How to Self-Host LLMs on Your Infrastructure

Ka‘imi Kahihikolo

Intro

Why Self-Host?

- Data security
- Cost limitations
- **You're a massive nerd**

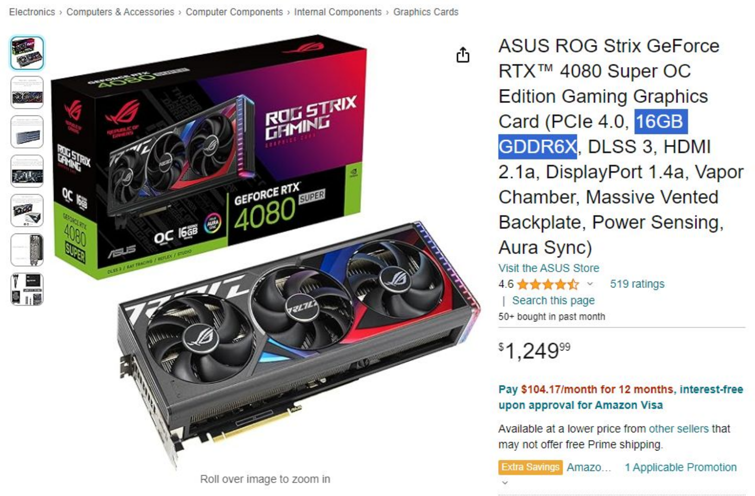
Things to consider:

- Do you have enough compute?
  - In particular, GPU vRAM (or graphics RAM)
- Understand the goals/limitations of your project.
  - Number of requests, frequency, and size (tokens in/out)
  - Observability/auditing requirements

Rough number crunching

- Mistral-7B: 7 Billion Parameters
- Each parameter stored as 16-bit float
- 16 bits = 2 bytes
- 7 Billion x 2 bytes = 14 Billion bytes = 14 Gigabytes

+10% overhead →15.4 GB of vRAM minimum


Electronics › Computers & Accessories › Computer Components › Internal Components › Graphics Cards

ASUS ROG Strix GeForce RTX™ 4080 Super OC Edition Gaming Graphics Card (PCIe 4.0, 16GB GDDR6X, DLSS 3, HDMI 2.1a, DisplayPort 1.4a, Vapor Chamber, Massive Vented Backplate, Power Sensing, Aura Sync)

Visit the ASUS Store

4.6 ★★★★☆ 519 ratings

| Search this page

50+ bought in past month

$1,249.99

Pay $104.17/month for 12 months, interest-free upon approval for Amazon Visa

Available at a lower price from other sellers that may not offer free Prime shipping.

Extra Savings Amazo... 1 Applicable Promotion

Roll over image to zoom in

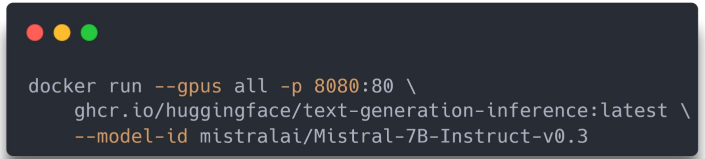
Text Generation Inference (TGI)

➔ **Cool Features**
- Easy to use
- Direct integration with LangChain & LlamaIndex

➔ **Advantages**
- Native support for models from HuggingFace
- Numerous parameters to control inference (quantization, tensor parallelism, etc)
- All dependencies in a single Docker image

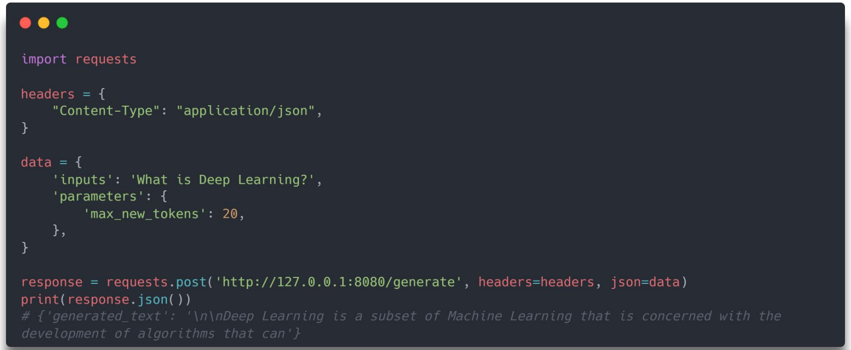
➔ **Limitations**
- Non-standard API
- Scalability

LAUNCH



```
docker run --gpus all -p 8080:80 \
    ghcr.io/huggingface/text-generation-inference:latest \
    --model-id mistralai/Mistral-7B-Instruct-v0.3
```

CONSUME



```python
import requests

headers = {
    "Content-Type": "application/json",
}

data = {
    'inputs': 'What is Deep Learning?',
    'parameters': {
        'max_new_tokens': 20,
    },
}

response = requests.post('http://127.0.0.1:8080/generate', headers=headers, json=data)
print(response.json())
# {'generated_text': '\n\nDeep Learning is a subset of Machine Learning that is concerned with the
development of algorithms that can'}
```

See More: https://huggingface.co/docs/text-generation-inference/

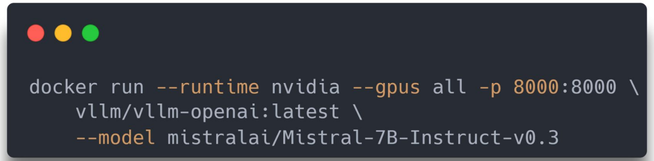
vLLM

➔ **Cool Features**
- Continuous request batching
- Fast and easy to use
- Direct integration with LangChain & LlamaIndex

➔ **Advantages**
- Very fast text generation
- OpenAI compatible server

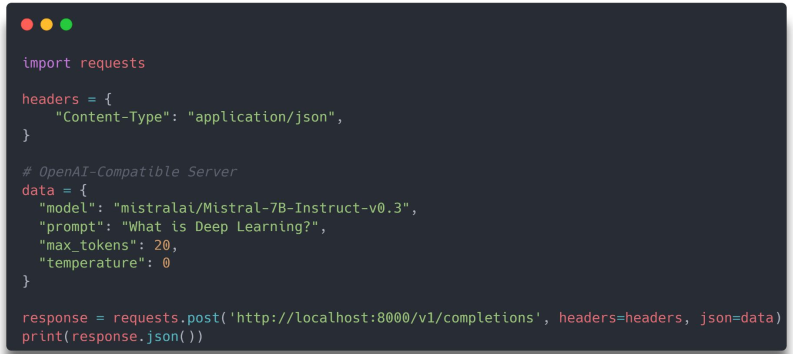
➔ **Limitations**
- Limited, but growing, list of supported models.
- Harder to add custom models.
- Some quantization support
- Scalability

LAUNCH



```
docker run --runtime nvidia --gpus all -p 8000:8000 \
    vllm/vllm-openai:latest \
    --model mistralai/Mistral-7B-Instruct-v0.3
```

CONSUME



```
import requests

headers = {
    "Content-Type": "application/json",
}

# OpenAI-Compatible Server
data = {
  "model": "mistralai/Mistral-7B-Instruct-v0.3",
  "prompt": "What is Deep Learning?",
  "max_tokens": 20,
  "temperature": 0
}

response = requests.post('http://localhost:8000/v1/completions', headers=headers, json=data)
print(response.json())
```

See More: https://docs.vllm.ai/en/stable/index.html

Ray Serve

➔ **Cool Features**

- Monitoring/observability with Prometheus/Grafana
- Autoscaling across replicas w/ zero downtime
- Fractional GPU support

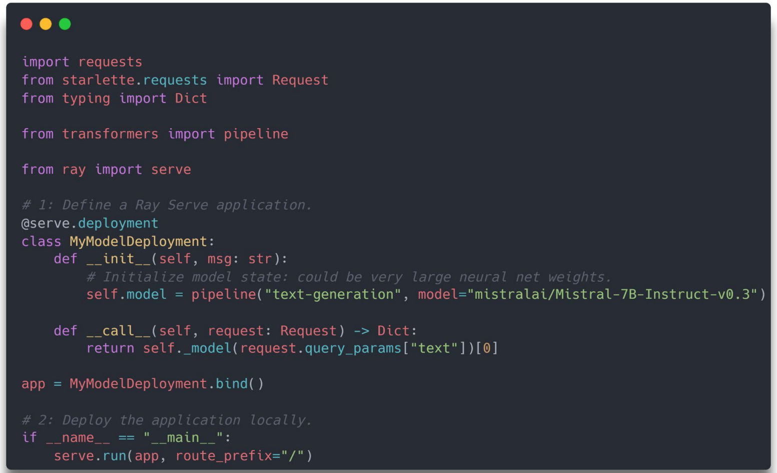
➔ **Advantages**

- Framework/vendor agnostic and flexible
- Integration with Ray ecosystem - most mature framework.
- Extensive documentation/examples

➔ **Limitations**

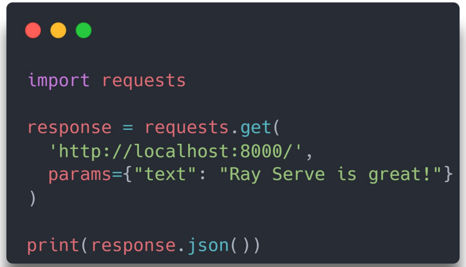
- Require deploying/managing a Ray Cluster
- More code than a single Docker command
- Not focused on LLMs

LAUNCH



```python
import requests
from starlette.requests import Request
from typing import Dict

from transformers import pipeline

from ray import serve

# 1: Define a Ray Serve application.
@serve.deployment
class MyModelDeployment:
    def __init__(self, msg: str):
        # Initialize model state: could be very large neural net weights.
        self.model = pipeline("text-generation", model="mistralai/Mistral-7B-Instruct-v0.3")

    def __call__(self, request: Request) -> Dict:
        return self._model(request.query_params["text"])[0]

app = MyModelDeployment.bind()

# 2: Deploy the application locally.
if __name__ == "__main__":
    serve.run(app, route_prefix="/")
```

CONSUME



```python
import requests

response = requests.get(
  'http://localhost:8000/',
  params={"text": "Ray Serve is great!"}
)

print(response.json())
```

See More: https://docs.ray.io/en/latest/serve/

TL;DR

1. **vLLM**: when you care about speed and the model you want is supported.
2. **TGI**: for native HuggingFace support; great for rapid prototyping
3. **Ray Serve**: for mature production-ready projects.

My favorite option: ***Ray Serve + vLLM together!***

Honorable Mentions:

4. **MLC LLM**: Natively deploy LLMs on the edge (ie Android or iPhone)
5. **CTranslate2**: C++ and Python library for efficient execution
6. **LocalAI**: OpenAI compatible framework optimized for a collection text, audio, and image models