backed by AMD, Arm, Intel, Microsoft, Nvidia, Qualcomm, and Meta — multi-vendor by design. AMD MI300X, Intel Gaudi, and (per the spec) Nvidia's own Blackwell can all run it.

Both formats use the same E2M1 element layout for the 4-bit values themselves: 1 sign bit, 2 exponent bits, 1 mantissa bit. So the per-element representation is identical. The block-scaling math is what diverges.

They are not interchangeable. A GGUF quantized to NVFP4 is not the same file as one quantized to MXFP4. The kernels, the calibration, and the per-block metadata layout are all different.

Why FP4 matters for local AI

Three reasons it's worth caring about even before the benchmarks settle.

Memory. Four bits per weight is half the weight footprint of Q8 and roughly 25% smaller than Q4_K_M after metadata overhead is accounted for. A Qwen 3.6-27B that sits at ~17GB in Q4_K_M lands closer to ~14GB in NVFP4 — see the sakamakismile NVFP4 upload for a real example. For users who are bumping the ceiling of a 16GB or 24GB card, that's the difference between fitting context and offloading to RAM.

Bandwidth-bound speed gains on capable hardware. LLM inference on a single GPU is mostly memory-bandwidth-bound, not compute-bound. Smaller weights mean fewer bytes to read per token, which means faster tokens per second — even before tensor-core acceleration enters the picture. On Blackwell, the FP4 tensor cores stack a compute speedup on top of the bandwidth win.

MoE fit. For mixture-of-experts models, where the active path per token is small but the resident weight footprint is huge, FP4 is structurally a better match than Q8 or BF16. Qwen 3.6-35B-A3B,

GPT-OSS, and DeepSeek V4-Flash are all MoE designs where FP4 + ik_llama.cpp's fused-MoE kernels could be the new sweet spot for memory-constrained homelab setups. The qualifier "could" is doing work in that sentence — community testing this week, not yet a settled answer.

Hardware reality

This part matters. FP4 doesn't make every card faster. The acceleration depends on whether your tensor cores natively support FP4 ops.

Hardware	Native FP4?	What you get
Blackwell (RTX 5090, RTX PRO 6000 Blackwell, B200, GB200)	Yes — FP4 tensor cores	Memory savings + real compute speedup
Ada Lovelace (RTX 4090, 4080, 6000 Ada)	FP8 native, FP4 by emulation	Memory savings + small speedup
Ampere (RTX 3090, 3060, A100)	No FP4 hardware	Memory savings only, no compute win
AMD ROCm (MI300X, RX 7900 XTX)	MXFP4-aligned in spec; status varies	Watch the space
Apple Silicon	MLX has its own FP4 path, separate from llama.cpp	MLX route, not llama.cpp
Intel Gaudi / Arc	MXFP4-aligned via OCP	Watch the space

Practical translation. A 5090 owner is the target audience for NVFP4. A 3090 owner gets the storage win and bandwidth win, but no tensor-core acceleration on top. That's still useful — fitting a 35B model in 24GB at higher effective quality than Q3 is a real outcome — but it's not the marketing-deck speed multiplier. Don't expect a Q4_K_M → NVFP4 swap on Ampere to feel dramatically faster.

For AMD, the OCP-standard MXFP4 alignment is the more likely path. AMD has shipped MI300X with native MXFP4 support, and the broader ROCm tooling is moving toward the OCP standard. ik_llama.cpp's MXFP4 work is the more relevant fork to watch if you're on Radeon.

For Apple Silicon, the answer is different again — MLX has its own quantization stack, and MLX FP4 is not the same code path as llama.cpp's NVFP4 or MXFP4.

What to actually try this week

If you have Blackwell hardware and want to test NVFP4 today:

1. Pull the existing NVFP4 uploads.

sakamakismile/Qwen3.6-27B-Text-NVFP4-MTP is a clean example. Quantized with `nvidia-modelopt` 0.43.0 using NVFP4_DEFAULT_CFG. Around 14GB after quantization. Vision tower stripped, MTP speculative-decoding head restored. Built and tested on RTX PRO 6000 Blackwell (SM120). Compatible with RTX 5090 and other Blackwell cards with 14GB+ VRAM.

2. Build llama.cpp from source.

The Blackwell-native path is in PR #22196, which is open at time of writing. Mainline llama.cpp has the merged kernels (CUDA dp4a, MMQ, SYCL, Vulkan) but the Blackwell-specific tensor-core dispatch lands when #22196 merges. Build from a recent master to get the merged paths; check out the PR branch to test the Blackwell-native dispatch.

3. Wait, briefly, for the usual GGUF publishers.

Unsloth, bartowski, and mradermacher have all shipped Q-class GGUFs for Qwen 3.6 and DeepSeek V4. NVFP4 / MXFP4 variants from these maintainers are likely to land in days, not weeks. They tend to ship cleaner conversions with broader testing than first-pass uploads.

4. ik_llama.cpp + MXFP4 for MoE testing.

If you're already running `ik_llama.cpp` for fused-MoE on Qwen 3.6-35B-A3B or DeepSeek V4-Flash, the MXFP4 path is worth testing against your current Q4_K_M setup. Memory drop is real. Speed depends on your hardware.

5. Compare against your existing Q4_K_M / Q5_K_M GGUFs.

The honest test: pick a workload you actually care about — a coding agent loop, a long-context summarization, a multi-turn refactor. Run it on Q4_K_M and FP4 of the same model. Compare quality and tok/s. Don't lean on a single benchmark number from a single thread.

Honest caveats

A few things to keep in mind before getting too excited.

Quality vs Q4_K_M is unsettled. No independent benchmark suite has run NVFP4 GGUFs against Q4_K_M GGUFs on a representative model set yet. Early r/LocalLLaMA threads suggest the gap is small for big models and possibly worse than Q4_K_M for small ones — but those are reports, not measurements. The HuggingFace LLM Leaderboard, EvalPlus, and Aider benchmarks haven't run NVFP4 conversions yet. Until they do, treat FP4 as experimental.

Some quants will misbehave. When a new quantization format lands, certain models exhibit looping output, broken tool-calling, or degraded instruction-following until the calibration recipe is tuned. That's normal. Expect a few "this NVFP4 conversion is broken, use that one instead" posts over the next two weeks as the community separates good uploads from bad.

Tooling around FP4 is new. Things that are mature for Q4_K_M — KV cache quantization, draft models for speculative decoding, prefix-caching corner cases — may have rough edges with FP4. The sakamakismile upload restores MTP speculative decoding specifically because the original NVFP4 conversion dropped it (0% draft acceptance until they grafted the MTP head back in).

Ollama and LM Studio integration is downstream. llama.cpp adding the type ID is the prerequisite for these tools picking it up. Expect a delay before `ollama pull qwen3.6:27b-nvfp4` is a thing. For now, llama-server or `vllm serve` are the practical paths.

The convert script for compressed-tensors NVFP4 (PR #21095) is still open. If you want to convert an arbitrary HF model to NVFP4 GGUF yourself today, you're checking out a branch, not running a released tool.

The bottom line

If you have a Blackwell GPU — RTX 5090, RTX PRO 6000 Blackwell, B200, GB200 — experiment. This is the format your hardware was built for. The acceleration is real, the memory savings are real, and the early uploads are usable today.

If you have Ada Lovelace (4090, 4080, 6000 Ada) — FP4 is mostly a memory story. Small bandwidth speedup, no native tensor-core win. Worth running if you need the headroom.

If you have Ampere (3090, 3060, A100) — pure memory savings. Run NVFP4 only if you're trying to fit a model that won't fit at Q4_K_M.

If you're on AMD or Apple — watch the space. MXFP4 in ik_llama.cpp is the right fork to track for Radeon. MLX has its own FP4 story that is not llama.cpp's story.

The FP4 release itself is news. The “is it better than Q4_K_M for my workload” question is not yet news. Independent benchmarks will land over the next two to three weeks. Current enthusiasm is grounded — the PRs are real, the Blackwell hardware fit is real, the memory math works. But “grounded” is not the same as “validated.” Try it on your own work before declaring victory.

Related Guides

- llama.cpp vs Ollama vs vLLM
- LLM Quantization Explained
- Model Formats: GGUF, GPTQ, AWQ, EXL2
- Qwen 3.6 Complete Guide
- DeepSeek V4 Flash vs Pro
- RTX 5090 Local AI Benchmarks
- TurboQuant: KV Cache Compression

Sources: llama.cpp PR #22196 — Blackwell-native NVFP4, PR #21896, PR #21074, PR #20644, PR #21095, PR #21455, PR #21539, ik_llama.cpp PR #1007 — MXFP4 in gguf-py, OCP Microscaling Formats v1.0, sakamakismile/Qwen3.6-27B-Text-NVFP4-MTP, Nvidia ModelOpt

Get notified when we publish new guides.

Subscribe — free, no spam
