How to run an LLM on your own server: Ollama, llama.cpp and vLLM compared

26 October 2026 · 7 min read · blogs.iictp.org/run-an-llm-on-your-own-server

Running a language model on hardware you control means prompts and documents never leave your building, the cost is the hardware rather than a meter, and the system keeps working offline. It also means you own sizing, updates and failure. This guide covers the three tools most people start with and how to choose between them.

TL;DR The short version

- Running a model on your own hardware keeps prompts and documents in-house and works offline, but you own sizing, updates and failure.
- Size by memory first, then compare the three usual starting points: Ollama, llama.cpp and vLLM.
- Try the smallest model that might do the task on a workstation and measure quality on your own examples, not a leaderboard.

Read online, listen to it, or change the text size: https://blogs.iictp.org/run-an-llm-on-your-own-server

Start with memory, not the tool

A model is mostly a large table of numbers called weights. The memory they need is roughly the number of parameters times the bytes stored per weight. A 7-billion-parameter model at 16-bit precision needs about 14 GB for weights alone. At 4-bit precision it needs about 3.5 GB. Add room for the context (the conversation the model is holding) and for the runtime itself.

So the first question is never which tool, but whether the model you want fits in the memory you have. Quantization, which stores weights at lower precision, is how large models are made to fit. Formats such as GGUF carry quantized weights in one file.

The three starting points

TOOL	WHAT IT IS	BEST FOR	WATCH OUT FOR
Ollama	A packaged runner: one command downloads a model and serves it, with a local HTTP API.	A laptop or single workstation; trying models quickly; developers building a prototype.	Less control over scheduling and batching than a dedicated serving engine.
llama.cpp	A C/C++ inference engine that reads GGUF files and runs on CPU, on GPU, and on Apple Silicon.	Constrained or mixed hardware; fine control over quantization and offloading.	You assemble more of the surrounding system yourself.

TOOL	WHAT IT IS	BEST FOR	WATCH OUT FOR
vLLM	A serving engine for GPUs, built for many concurrent requests, with an OpenAI-compatible API.	A shared team service on server GPUs where throughput matters.	It expects a capable GPU; it is not the tool for a laptop.

Choose by hardware and by how many people will use it at once.

A sensible path

1. Write down the task and the smallest model that might do it. Smaller models are cheaper to run and often enough.
2. Try it on a workstation with Ollama or llama.cpp. Measure the answer quality on your own examples, not on a leaderboard.
3. Decide how many people will use it at once. One user and fifty users are different engineering problems.
4. If many users share it, move to a serving engine such as vLLM on server GPUs, and put it behind your normal authentication and logging.
5. Plan updates: models, runtimes and drivers all change, and a local deployment is yours to patch.

What “private” does and does not cover

Keeping inference local removes the third-party provider from the data path. It does not by itself secure the system. You still need access control, logging, network isolation and a plan for what the model is allowed to read. Treat the model server like any other service that touches sensitive data.

> This is the practical ground covered by the Local LLM Practitioner credential. See the exam facts for what each credential tests.

Sources and further reading

1. llama.cpp project. github.com/ggml-org/llama.cpp. Read 26 October 2026.
2. vLLM project documentation. docs.vllm.ai. Read 26 October 2026.
3. Ollama documentation. ollama.com. Read 26 October 2026.

Published by IICTP, an independent technical credential body, on 26 October 2026. The current version of this post is at https://blogs.iictp.org/run-an-llm-on-your-own-server. More guides: blogs.iictp.org.