llama.cpp vs Ollama vs vLLM: When to Use Each

February 3, 2026 · by Mark Bartlett

Download this guide as PDF

> **Quick Answer:** Ollama for easy local use — it's llama.cpp with a friendly wrapper, handles model management, and just works. llama.cpp directly for maximum control, CPU inference, or when you need features Ollama hasn't exposed yet. vLLM for production serving or when you need to handle multiple users — it's 23% faster than llama.cpp at 16 concurrent requests and scales to hundreds of users with PagedAttention. For multi-GPU setups, skip llama.cpp/Ollama entirely and use vLLM or ExLlamaV2. The common mistake is using Ollama for production APIs — it works, but vLLM will handle 4x the load on the same hardware.

📚 **More on this topic:** Ollama Troubleshooting Guide · Run Your First Local LLM · VRAM Requirements · Open WebUI Setup · Planning Tool

Three tools dominate local LLM inference: llama.cpp, Ollama, and vLLM. They solve different problems, and picking the wrong one either wastes your hardware or makes your life harder than it needs to be.

Here's the short version: Ollama is llama.cpp with training wheels. vLLM is for serving models to multiple users. llama.cpp is the engine under Ollama's hood that you can also drive directly.

This guide covers when each one makes sense, with actual benchmarks instead of vibes.

What Each Tool Actually Is

llama.cpp

The original. Georgi Gerganov wrote it to run LLaMA on a MacBook, and it's become the de facto standard for running quantized models on consumer hardware. Written in C++, it handles GGUF model files and runs on basically anything — CPUs, NVIDIA GPUs, AMD GPUs, Apple Silicon, even phones.

Key insight: llama.cpp is an inference engine, not a user-facing application. You interact with it via command line or build applications on top of it.

Ollama

A friendly wrapper around llama.cpp. It handles model downloads, versioning, and provides a simple CLI and REST API. When you run `ollama run llama3`, it's using llama.cpp underneath to actually do the inference.

Key insight: Ollama adds convenience, not performance. The inference speed is llama.cpp speed with a small overhead from the Go-based server layer.

vLLM

A production inference engine built for serving. Uses PagedAttention to efficiently handle multiple concurrent requests, tensor parallelism for multi-GPU setups, and continuous batching to maximize throughput. Written in Python, optimized for NVIDIA GPUs.

Key insight: vLLM is designed for "many users, one server" — the opposite of the desktop use case Ollama targets.

Head-to-Head Comparison

Feature	llama.cpp	Ollama	vLLM
Primary use	Engine/library	Desktop local AI	Production serving
Setup difficulty	Medium	Easy	Medium-Hard
Model format	GGUF	GGUF (via llama.cpp)	HuggingFace, GPTQ, AWQ, GGUF
Single-user speed	Fastest	~Same as llama.cpp	Slightly slower
Multi-user throughput	Poor	Poor	Excellent
Multi-GPU support	Limited	Limited	Excellent
CPU inference	Yes	Yes	Limited
Memory efficiency	Good	Good	Best (PagedAttention)
API	HTTP server optional	REST API built-in	OpenAI-compatible
Model management	Manual	Automatic	Manual

Performance Benchmarks

Real numbers from a Qwen 2.5 3B on RTX 4090:

Single User (You Alone)

Scenario	llama.cpp	vLLM	Winner
2K prompt + 256 gen	90.0s	94.4s	llama.cpp (4.7% faster)
30K prompt + 256 gen	231.6s	245.8s	llama.cpp (5.8% faster)

For single-user desktop use, llama.cpp (and therefore Ollama) is slightly faster. The difference is small — under 6% — but it's real.

Multiple Users (Concurrent Requests)

Scenario	llama.cpp	vLLM	Winner
16 concurrent, 2K prompt	265.5s	215.3s	vLLM (23% faster)
16 concurrent, 24K prompt	3640.7s	3285.3s	vLLM (11% faster)

Once you have multiple users hitting the same model, vLLM pulls ahead significantly. At 16 concurrent requests, it's 23% faster. At higher concurrency, the gap widens further.

The PagedAttention Advantage

vLLM's secret weapon is PagedAttention, which eliminates 60-80% of the memory waste from KV cache fragmentation. In practice:

- Standard implementation on 24GB VRAM: ~32 concurrent sequences
- vLLM with PagedAttention: **~128 concurrent sequences**

Same hardware, 4x the capacity. That's why production deployments use vLLM.

Ollama: When to Use It

Use Ollama when:

- You're running models for yourself on your own machine

- You want model management handled automatically
- You're building local apps that need an LLM API
- You don't want to think about quantization formats or compile flags
- You're using Open WebUI or similar frontends

Don't use Ollama when:

- You're serving multiple users simultaneously
- You need multi-GPU tensor parallelism
- You need maximum control over inference parameters
- You're building a production API

Ollama's Real Value

Ollama's killer feature isn't performance — it's convenience. Compare the workflows:

With Ollama:

```
ollama run qwen3:8b
```

With llama.cpp directly:

```
# Find and download GGUF file manually
# Figure out the right quantization
./llama-cli -m ./models/qwen3-8b-q4_k_m.gguf \
  -c 8192 -n 256 --temp 0.7 -p "Your prompt here"
```

Ollama handles model discovery, downloads, versioning, and provides a consistent interface. For desktop use, that convenience is worth the tiny overhead.

Ollama Limitations

- **No tensor parallelism:** Multi-GPU support exists but doesn't split models across GPUs efficiently
- **Limited quantization control:** You get what's in the Ollama library, can't easily use custom quants
- **Server overhead:** The Go layer adds some latency, noticeable at very high request rates

- **Batching:** Handles concurrent requests poorly compared to vLLM

llama.cpp: When to Use It Directly

Use llama.cpp when:

- You need CPU inference or heavy CPU offloading
- You want maximum control over quantization and inference settings
- You're building something that needs to embed inference directly
- You're on unusual hardware (ARM, older GPUs, etc.)
- You need features Ollama hasn't exposed yet

Don't use llama.cpp when:

- You just want to chat with a model (use Ollama)
- You need high-concurrency serving (use vLLM)
- You're doing multi-GPU inference (use vLLM or ExLlamaV2)

llama.cpp's Unique Strengths

CPU offloading — llama.cpp is the only tool that gracefully handles models too large for your VRAM by offloading layers to system RAM. Yes, it's slow (~1 tok/s for huge models), but it works. vLLM can't do this at all.

Hardware compatibility — Runs on NVIDIA, AMD, Intel, Apple Silicon, and even pure CPU. If you have unusual hardware, llama.cpp probably supports it.

Quantization ecosystem — The GGUF format and llama.cpp's quantization tools are the standard. K-quants (Q4_K_M, Q5_K_M) and I-quants give you fine-grained control over the size/quality tradeoff:

Quant	Bits/Weight	Perplexity Impact	Use Case
Q4_K_M	~4.5 bpw	+0.05 ppl	Default recommendation
Q5_K_M	~5.3 bpw	+0.01 ppl	Quality sweet spot
Q3_K_M	~3.7 bpw	+0.66 ppl	VRAM constrained
Q6_K	~6.0 bpw	+0.004 ppl	Near-lossless
IQ2_XS	~2.3 bpw	Higher	Extreme compression

Speculative decoding — Use a small draft model to speed up generation. Reports of ~12 tok/s vs 8-9 tok/s baseline with good draft model matches.

Basic llama.cpp Usage

```
# Run inference
./llama-cli -m model.gguf -p "Your prompt" -n 256

# Start a server
./llama-server -m model.gguf -c 4096 --host 0.0.0.0 --port 8080

# With GPU layers (offload 35 layers to GPU)
./llama-cli -m model.gguf -ngl 35 -p "Your prompt"

# Enable flash attention
./llama-cli -m model.gguf -fa -p "Your prompt"
```

---

vLLM: When to Use It

Use vLLM when:

- You’re serving a model to multiple users
- You need an OpenAI-compatible API
- You have multi-GPU setups and want tensor parallelism
- Throughput and cost efficiency matter more than setup simplicity
- You’re running a production inference service

Don’t use vLLM when:

- You’re the only user (Ollama is simpler)
- You need CPU inference or offloading
- You’re on non-NVIDIA hardware (limited support)
- You want the simplest possible setup

vLLM’s Production Numbers

The throughput improvements are dramatic:

- **14-24x higher throughput** vs standard HuggingFace Transformers
- **3-10x improvement** from continuous batching alone
- Stripe reduced inference costs by **73%** using vLLM

Concrete example: Serving Mixtral-8x7B on 2x A100 at 100 requests/second:

- vLLM P50 latency: 180ms
- Standard serving P50 latency: 650ms

At scale, vLLM isn’t optional — it’s required to make the math work.

vLLM Setup

```
# Install
pip install vllm

# Start server
vllm serve meta-llama/Llama-3.1-8B-Instruct

# With tensor parallelism (multi-GPU)
vllm serve meta-llama/Llama-3.1-70B-Instruct --tensor-parallel-size 4

# With quantization
vllm serve model-name --quantization awq
```

The API is OpenAI-compatible, so existing code that calls OpenAI can point at your vLLM server instead:

```
from openai import OpenAI

client = OpenAI(base_url="http://localhost:8000/v1", api_key="none")
response = client.chat.completions.create(
    model="meta-llama/Llama-3.1-8B-Instruct",
    messages=[{"role": "user", "content": "Hello!"}]
)
```

vLLM Hardware Requirements

vLLM is GPU-hungry and primarily targets NVIDIA:

Model Size	Minimum VRAM	Recommended
7B	16 GB	24 GB
13B	24 GB	40 GB
30B+	40 GB+	80 GB+

For consumer GPUs, an RTX 3090 or 4090 can run 7-13B models in vLLM. Larger models need professional cards or multi-GPU setups.

Multi-GPU: Critical Guidance

Do NOT use llama.cpp or Ollama for multi-GPU inference.

This is the most common mistake. llama.cpp's multi-GPU support exists but isn't optimized for tensor parallelism. Benchmark data shows it performs poorly compared to purpose-built solutions.

For multi-GPU setups:

- **vLLM** — Best for FP16/BF16 models, excellent tensor parallelism
- **ExLlamaV2** — Best for quantized models (EXL2 format), good multi-GPU support

If you have 2+ GPUs and want to run large models across them, skip Ollama entirely.

Other Tools Worth Knowing

ExLlamaV2

The speed king for quantized inference on NVIDIA GPUs. Uses EXL2 format (similar to GPTQ but better quality). Benchmark: 120-150 tok/s on RTX 4090 for 13B models vs 80-100 tok/s for llama.cpp.

Use when: You need maximum speed for quantized models on NVIDIA, especially multi-GPU.

Skip when: You need CPU offloading or non-NVIDIA hardware.

kobold.cpp

A fork of llama.cpp focused on creative writing and roleplay. Adds features like soft prompts, memory management, and UI tailored for story generation.

Use when: Fiction writing, roleplay, story continuation.

text-generation-inference (TGI)

HuggingFace's production inference server. Similar to vLLM in goals, different implementation. Good HuggingFace integration.

Use when: You're already deep in the HuggingFace ecosystem.

Decision Flowchart

Are you the only user?

- Yes → **Ollama** (or llama.cpp if you need more control)
- No → Continue

Do you have multiple GPUs you want to use together?

- Yes → **vLLM** (or ExLlamaV2 for quantized models)
- No → Continue

Are you serving >5 concurrent users?

- Yes → **vLLM**
- No → **Ollama** is probably fine

Do you need CPU inference or offloading?

- Yes → **llama.cpp** (only real option)
- No → See above

Are you building a production API?

- Yes → **vLLM**
- No → **Ollama**

Using Multiple Tools Together

You don't have to pick just one. A common setup:

1. **Ollama for development** — Quick testing, trying new models, local experimentation
2. **vLLM for production** — Serving the final model to users

Or for power users:

1. **Ollama for daily use** — Chat, quick queries, Open WebUI
2. **llama.cpp directly** — When you need specific quantization or settings Ollama doesn't expose

Ollama → llama.cpp Migration

If you've been using Ollama and want to try llama.cpp directly, your models are already downloaded. Find them at:

- macOS: `~/.ollama/models/`
- Linux: `~/.ollama/models/` or `/usr/share/ollama/.ollama/models/`
- Windows: `C:\Users\<user>\.ollama\models\`

The actual GGUF files are in `blobs/` with hash names. You can use them directly with llama.cpp.

Bottom Line

The choice is simpler than it looks:

Ollama — You're running models locally for yourself. It just works. Start here.

llama.cpp — You need control Ollama doesn't give you, or you need CPU inference/offloading. Power user territory.

vLLM — You're serving models to other people, or you have multi-GPU setups. Production territory.

The mistake to avoid: using Ollama for production APIs when vLLM would handle 4x the load. Ollama is fantastic for what it's designed for — desktop local AI. It's not designed for serving hundreds of concurrent users, and it shows in benchmarks.

For most readers of this site — hobbyists running models on their own hardware — Ollama is the right answer. You'll know when you've outgrown it.

```
# Start here
ollama run qwen3:8b

# Graduate to this when you need to
vllm serve Qwen/Qwen3-8B-Instruct
```

Source: https://insiderllm.com/guides/llamacpp-vs-ollama-vs-vllm/

Free guides for running AI locally