Table of Contents

[Project Overview 3](#_Toc204171588)

[Week-by-Week Progress 3](#_Toc204171589)

[📅 Day-by-Day Journey Log 4](#_Toc204171590)

[🗓️ Week 1 4](#_Toc204171591)

[🗓️ Week 2 4](#_Toc204171592)

[🗓️ Week 3 5](#_Toc204171593)

[System Architecture and Diagrams: 5](#_Toc204171594)

[🔧 Docker Hierarchy Diagram 5](#_Toc204171595)

[📦 End-to-End Flow of Inference System 6](#_Toc204171596)

[🧱 Docker Compose Networked Architecture 6](#_Toc204171597)

[🛠️ Local Setup → Image → Container → App Flow 7](#_Toc204171598)

[Challenges Faced & Solutions 7](#_Toc204171599)

[Current State & Next Steps 7](#_Toc204171600)

[Flow of final working 8](#_Toc204171601)

[Detailed Local vLLM Setup (CPU-Based) 8](#_Toc204171602)

[System Update and Compiler Setup 8](#_Toc204171603)

[Creating and Activating Virtual Environment 9](#_Toc204171604)

[Cloning and Installing vLLM 9](#_Toc204171605)

[Installing Build Requirements 9](#_Toc204171606)

[Compiling vLLM for CPU 9](#_Toc204171607)

[Tuning Memory Allocation 9](#_Toc204171608)

[Inspecting System Resources 9](#_Toc204171609)

[Optimizing for CPU Inference 9](#_Toc204171610)

[Starting vLLM Server 10](#_Toc204171611)

[🐳 Docker Commands Reference 10](#_Toc204171612)

[🔨 Image Building 10](#_Toc204171613)

[📦 Image and Container Management 10](#_Toc204171614)

[🚀 Running Containers 11](#_Toc204171615)

[🔗 Networking 11](#_Toc204171616)

[💾 Volume Management 11](#_Toc204171617)

[🧹 Cleanup 11](#_Toc204171618)

InstructStack: Containerized Inference Platform for LLMs using vLLM + Docker + OpenAI-Compatible APIs

# Project Overview

InstructStack is a containerized inference platform designed to serve Large Language Models (LLMs) efficiently using:  
- vLLM as the inference engine  
- Docker for containerization  
- OpenAI-Compatible REST APIs  
  
This project is designed to work on both CPU and GPU hardware, with a strong focus on scalable, memory-optimized inference, local model deployment, and ease of use.

# Week-by-Week Progress

**Week0+1**: R&D, Learning Docker, Initial Setup  
- Learned the fundamentals of Docker: containers, images, volumes, networks, port mapping, Dockerfiles, and Compose.  
- Attempted to install large models like LLaMA — realized infeasibility on CPU.  
- Installed vLLM in WSL (Ubuntu), since Windows lacked compatibility.  
- Installed using requirements/cpu.txt, resolved compatibility and memory issues.  
- First success using sshleifer/tiny-gpt2 model.  
**Challenges and Errors**: Setting Up vllm with all the cpu and its related requirements.

Done this process **2 times** in week 1 .   
**Week 2 +3**: Dockerization and Automation  
- Created custom Dockerfile for vLLM and FastAPI.  
- Made Docker images and containers for both apps.  
- Used Docker Compose to spin up both services with a shared network.  
- Used Docker volumes to mount local HuggingFace models.

**-** More Models, Frontend, and Communication  
- Built FastAPI app that takes model name, prompt, and max tokens.  
- Created HTML frontend inside FastAPI container.  
- Backend hit OpenAI-compatible vLLM APIs via curl.

**Challenges and errors**: Building docker image from dockerfile and its related files and then after building getting errors for running vllm inside containers and server crashes when hitting server with prompt.

## 📅 Day-by-Day Journey Log

### 🗓️ Week 1

#### ✅ Day 1 – Introduction & Project Kickoff

* Understood the project scope.
* Learned basic Docker concepts: containers, images, Dockerfile, Docker Compose.
* Installed Docker Desktop and began exploring terminal commands.

#### ✅ Day 2 – Exploring Docker Further & Initial Directory Setup

* Deepened knowledge of Docker internals: volumes, networks, port mapping.
* Explored Docker Compose syntax and multi-container orchestration.
* Created initial project directory.
* Attempted downloading a model (tinyllama) but deferred setup.

#### ✅ Day 3 – First Attempt with vLLM on Windows

* Tried setting up vLLM in Windows but hit compatibility issues (e.g., unsupported modules).
* Shifted to WSL (Windows Subsystem for Linux).

#### ✅ Day 4 – vLLM Local CPU Setup in WSL (Ubuntu)

* Installed dependencies:
* sudo apt-get update -y
* sudo apt-get install -y gcc-12 g++-12 libnuma-dev python3-dev

sudo update-alternatives --install /usr/bin/gcc gcc /usr/bin/gcc-12 10 --slave /usr/bin/g++ g++ /usr/bin/g++-12

* Created and activated conda virtual environment.
* Cloned vLLM GitHub repo:
* git clone https://github.com/vllm-project/vllm.git vllm\_source
* cd vllm\_source
* pip install --upgrade pip
* pip install vllm
* pip install "cmake>=3.26.1" wheel packaging ninja "setuptools-scm>=8" numpy
* pip install -v -r requirements/cpu.txt --extra-index-url https://download.pytorch.org/whl/cpu

VLLM\_TARGET\_DEVICE=cpu python setup.py install

* Set performance variables and linked libtcmalloc:
* export LD\_PRELOAD=/usr/lib/x86\_64-linux-gnu/libtcmalloc\_minimal.so.4
* export VLLM\_CPU\_KVCACHE\_SPACE=4

export VLLM\_CPU\_OMP\_THREADS\_BIND=0-7

* Successfully served tinygpt2 model with vLLM.

#### ✅ Day 5 – Errors & vLLM Reinstallation

* Tried running larger models but encountered errors.
* Reinstalled vLLM cleanly and recovered functionality.

### 🗓️ Week 2

#### ✅ Day 1 – Writing Dockerfiles & Building Images

* Started writing Dockerfiles for vLLM and FastAPI.
* Attempted to build image using:
* docker build -t hamzaak4/vllm-cpu-image:Latest1.1 .

docker build -t fastapi-vllm:Latest .

* Faced build issues.
* Simultaneously began learning FastAPI and created a basic prompt input form.

#### ✅ Day 2 – Docker Errors Persist, Frontend Advances

* Image builds continued to fail.
* Advanced the FastAPI backend and frontend for testing.

#### ✅ Day 3 – First Successful Image Build 🎉

* Successfully built the vLLM image.
* Ran container:

docker run --rm --privileged -it -p8000:8000 --name vllm\_server hamzaak4/vllm-cpu-image:Latest1.1 /bin/bash

#### ✅ Day 4 – Endpoint Errors, Debugging

* Served model but FastAPI crashed when hitting vLLM endpoint.
* Diagnosed networking and environment variable issues.

### 🗓️ Week 3

#### ✅ Day 1 to Day 3 – Full Containerization & Networking

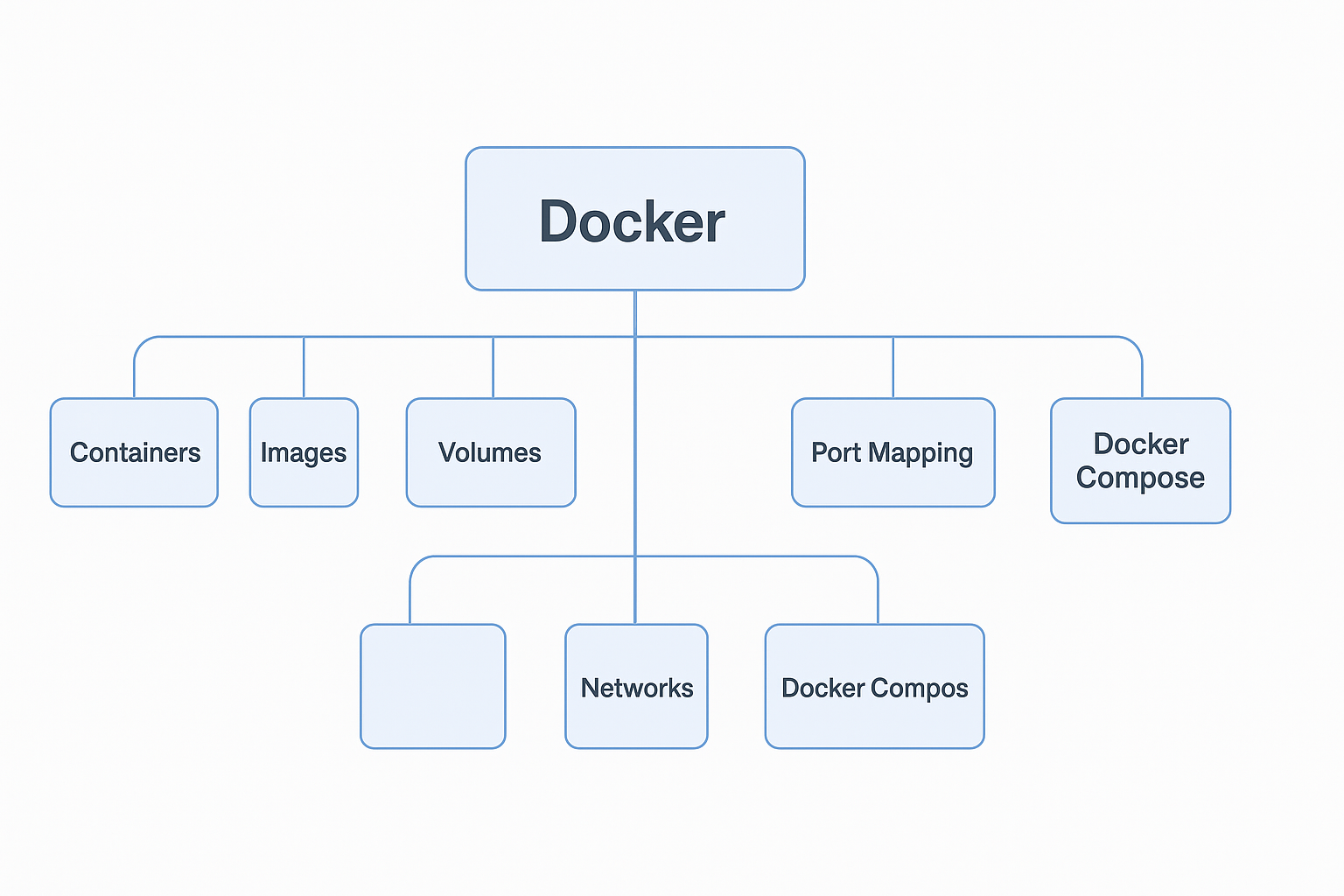
* Created a shared volume for model weights:

docker volume create vllm-models

* Created Docker Compose YAML for running FastAPI and vLLM containers with proper networking.
* Finalized architecture with:
  + vLLM container mounting models from volume.
  + FastAPI sending user prompts to vLLM endpoint.
  + FastAPI displaying responses in the frontend.

# System Architecture and Diagrams:

## 🔧 Docker Hierarchy Diagram



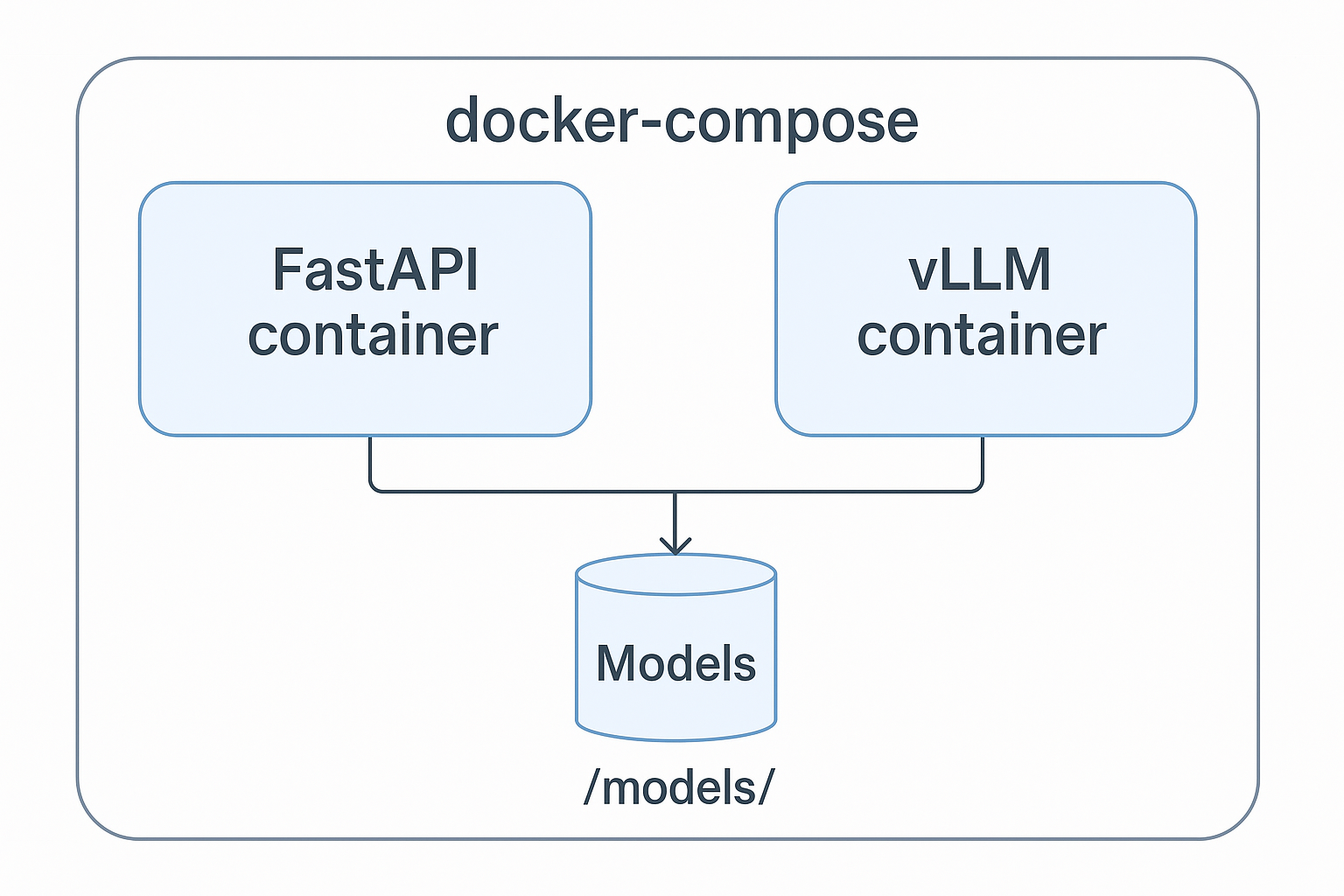
In the start of the project I have learned about these topics and have practically used them in my project in the week2. I will be sharing all the commands that I have used in my docker journey in this project. **I was unfamiliar with docker before this project but I have done practically in it and learning about docker and then using it practically that’s my learning**

## 📦 End-to-End Flow of Inference System

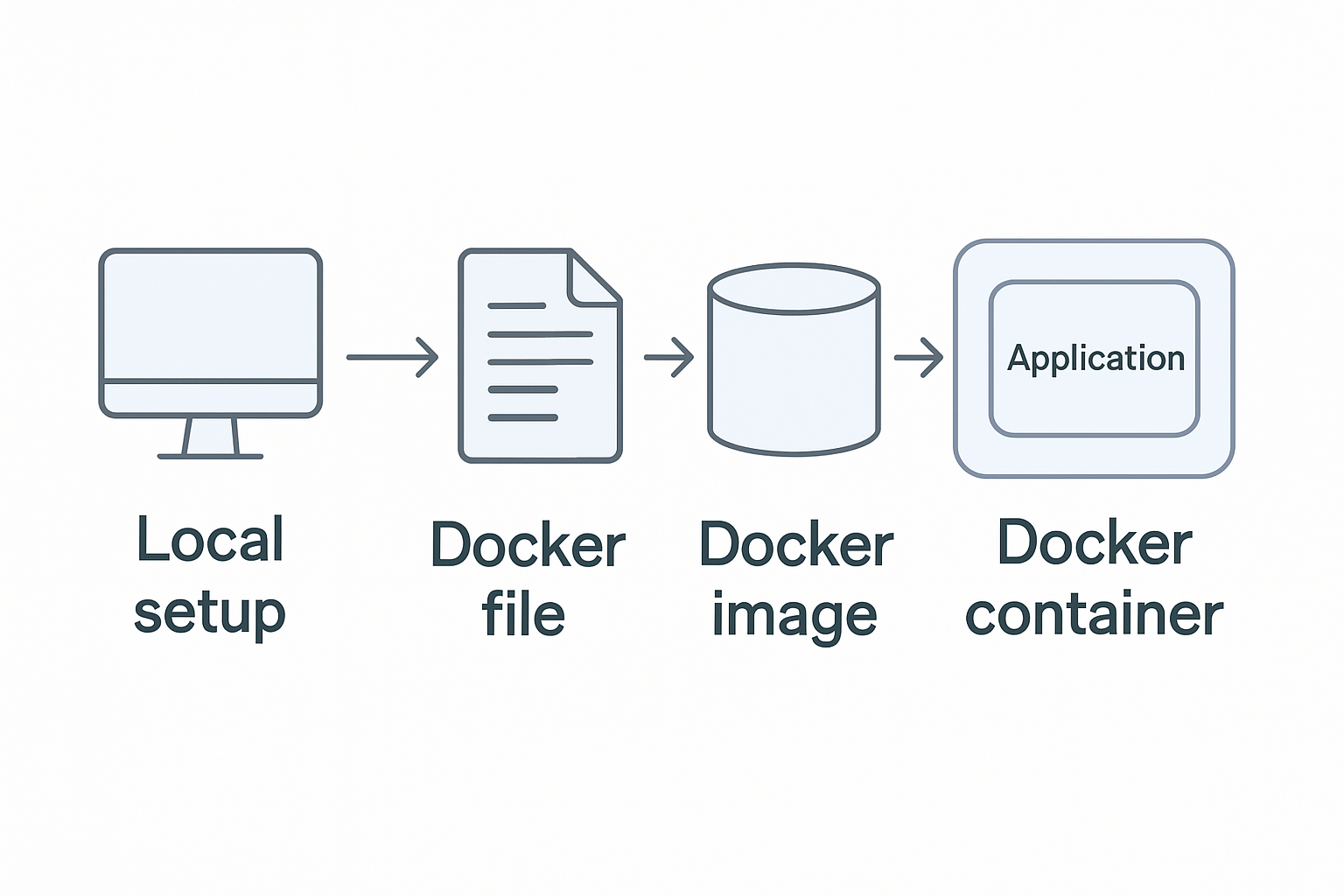
This image is a flow of working installing python means just running some commands before setting up the vllm and then installing vllm with requirements for cpu and then vllm is setup then dockerize it.

## 🧱 Docker Compose Networked Architecture

Running manually commands for vllm container and fastapi and then working with it here comes the docker compose and makes the differences. Now only one command and that would done the work for you.



## 🛠️ Local Setup → Image → Container → App Flow



So When I have to start this project first I learnt about the vllm that what it is then I got to know it’s a inference engine which used special attension mechanism called paged attension which is an excellent memory efficient technique and about KV Cache which plays a key role in continuous batching offered by vllm.

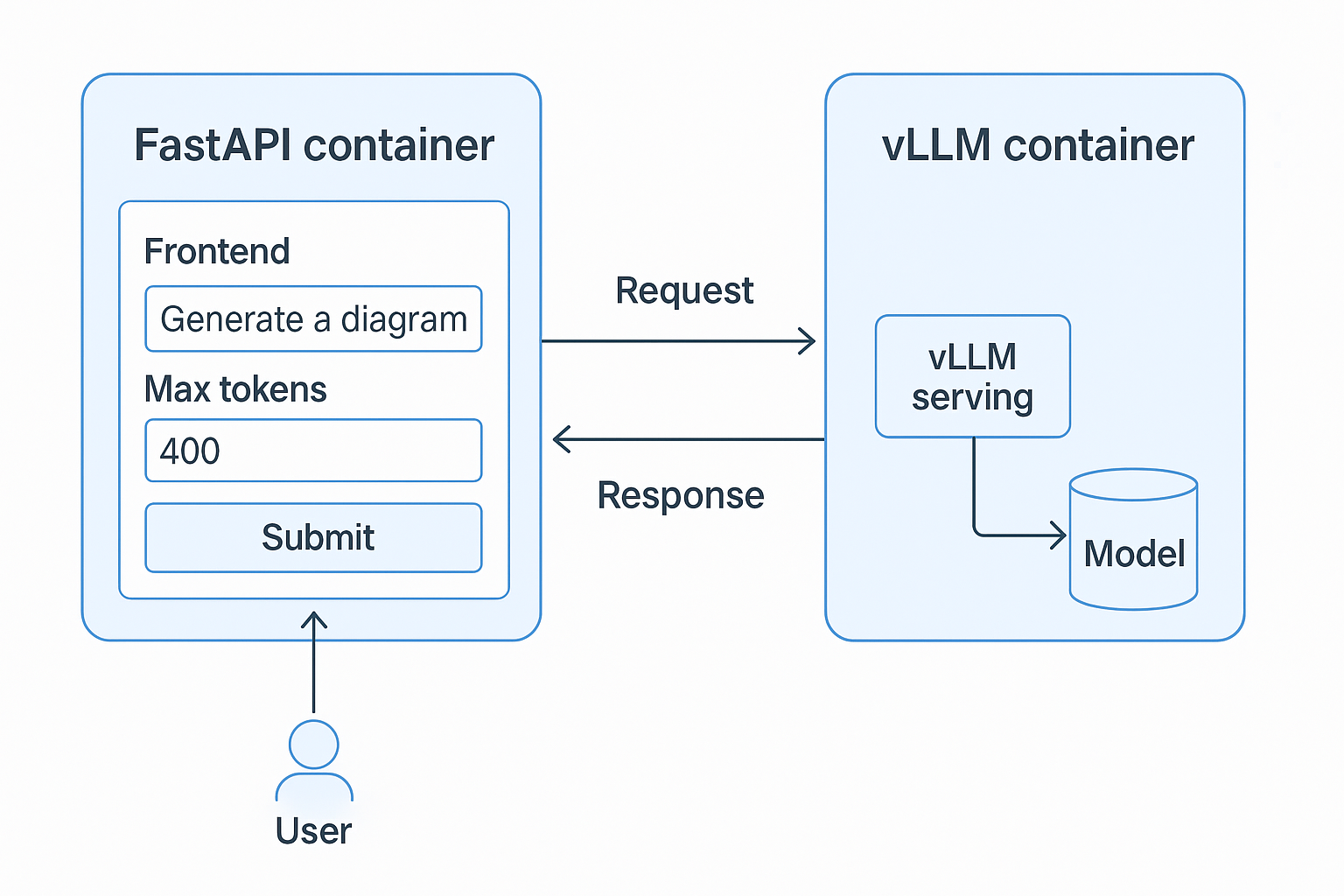
# Challenges Faced & Solutions

|  |
| --- |
| Challenge | Solution   ---------------------------------- --------------------------------------------------------------   Tried large models on CPU Switched to smaller models (tiny-gpt2, opt-125m)   Windows compatibility issues Moved to WSL + Ubuntu   vLLM not installing properly Disabled Windows Defender; used Conda & requirements/cpu.txt   vLLM crashing in Docker Used environment variables to fix tensor errors   FastAPI and vLLM not communicating | Used Docker Compose for networking   Long Docker build time Split long RUN commands in Dockerfile |

# Current State & Next Steps

**Achieved So Far:**- Deployed vLLM on CPU locally and inside Docker.  
- Built a working FastAPI app with HTML UI.  
- Integrated the system with OpenAI-compatible API calls.  
- Automated everything using Docker Compose.  
- Enabled offline model serving from local HuggingFace downloads.  
**Next Targets:**  
- Transition to GPU-based inference.  
- Push final Docker images to Docker Hub.  
- Integrate SQL-focused LLMs (e.g., text-to-SQL).  
- Explore Kubernetes for cluster-level orchestration.

# Flow of final working



This is the actual flow of the containerized application. What I have learned is that how to dockerize the application by writing a dockerfile and then building a image from that docker and then docker container from it and running application inside it that’s also a challenge.

# Detailed Local vLLM Setup (CPU-Based)

## System Update and Compiler Setup

* sudo apt-get update -y
* sudo apt-get install -y gcc-12 g++-12 libnuma-dev python3-dev
* sudo update-alternatives --install /usr/bin/gcc gcc /usr/bin/gcc-12 10 --slave /usr/bin/g++ g++ /usr/bin/g++-12

📘 These commands install required compilers and development libraries necessary for compiling vLLM and its dependencies.

## Creating and Activating Virtual Environment

📘 Using Conda, a new isolated virtual environment was created for dependency management.

# conda create -n vllm\_env python=3.11

# conda activate vllm\_env

## Cloning and Installing vLLM

* git clone https://github.com/vllm-project/vllm.git vllm\_source
* cd vllm\_source
* pip install --upgrade pip
* pip install vllm

📘 The vLLM repository is cloned and the base package is installed for further configuration.

## Installing Build Requirements

* pip install "cmake>=3.26.1" wheel packaging ninja "setuptools-scm>=8" numpy
* pip install -v -r requirements/cpu.txt --extra-index-url https://download.pytorch.org/whl/cpu

📘 These packages are essential to build and run vLLM efficiently on CPUs.

## Compiling vLLM for CPU

* VLLM\_TARGET\_DEVICE=cpu python setup.py install

📘 This explicitly sets the target device to CPU during installation.

## Tuning Memory Allocation

* find / -name \*libtcmalloc\* 2>/dev/null
* export LD\_PRELOAD=/usr/lib/x86\_64-linux-gnu/libtcmalloc\_minimal.so.4

📘 Google's TCMalloc is used to improve memory allocation performance during inference.

## Inspecting System Resources

* lscpu -e
* free -h

📘 These commands inspect CPU and memory layout for optimal environment variable tuning.

## Optimizing for CPU Inference

* export VLLM\_CPU\_KVCACHE\_SPACE=4
* export VLLM\_CPU\_OMP\_THREADS\_BIND=0-7

📘 These variables control key-value cache memory size and OpenMP thread pinning for multithreaded performance.

## Starting vLLM Server

* python -m vllm.entrypoints.openai.api\_server --model /path/to/model --device cpu
* vllm serve /path/to/model --device cpu

📘 This command is used to hit the vllm endpoint to test it with prompt

* curl http://localhost:8000/v1/completions \

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

-d '{

"model": "facebook/opt-125m",

"prompt": "Once upon a time",

"max\_tokens": 20

}'

📘 The vLLM API server is launched to serve LLMs locally via OpenAI-compatible endpoints.

# 🐳 Docker Commands Reference

Below is a list of Docker CLI commands you used during development and deployment of the vLLM system, along with a brief explanation for each:

### 🔨 Image Building

|  |  |
| --- | --- |
| Command | Description |
| docker build -t hamzaak4/vllm-cpu-image:Latest1.1 . | Builds a Docker image from the Dockerfile in the current directory and tags it as vllm-cpu-image:Latest1.1. |
| docker build -f Dockerfile -t fastapi-vllm:Latest . | Builds a Docker image for the FastAPI app using a specific Dockerfile and tags it. (Make sure to include the -t tag and context .). |

### 📦 Image and Container Management

|  |  |
| --- | --- |
| Command | Description |
| docker images | Lists all locally available Docker images. |
| docker ps | Shows currently running Docker containers. |
| docker ps -a | Lists all containers, including stopped ones. |
| docker stop <container\_name> | Stops a running container by name or ID. |
| docker rm <container\_name> | Removes a stopped container. |
| docker rmi <image\_name> | Removes a Docker image. |

### 🚀 Running Containers

|  |  |
| --- | --- |
| Command | Description |
| docker run --rm --privileged -it -p8000:8000 --name vllm\_server hamzaak4/vllm-cpu-image:Latest1.1 /bin/bash | Runs the vLLM container interactively, maps port 8000, and removes the container after exit. |
| docker run --rm -it -p9000:9000 --name fastapi-app fastapi-vllm:Latest /bin/bash | Runs the FastAPI app container interactively, maps port 9000. |

### 🔗 Networking

|  |  |
| --- | --- |
| Command | Description |
| docker network ls | Lists all Docker networks. |
| docker network create <name> | Creates a custom Docker network. |
| docker inspect <container\_name> | Displays low-level information (useful for IP, network details, etc.). |

### 💾 Volume Management

|  |  |
| --- | --- |
| Command | Description |
| docker volume create <name> | Creates a new Docker volume (used to persist data like model files). |
| docker volume ls | Lists all Docker volumes. |
| docker volume inspect <volume\_name> | Shows metadata and mount path of the volume. |
| docker volume rm <volume\_name> | Deletes a volume. Be sure it’s not attached to any container. |

### 🧹 Cleanup

|  |  |
| --- | --- |
| Command | Description |
| docker system prune | Removes all unused containers, networks, images, and volumes (be cautious!). |
| docker container prune | Removes all stopped containers. |
| docker image prune | Deletes dangling images. |