**Multi-Model LLM Serving: vLLM, Ollama, and Other Approaches**

**📌 Objective**

The main goal of this exploration is to achieve **multi-model LLM serving**—particularly with **vLLM**—and to investigate other tools like **Ollama** and **llama.cpp** for comparison. This write-up documents the approach, technical constraints, and learnings associated with deploying and serving LLMs efficiently using these frameworks.

**⚙️ vLLM: Multi-Model Serving Setup**

To serve models using **vLLM**, either of the following commands can be used:

vllm serve facebook/opt-125m

or

python -m vllm.entrypoints.openai.api\_server --model <model\_name>

* The <model\_name> must refer to a **Hugging Face model repository** or a **local path to a downloaded model**.
* On running either command, the model is served through an **OpenAI-compatible endpoint**.

**❗ Multi-Model Limitation in vLLM**

* You **cannot serve multiple models simultaneously in a single vLLM instance**. Only **one model** can be specified per server instance.
* To serve multiple models concurrently:
  + Run **multiple vLLM servers** with different model names.
  + Each instance must be on a **separate port**.
  + In a containerized setup, you create **separate containers**, each serving a different model.

This approach ensures **parallel model serving** and allows concurrent request handling by mapping traffic via different ports. This setup also achieves **concurrency** since multiple requests can be handled simultaneously by each model instance.

**🐘 Ollama: An Alternative Model Server**

**✅ Installation**

To install **Ollama** (inside WSL or any Linux shell):

curl -fsSL https://ollama.com/install.sh | sh

**✅ Running a Model**

To serve a model with Ollama:

ollama run <model\_name>

**⚠️ Model Source Restriction**

* The model name **must exist** in Ollama’s internal model registry.
* Ollama uses a **proprietary model format** based on **GGUF**.
* Hugging Face models in formats like .bin, .safetensors, .pt, etc., **cannot be used directly**.
* Therefore, custom Hugging Face models must be **converted** into the GGUF format first.

**📁 Hugging Face to GGUF: Conversion Workflow**

1. **Install Hugging Face CLI**:

pip install -U "huggingface\_hub[cli]"

pip install huggingface[hf\_transfer]

1. **Set environment variable**:

export HF\_HUB\_ENABLE\_HF\_TRANSFER=1

1. **Download GGUF-compatible version (if available)**:

huggingface-cli download <model\_name> <version>

1. **Write a Modelfile** (similar to a Dockerfile for Ollama):

* This defines how the model is constructed.

⚠️ Important: Ensure the file is named Modelfile (no .txt extension).

1. **Create the model**:

ollama create <custom\_model\_name> -f Modelfile

1. **Run the model**:

ollama run <custom\_model\_name>

1. **Verify installed models**:

ollama list

**❌ Issue with OPT Models (e.g., facebook/opt-125m)**

* Attempted to convert facebook/opt-125m to GGUF using convert\_hf\_to\_gguf.py from llama.cpp.
* The conversion failed with the following error:

ERROR:hf-to-gguf:Model OPTForCausalLM is not supported

Meta’s OPT models are not yet supported for GGUF conversion.

* Even smaller models like tiny-gpt2 are currently **not supported** for GGUF conversion using llama.cpp.

**🧰 llama.cpp: A Foundation for GGUF and Quantization**

* **llama.cpp** is a C/C++ open-source project by **Georgi Gerganov**.
* It powers many backends, including Ollama.
* Provides:
  1. **Quantization** (1.5, 2, 3, 4-bit) — helpful for low-resource CPUs.
  2. **GGUF format support** — the format required by Ollama.

**Summary:**

LLMs exist in two primary formats:

* **Baseline** (standard weights: .bin, .pt, .safetensors)
* **GGUF** (quantized, optimized format used by llama.cpp and Ollama)

**🔁 Multi-Model Serving in Ollama**

* Like vLLM, **Ollama can only serve one model per terminal instance**.
* To serve multiple models:
  + Run multiple ollama run <model\_name> commands in **separate terminals** or **containers**.
  + Each instance will serve a different model.

**❗ Limitation:**

* Ollama’s model registry is limited.
* You cannot use raw Hugging Face models unless you **convert** them to GGUF **and** ensure they are of a **supported architecture**.

**🔄 Comparison: vLLM vs. Ollama**

|  |  |  |
| --- | --- | --- |
| Feature | vLLM | Ollama |
| Supported Models | Any Hugging Face model (bin, pt) | Limited to GGUF-supported models |
| Model Format | Native HF / Local | Requires GGUF format |
| Multi-model Serving | Multiple containers on different ports | Multiple terminals/containers |
| Concurrency Support | Yes (natively with async support) | Yes, terminal-based |
| Containerization | Fully Dockerized | CLI-based (Docker optional) |
| Model Customization | Open via Hugging Face | Requires GGUF + Modelfile |
| Quantization | Not built-in | Built-in (via llama.cpp) |
| GPU Utilization Control | Fine-grained (CUDA Graphs, memory flags) | Less control |

**Conclusion**: While Ollama is good for quick setups and small-scale inference with limited model support, **vLLM is far more flexible, performant, and production-ready**—especially for **multi-model deployments and GPU concurrency.**

**🧠 Other LLM Inference Server Alternatives**

Here are additional inference-serving frameworks commonly used for LLMs:

1. **vLLM** — High-throughput, OpenAI-compatible, GPU-optimized
2. **Triton Inference Server** (NVIDIA)
3. **TensorRT-LLM** — Fast inference using TensorRT graph optimizations
4. **DeepSpeed-MII** — Microsoft’s high-performance inference and fine-tuning engine
5. **HuggingFace TGI** (Text Generation Inference)
6. **llama.cpp / GGUF** — Lightweight, quantized CPU/GPU inference
7. **Ray Serve** — General-purpose distributed serving (honorable mention)

**✅ Final Takeaways**

* **vLLM** remains one of the best solutions for deploying Hugging Face models in production at scale.
* **Ollama** and **llama.cpp** are promising for **low-resource environments** and quick deployments—but lack the flexibility and broad model compatibility of vLLM.
* To use Ollama with Hugging Face models, GGUF conversion is required—and only a few architectures (like LLaMA) are supported at the moment.
* The journey highlights the growing ecosystem of **LLM-serving frameworks**, each suited for different resource, performance, and infrastructure needs.

**📊 Comparison Table**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Feature / Framework** | **vLLM** | **Triton** | **TensorRT-LLM** | **DeepSpeed-MII** | **HF TGI** | **llama.cpp** | **Ray Serve** |
| ✅ **Model Format Required** | HuggingFace Transformers | ONNX, TorchScript, TF, etc. | Custom engine (.plan) | HuggingFace Transformers | HF Transformers | GGUF (quantized) | Any Python model |
| 🤖 **Multi-Model Support** | ❌ No | ✅ Yes | ⚠️ Partial (manual routing) | ⚠️ Partial (manual deploys) | ⚠️ Experimental | ❌ One at a time | ✅ Yes |
| ⚙️ **Supports Batching** | ✅ Yes (smart batching) | ✅ Yes (dynamic/static) | ✅ Yes | ✅ Yes | ✅ Yes | ❌ No | ✅ Yes |
| 🔁 **Concurrency Support** | ✅ Token-level concurrency | ✅ Full | ✅ Engine-level | ✅ Thread-based | ✅ | ❌ Limited | ✅ Yes |
| 🔌 **OpenAI API Compatible** | ✅ Yes | ❌ (requires proxy) | ⚠️ Not directly | ❌ (custom API) | ✅ Yes | ⚠️ Experimental | ❌ (custom needed) |
| 🚀 **Command to Serve** | vllm.entrypoints... | tritonserver --repo=... | trtllm-server --engine=... | mii.serve(...) | text-generation-launcher | ./server -m model.gguf | serve.run(...) |
| 🧪 **How to Test** | OpenAI API / curl | Triton client / curl | gRPC/HTTP client | Python client (mii) | OpenAI-style API | curl / Python bindings | HTTP/REST |
| 🧠 **Quantization Support** | ⚠️ Partial (AWQ/WIP) | ✅ INT8, FP16 | ✅ INT4, INT8 | ✅ Bitsandbytes | ✅ bitsandbytes | ✅ 4/5/6-bit GGUF | ⚠️ Manual |
| 📦 **Install Size / Ease** | ✅ Easy (pip install) | ⚠️ Heavy (docker preferred) | ⚠️ Hard (engine build needed) | ⚠️ Code setup needed | ✅ Easy via HF CLI | ✅ Small binary | ✅ Python only |
| 📁 **Model Sources** | HuggingFace only | Any format | HuggingFace → TensorRT | HuggingFace | HuggingFace | GGUF models only | HuggingFace / Custom |
| 🔥 **Performance (GPU)** | ⚡⚡ Very Fast | ⚡ Depends on backend | 🔥 Fastest (H100 optimized) | ⚡ Fast | ⚡ Medium-fast | ❌ CPU-bound | ⚠️ Depends |
| 📍 **Offline / CPU Support** | ❌ GPU only | ⚠️ Limited CPU support | ❌ GPU only | ⚠️ GPU preferred | ⚠️ Limited CPU | ✅ Yes (CPU/edge-ready) | ✅ Yes |
| 🧩 **Extensibility** | ❌ Limited | ✅ Highly extensible | ❌ Closed engine | ⚠️ Needs code changes | ⚠️ Not modular | ⚠️ Custom builds for changes | ✅ Full Python control |