# Deploying Meta's Llama 3.2 with vLLM and Docker: A Step-by-Step Guide

In this guide, I'll walk you through the process of deploying Meta's Llama 3.2 language model using vLLM (a high-performance inference engine) and Docker with GPU acceleration. This setup allows you to run your own local AI inference server with OpenAI-compatible endpoints.

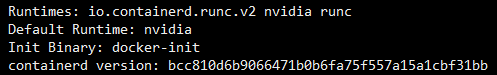
## Prerequisites

Before we begin, you'll need:

* A machine with NVIDIA GPU(s)
* Docker installed
* NVIDIA Container Toolkit (nvidia-docker) configured
* A Hugging Face account

### Step 1: Configure NVIDIA Docker Support

First, ensure your system is configured to use GPUs within Docker containers. The NVIDIA Container Toolkit (nvidia-docker) provides this functionality. If you haven't set this up yet, you'll need to install the NVIDIA Container Toolkit and configure Docker to use the NVIDIA runtime. This allows Docker containers to access your GPU(s) for acceleration.



### Step 2: Pull the vLLM Docker Image

vLLM provides optimized Docker images for running language models. Pull the latest vLLM OpenAIcompatible image with:

$ docker pull vllm/vllm-openai

This image includes the vLLM inference engine with an API compatible with OpenAI's endpoints.

### Step 3: Obtain a Hugging Face API Token

To download models from Hugging Face, you'll need an API token:

1. Go to: <https://huggingface.co/settings/tokens>
2. Create a new token with write permissions
3. Copy the token for use in the next steps

This token allows the container to authenticate with Hugging Face and download the model files.

### Step 4: Accept Terms and Conditions for Llama 3.2

Meta's Llama 3.2 models are "gated," meaning you need to accept their usage terms before

downloading:

1. Visit: <https://huggingface.co/settings/gated-repos>
2. Find and accept the terms for Meta's Llama 3.2 models
3. This grants your account permission to download these models

### Step 5: Deploy the Container

Now, deploy the vLLM container with Llama 3.2:

$ docker run --runtime nvidia --gpus all \

-v ~/.cache/huggingface:/root/.cache/huggingface \

--env "HUGGING\_FACE\_HUB\_TOKEN=<HF\_TOKEN>" \

-p 8000:8000 \

--ipc=host \

vllm/vllm-openai:latest \

--model meta-llama/Llama-3.2-1B

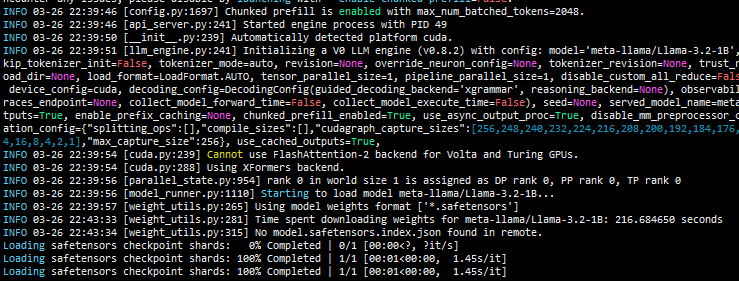
--dtype=half

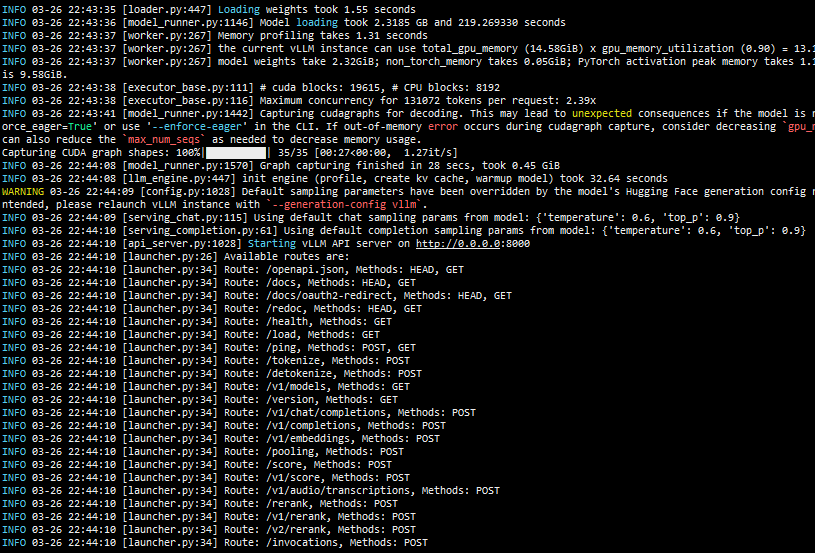
Let's break down this command:

* **--runtime nvidia --gpus all** : Enables GPU access in the container
* **-v ~/.cache/huggingface:/root/.cache/huggingface** : Mounts your local Hugging Face cache to avoid repeated downloads
* **--env "HUGGING\_FACE\_HUB\_TOKEN=<HF\_TOKEN>"** : Sets your Hugging Face token (replace <HF\_TOKEN> with your actual token)
* **-p 8000:8000** : Maps port 8000 from the container to your host
* **--ipc=host** : Sets up shared memory for improved performance
* **--model meta-llama/Llama-3.2-1B** : Specifies the model to load (1B parameter version)
* **--dtype=half** : Uses half-precision (FP16) for better performance

When you run this command, the container will:

1. Download the Llama 3.2 1B model from Hugging Face (if not already cached)
2. Load the model into GPU memory
3. Start the OpenAI-compatible API server on port 8000





### Step 6: Test the Deployment

Once the server is running, test it with a simple curl command:

$ curl -X POST "http://<host\_ip>:8000/v1/completions" \

-H "Content-Type: application/json" \

-d '{

"model": "meta-llama/Llama-3.2-1B",

"prompt": "capital of india and pakistan",

"temperature": 0.7,

"top\_k": -1,

"max\_tokens": 500

}'

Replace **<host\_ip>** with your server's IP address (use localhost if running locally).

A computer screen with text on it

AI-generated content may be incorrect.

This command sends a request to the completion’s endpoint with:

* The model identifier
* A system prompt
* Temperature setting of 0.7 (controls randomness)
* No top-k filtering
* Maximum response length of 500 tokens

If everything is working correctly, you should receive a response with a joke from the model.

### Conclusion

Congratulations! You've successfully deployed Meta's Llama 3.2 model using vLLM and Docker. This setup gives you a local alternative to cloud-based AI APIs, with complete control over your data and

infrastructure.

This approach is particularly useful for:

* Development and testing
* Reducing API costs for high-volume applications
* Customizing the inference parameters

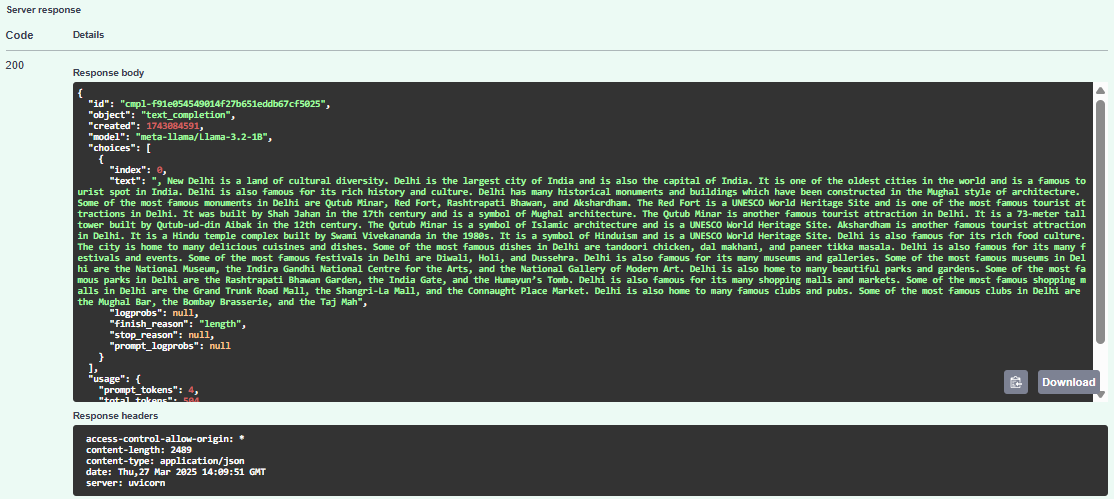
### API Screenshots

A screenshot of a computer

AI-generated content may be incorrect.

A black rectangular object with white lines

AI-generated content may be incorrect.



### FASTAPI

A screenshot of a computer

AI-generated content may be incorrect.

A green and white striped background

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.