OFFLINE AI CHATBOT SYSTEM USING OPEN-SOURCE LLMS AND LOCAL NETWORK TOPOLOGY

A Mini Project Report

Submitted in partial fulfillment of the requirements for the award of the degree of

Bachelor of Technology

in

Computer Science and Engineering (Artificial Intelligence & Data Science)

Submitted by:

Pavan Adabala (122210701103)

Under the Guidance of:

Dr. A. B. Manju

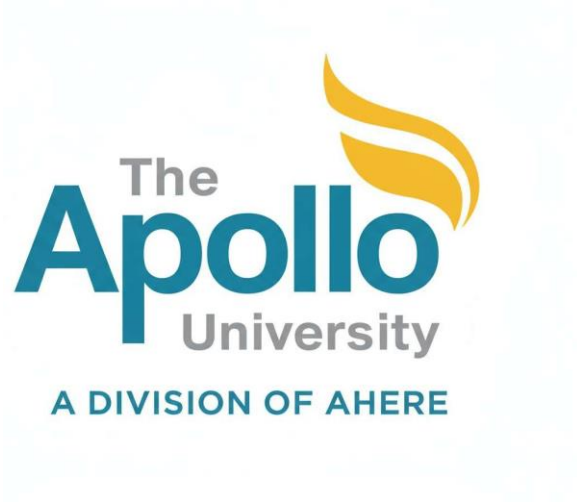
(Department of Computer Science and Engineering)

Academic Year: 2024–2025

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING (AI&DS)

THE APOLLO UNIVERSITY

CHITTOOR – ANDHRA PRADESH, INDIA



CERTIFICATE

This is to certify that the project report entitled **"Offline AI Chatbot for University Campus"** is a bonafide work carried out by **Pavan Adabala (122210701103)** in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in Computer Science and Engineering (AI & DS) from The Apollo University during the academic year 2024-2025.

The project work has been carried out under my guidance and supervision and is a result of their own effort.

Signature of Project Guide
Dr. A. B. Manju
Project Guide
Department of CSE
The Apollo University

Signature of Dean
Prof. D. Jagadeesan
Dean of SOT
The Apollo University

DECLARATION

I, **Pavan Adabala**, bearing Roll Number **122210701103**, a student of B.Tech in Computer Science and Engineering (AI & DS) at The Apollo University, hereby declare that the project report entitled **"Offline AI Chatbot for University Campus"** submitted by me in partial fulfillment of the B.Tech Degree is a record of Bonafide work carried out by me under the guidance of **Dr. A B Manju**.

I further declare that the work reported in this project has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

Pavan Adabala
122210701103

Date:
Place: Chittoor

ACKNOWLEDGEMENT

I would like to express my deepest gratitude to my project guide, **Dr. A B Manju**, for his invaluable guidance, constant encouragement, and insightful feedback throughout the duration of this project. His expertise and vision were instrumental in shaping this work and bringing it to fruition.

I am also immensely thankful to the Head of the Department and all the faculty members of the Department of Computer Science and Engineering (AI & DS) at **The Apollo University** for providing the necessary infrastructure and a conducive environment for research and development.

I extend my sincere thanks to the technical staff for their support in setting up the laboratory and network infrastructure, which was crucial for the implementation and testing phases of the project.

My heartfelt appreciation goes to my friends and classmates who have provided me with their support, engaged in stimulating discussions, and helped me overcome various challenges encountered during the project.

Finally, I am profoundly grateful to my family for their unwavering love, support, and encouragement. Their belief in me has always been my greatest source of strength.

Pavan Adabala

Date:
Place: Chittoor

ABSTRACT

The proliferation of AI-powered conversational agents has transformed user interaction with technology. However, the majority of these systems rely on cloud-based APIs, raising significant concerns about data privacy, latency, and operational costs, especially within sensitive environments like university campuses. This project addresses these challenges by designing and implementing a fully Offline AI Chatbot System tailored for a university campus.

The system leverages open-source Large Language Models (LLMs) such as Mistral 7B, Qwen 2.5, and a specialized Med-LLaMA, which are hosted locally on a dedicated server using LM Studio. This architecture ensures that all data processing occurs within the campus network, guaranteeing complete data privacy and eliminating dependency on internet connectivity. The chatbot is accessible to students and faculty via a browser-based interface powered by OpenWebUI, connected to the campus Wireless Local Area Network (WLAN).

The chatbot's functionalities are twofold: it provides academic support by answering queries related to subjects, lab procedures, and campus information, and it offers health and wellness guidance through the Med-LLaMA model. This integrated health module provides symptom-based advice and dietary suggestions, promoting student well-being. For secure, temporary remote access, the system incorporates Ngrok tunneling, creating an HTTPS endpoint for authenticated users outside the primary network.

The project's core contribution is the creation of a self-sustained, private, and intelligent AI ecosystem. By deploying edge-based AI, this work demonstrates a scalable, secure, and cost-effective model for digital self-reliance in educational institutions, enhancing real-time interaction and information accessibility without compromising security

Table of Contents

List of Tables

List of Figures

Chapter 1:
Introduction

1.1 Overview of the Project

In recent years, the advancement of Artificial Intelligence (AI), particularly in the field of Natural Language Processing (NLP), has led to the development of sophisticated conversational agents known as chatbots. These systems, powered by Large Language Models (LLMs), are capable of understanding, generating, and interacting in human-like text. The proposed project, "Offline AI Chatbot for University Campus," aims to harness this technology to create a self-contained, secure, and efficient information and assistance tool for the academic community of The Apollo University.

This project involves the design and implementation of an AI chatbot that operates entirely within the university's local network. By hosting open-source LLMs like Mistral 7B and Qwen 2.5 on a local server using LM Studio, the system eliminates the reliance on external cloud services. This offline-first approach ensures data privacy, reduces response latency, and negates recurring costs associated with API calls. The chatbot provides academic support, campus information, and includes a specialized health advisory module using the Med-LLaMA model, making it a comprehensive tool for student and faculty welfare.

1.2 Motivation and Need for Offline Chatbots

While commercial chatbots like ChatGPT and Google Gemini offer powerful capabilities, their cloud-based nature presents several challenges for institutional use:

Data Privacy: Student queries, which may contain sensitive academic or personal information, are sent to third-party servers, posing a significant privacy risk.

Internet Dependency: Access to these services is contingent on a stable and fast internet connection, which may not always be available across an entire campus or can be a bottleneck during peak hours.

Cost: Heavy usage of commercial LLM APIs can lead to substantial recurring costs, making them financially unsustainable for many educational institutions.

Latency: Round trips to external servers can introduce noticeable delays in response times, affecting the user experience.

Lack of Customization: Cloud-based models are generic and cannot be easily fine-tuned with specific, private institutional data to provide contextually accurate answers about campus-specific matters.

An offline chatbot addresses these issues directly by creating a private AI ecosystem. It empowers the institution with full control over its data, ensures uninterrupted service regardless of external internet status, offers a one-time setup cost, and provides near-instantaneous responses. This makes it an ideal solution for a modern, digitally-enabled university campus.

1.3 Objectives of the Project

The primary objectives of this project are:

To design and develop a fully functional offline AI chatbot system for the university campus.

To set up a local server environment using LM Studio to host and serve open-source LLMs.

To integrate and evaluate the performance of multiple LLMs, including Mistral 7B for general academic queries and Med-LLaMA for health-related advice.

To develop a user-friendly, browser-based interface using OpenWebUI for seamless interaction with the chatbot.

To configure the system to be accessible via the campus Wireless Local Area Network (WLAN).

To implement a secure remote access solution using Ngrok tunneling for authorized users.

To ensure data privacy and security by processing all user queries and data within the local network.
To evaluate the system's performance in terms of response accuracy, latency, and user satisfaction.

1.4 **Scope of Work**

The scope of this project encompasses the end-to-end development of the offline chatbot system. This includes the selection and configuration of hardware, installation of software, model selection and fine-tuning, front-end development, and network integration. The chatbot's functionalities will cover:

Academic Assistance: Answering questions about course syllabi, explaining complex concepts, providing code snippets for programming labs, and guiding students on academic procedures.

Campus Information: Providing details about campus facilities, event schedules, department contacts, and administrative processes.

Health & Wellness Support: Offering preliminary advice on common health issues, suggesting dietary plans, and providing information on wellness resources available on campus, powered by the Med-LLaMA model.

The system is designed to be scalable, allowing for the future addition of new models, datasets, and functionalities. However, the project's scope does not extend to providing certified medical diagnoses or replacing professional medical consultation.

1.5 **Problem Definition**

Educational institutions require a modern, efficient, and readily available information dissemination system to cater to the diverse needs of students and faculty. Existing methods, such as notice boards, emails, or administrative helplines, are often slow, inefficient, and not available 24/7. While cloud-based AI chatbots offer a potential solution, they introduce critical concerns regarding data privacy, internet dependency, and operational costs. Therefore, the problem is to create an intelligent, secure, and self-reliant AI assistant that operates exclusively within the campus network, providing instant, context-aware information and support without compromising sensitive institutional or personal data.

1.6 **Expected Outcomes**

Upon successful completion, this project is expected to deliver:

A fully operational offline AI chatbot accessible within the campus WLAN.

A documented framework for hosting and managing multiple LLMs locally using LM Studio.

A comparative analysis of the performance of different open-source LLMs (Mistral 7B, Qwen 2.5, Med-LLaMA) for specific tasks.

A secure and responsive web interface for user interaction.

A significant reduction in query response time compared to internet-based chatbots.

A demonstration of a cost-effective and privacy-preserving AI solution for educational institutions.

Chapter 2:
Literature Review

2.1 Introduction to AI Chatbots

An AI chatbot is a computer program designed to simulate human conversation through text or voice commands. Early chatbots were based on simple rule-based systems, such as ELIZA (1966), which used pattern matching to generate responses. The advent of machine learning and deep learning has revolutionized this field. Modern chatbots, particularly those built on Large Language Models (LLMs), leverage transformer architectures to understand context, nuance, and sentiment, enabling far more sophisticated and meaningful interactions. This literature review explores the evolution of LLMs, compares the dominant paradigms of online and offline chatbots, and examines existing implementations in academic settings to identify the research gaps this project aims to fill.

2.2 Evolution of LLMs

The development of LLMs has been a cornerstone of recent AI advancements. The journey began with models like Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) networks, which were proficient at handling sequential data but struggled with long-range dependencies. The introduction of the Transformer architecture in Vaswani et al.'s (2017) paper, "Attention Is All You Need," marked a significant turning point. This architecture's self-attention mechanism allowed models to weigh the importance of different words in the input, leading to superior context understanding.

This led to the creation of large-scale pre-trained models like:

GPT (Generative Pre-trained Transformer) series: Developed by OpenAI, models like GPT-3 and GPT-4 have set industry benchmarks for text generation and understanding, but are primarily accessible via proprietary APIs.

LLaMA (Large Language Model Meta AI): Released by Meta, LLaMA and its successors (LLaMA 2, LLaMA 3) provided powerful open-source alternatives, spurring a wave of community-driven fine-tuning and development. Med-LLaMA is a derivative specialized in the medical domain.

Mistral: Mistral AI has released highly efficient models like Mistral 7B, which offer performance comparable to much larger models, making them ideal for local deployment on consumer-grade hardware.

Qwen: Developed by Alibaba Cloud, the Qwen series of models provide strong multilingual capabilities and are also available in various sizes for different deployment scenarios.

The availability of such powerful open-source models has made projects like this offline chatbot feasible, democratizing access to state-of-the-art AI technology.

2.3 Comparison of Online vs Offline Chatbots

The deployment architecture of a chatbot is a critical decision that impacts its performance, security, and cost. The two primary models are online (cloud-based) and offline (local/edge-based).

Table 2.1: Comparison of Online vs Offline Chatbots

Feature	Online (Cloud-Based) Chatbot	Offline (Local) Chatbot
Processing Location	Remote servers owned by a third-party (e.g., OpenAI, Google).	On-premise server within the organization's network.
Data Privacy	Low. Data is sent over the internet and processed externally, raising privacy concerns.	High. All data remains within the local network, ensuring maximum privacy and control.
Internet Dependency	High. Requires a constant and stable internet connection to function.	None. Operates independently of external internet access.
Latency	Variable to high, depending on network traffic and server load.	Very low, as data transfer is limited to the local network.
Cost Model	Recurring (per API call or subscription-based). Can be expensive at scale.	One-time hardware investment. No recurring operational costs for the model itself.
Scalability	Extremely high, managed by the cloud provider.	Limited by the capacity of the local server hardware.
Maintenance	Managed by the provider. Models are updated automatically.	Requires in-house management of hardware, software, and model updates.

2.4 **Existing Campus Chatbot Systems**

Several universities have implemented chatbots to assist students. For instance, Georgia Tech's "Jill Watson" started as a virtual teaching assistant to answer student questions in an online course. Deakin University in Australia launched "Deakin Genie," a voice-activated virtual assistant. However, a majority of these systems either rely on cloud-based NLP services from Google Dialog flow, Amazon Lex, or Microsoft Bot Framework, or are simple rule-based systems with limited conversational ability.

While effective, these systems inherit the limitations of their underlying cloud architecture. Few published works focus on deploying modern, LLM-based chatbots in a completely offline environment for a university. The novelty of this project lies in leveraging recent advancements in open-source LLMs that are efficient enough to be self-hosted, thereby creating a truly private and autonomous AI assistant for the campus.

2.5 **Summary of Research Gaps**

The literature review reveals a clear gap in the research and implementation of self-hosted, LLM-powered chatbots within educational institutions. The key gaps identified are:

A lack of focus on data privacy and digital sovereignty in campus AI implementations.

Limited exploration of open-source LLMs for creating specialized, offline campus assistants.

Insufficient research on the practical challenges and performance benchmarks of hosting LLMs on institutional-grade, non-hyperscale hardware.

A need for integrated systems that combine academic support with other crucial services like student wellness and health advisories in a single, secure platform.

This project directly addresses these gaps by building a system that is, by design, private, autonomous, and customized for the specific needs of The Apollo University, demonstrating a viable alternative to cloud-dependent AI solutions.

Chapter 3:
System Analysis

3.1 Feasibility Study

A feasibility study was conducted to assess the viability of the project from technical, operational, and economic perspectives.

3.1.1 Technical Feasibility

The project's technical feasibility hinges on the availability of powerful open-source LLMs and the hardware to run them. The advent of quantized models (e.g., GGUF format) and efficient models like Mistral 7B makes it possible to run sophisticated AI on consumer-grade to prosumer-grade hardware. Software tools like LM Studio abstract away the complexities of model loading and provide a simple, OpenAI-compatible API endpoint. The front-end can be built with standard web technologies (HTML, CSS, JavaScript) or frameworks like OpenWebUI. The required network infrastructure (WLAN) is already standard in most university campuses. Therefore, the project is technically feasible with currently available technology.

3.1.2 Operational Feasibility

Operationally, the system is designed to be highly beneficial. It will provide 24/7 automated support to students and faculty, reducing the workload on administrative staff. The web-based interface is intuitive and requires no special training for users. The system can be maintained by the university's IT department. Its offline nature ensures continuous operation even during internet outages, a critical advantage for a campus environment. The project aligns with the university's goal of leveraging technology to enhance the student experience.

3.1.3 Economic Feasibility

The primary cost is the one-time investment in a dedicated server with a powerful GPU. While this initial capital expenditure can be significant, it is offset by the elimination of recurring subscription fees or API call charges associated with cloud-based services. A single server can handle thousands of queries daily at no additional operational cost. Compared to the long-term, escalating costs of a commercial AI service used by the entire campus population, the self-hosted model presents a more economically sustainable solution. The use of open-source software (LM Studio, OpenWebUI, Linux) further reduces costs.

3.2 System Requirements Specification

The requirements for the system are categorized into hardware and software.

Table 3.1: **Hardware Requirements**

Component	Specification	Purpose
Server CPU	Multi-core processor (e.g., Intel Core i7/i9, AMD Ryzen 7/9)	General processing, running OS and applications.
Server RAM	32 GB DDR4 or higher	Loading LLMs into memory. More RAM allows for larger models.

Server GPU	NVIDIA GeForce RTX 30/40 series with >= 12 GB VRAM	Crucial for accelerating LLM inference speed.
Storage	1 TB NVMe SSD or higher	Storing multiple LLM files (can be large), OS, and application data.
Network Interface	Gigabit Ethernet or Wi-Fi 6	Connecting the server to the campus WLAN.
Client Devices	Any device with a modern web browser (Laptop, Smartphone, Tablet)	Accessing the chatbot's web interface.

Table 3.2: **Software Requirements**

Software	Specification	Purpose
Operating System	Windows 10/11, macOS, or Linux (Ubuntu recommended)	Host OS for the server.
LLM Hosting	LM Studio	To download, manage, and serve LLMs via a local API.
Web Interface	OpenWebUI (self-hosted)	Provides a chat-style user interface for interacting with the LLMs.
LLM Models	Mistral 7B, Qwen 2.5, Med-LLaMA (GGUF format)	The AI brains of the chatbot.
Remote Access	Ngrok	To create a secure tunnel for temporary external access.
Web Browser	Google Chrome, Firefox, Safari (latest versions)	Client-side software for users.

3.3 **Functional vs Non-Functional Requirements**

3.3.1 Functional Requirements

Functional requirements define what the system is supposed to do.

Table 3.3: Functional Requirements

ID	Requirement	Description

FR1	User Query Processing	The system must accept text-based queries from users through a web interface.
FR2	LLM Integration	The system must be able to route queries to the appropriate locally hosted LLM.
FR3	Model Selection	The user interface should allow users to select which model to interact with (e.g., General, Medical).
FR4	Response Generation	The system must generate and display relevant, coherent responses from the LLM.
FR5	Local Access	The system must be accessible to any user connected to the campus WLAN.
FR6	Remote Access	The system should provide a secure URL via Ngrok for authenticated remote access.
FR7	Conversation History	The system should maintain the context of the current conversation session.

3.3.2 **Non-Functional Requirements**

Non-functional requirements define how the system should perform.

Table 3.4: Non-Functional Requirements

ID	Requirement	Description
NFR1	Performance	The system should provide responses with an average latency of less than 5 seconds for a typical query.
NFR2	Security	All data communication must be confined to the local network. Remote access must be authenticated and encrypted (HTTPS).
NFR3	Scalability	The system should be able to handle at least 50 concurrent users without significant performance degradation.
NFR4	Reliability	The system should have an uptime of 99.9% and run continuously without crashes.
NFR5	Usability	The user interface must be simple, intuitive, and accessible on both desktop and mobile devices.

NFR6	Privacy	No user data or queries should be transmitted outside the university's network.

3.4 Use Case Diagrams and Scenarios

A Use Case diagram illustrates the system's intended functionalities and user interactions. The primary actors are the Student/Faculty (User) and the System Administrator.

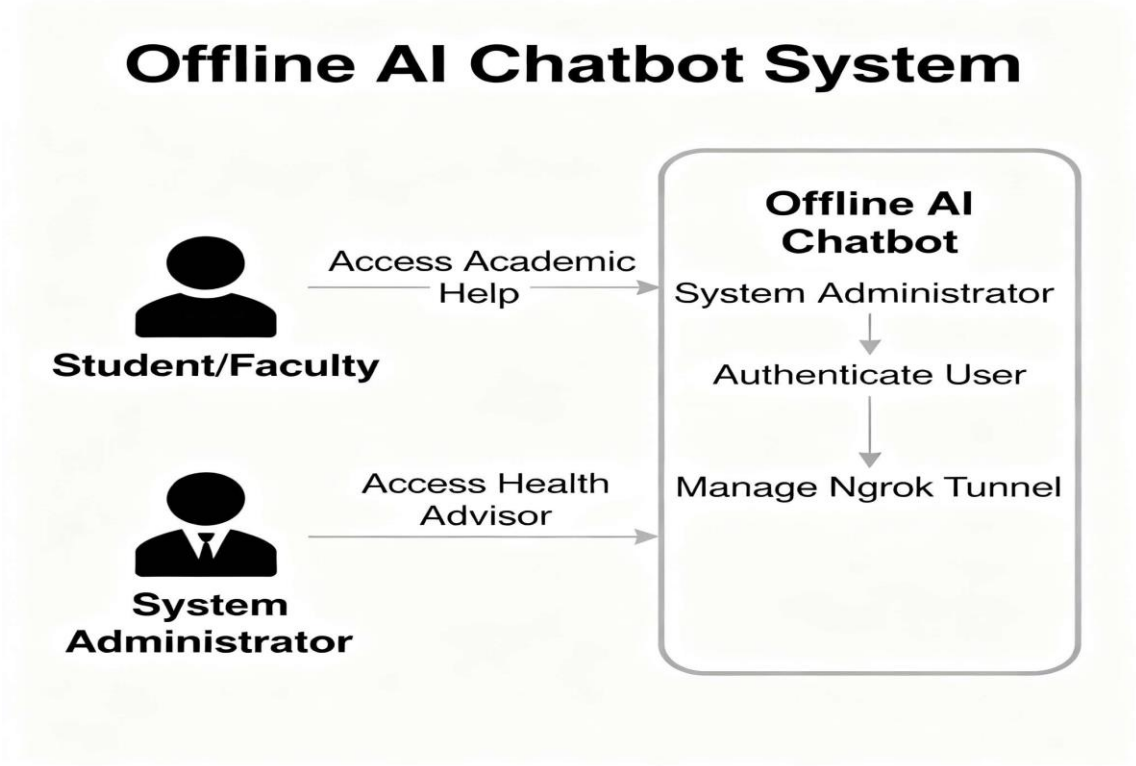


Figure 3.1: Use Case Diagram for the Chatbot System

Use Case Scenarios:

Scenario 1: Student seeks Academic Help

Students connect to the campus WLAN.
Student opens a web browser and navigates to the chatbot's local URL.
Student selects the 'Academic' model (Mistral 7B).
Student types: "Explain the concept of recursion in Python with an example."
The system processes the query and generates a detailed explanation with a code snippet.
The student receives the response in the chat interface.

Scenario 2: Faculty seeks Health Information (Remotely)

Administrator starts the Ngrok tunnel and shares the secure URL with the faculty member.
Faculty member accesses the URL from their home internet connection.
User authenticates to access the interface.
User selects the 'Health Advisor' model (Med-LLaMA).
User types: "What are some home remedies for a common cold?"

Chapter 4:
System Design

4.1 System Architecture

The system is designed using a multi-tiered architecture, ensuring modularity and separation of concerns. The architecture consists of three main layers: the Presentation Layer (Client-side), the Application Layer (Server-side), and the Model Layer (AI Engine).

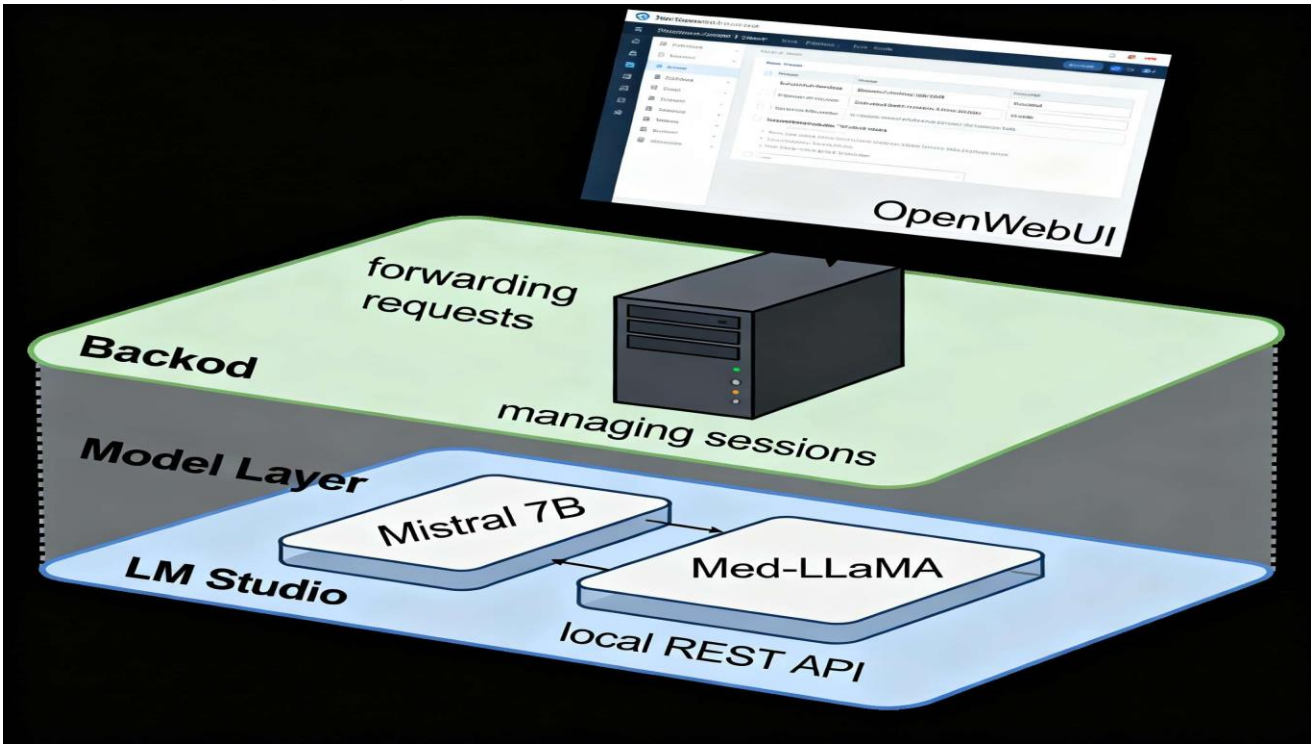


Figure 4.1: System Architecture Diagram

Presentation Layer: This is the user-facing component, a web-based interface provided by OpenWebUI. Users interact with the system through this layer from their browsers. It is responsible for rendering the chat interface, sending user input to the application layer, and displaying the model's response.

Application Layer: This layer acts as the middleman. The OpenWebUI backend server receives requests from the front-end. It then forwards these requests to the appropriate API endpoint provided by the Model Layer. It manages user sessions and conversation history.

Model Layer: This is the core AI engine of the system, managed by LM Studio. LM Studio loads the selected LLMs (Mistral 7B, Med-LLaMA, etc.) into the server's memory/VRAM and exposes a local, OpenAI-compatible REST API endpoint (e.g., `http://localhost:1234/v1/chat/completions`). This layer handles the actual text generation and inference.

This decoupled architecture allows for easy swapping of components. For example, the OpenWebUI front-end could be replaced with another client, or LM Studio could be swapped with another LLM serving tool, without affecting the other layers.

4.2 **Network Topology**

The system is designed to operate primarily within the university's campus network. A dedicated server is connected via Ethernet to the core network switch, making it accessible to any device connected to the campus's WLAN through a static local IP address.

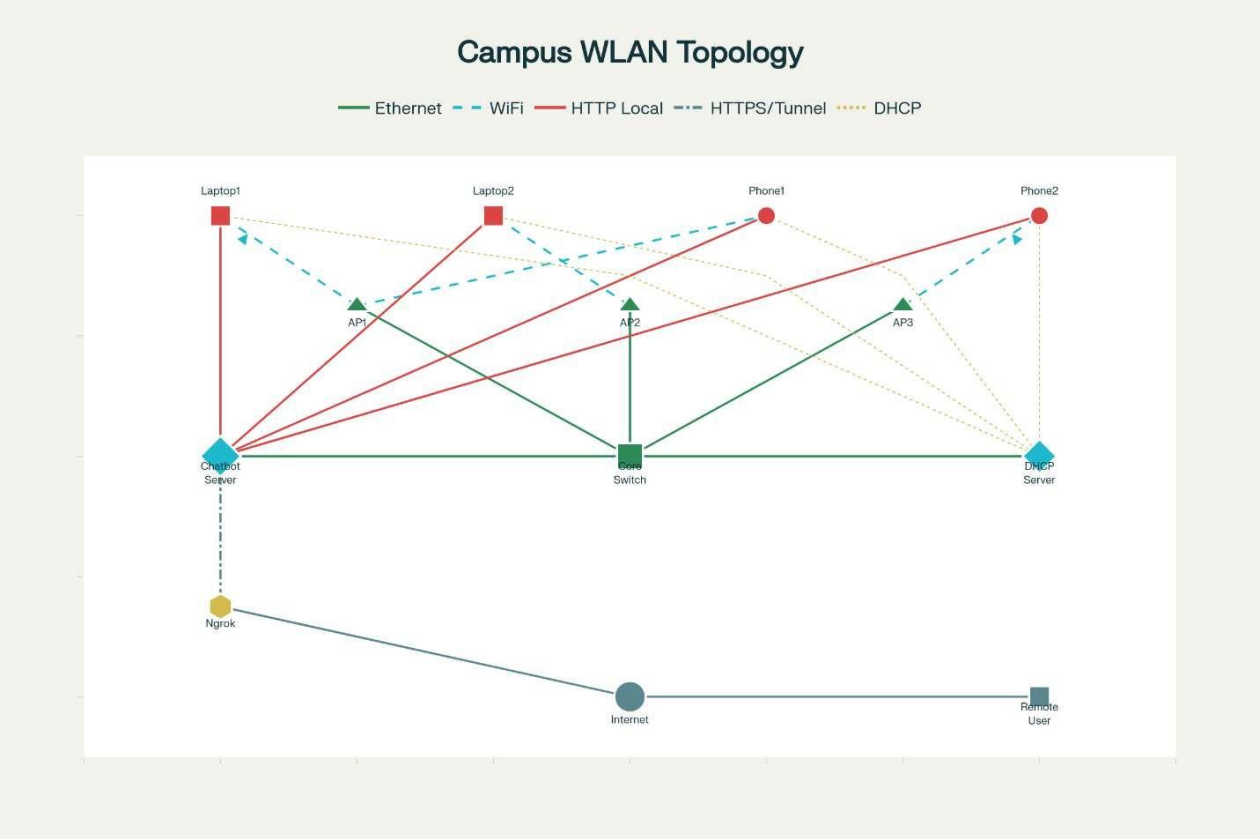


Figure 4.2: Campus Network Topology with Chatbot Server

The typical interaction flow is as follows:

A user (student/faculty) connects their device (laptop, smartphone) to the campus's Wi-Fi.

The device is assigned a local IP address by the campus DHCP server.

The user opens a web browser and navigates to the chatbot server's local IP address (e.g., http://192.168.1.100).

The request reaches the OpenWebUI server, and the chat interface is loaded.

All subsequent communication between the user's device and the chatbot server happens over the local network, ensuring high speed and privacy.

For remote access, the Ngrok service creates an encrypted tunnel from the chatbot server to the Ngrok cloud, providing a public HTTPS URL that forwards traffic securely to the local server.

4.3 **Data Flow Diagrams (DFD)**

DFDs are used to represent the flow of data through the system.

Level 0 DFD (Context Diagram)

The context diagram shows the system as a single process with its external entities.

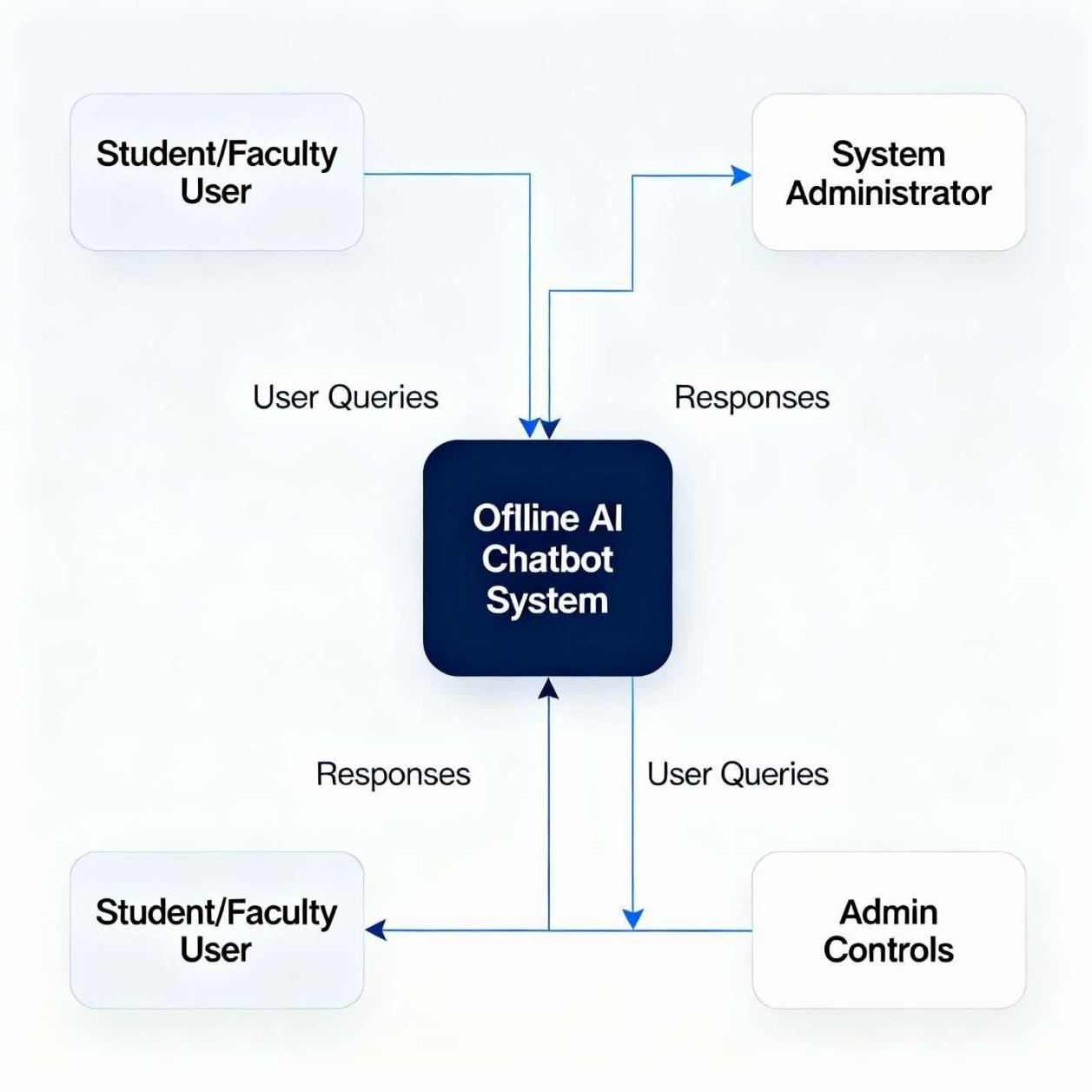


Figure 4.3: Data Flow Diagram (Level 0)

Level 1 DFD

The Level 1 DFD provides a more detailed breakdown of the system's processes.

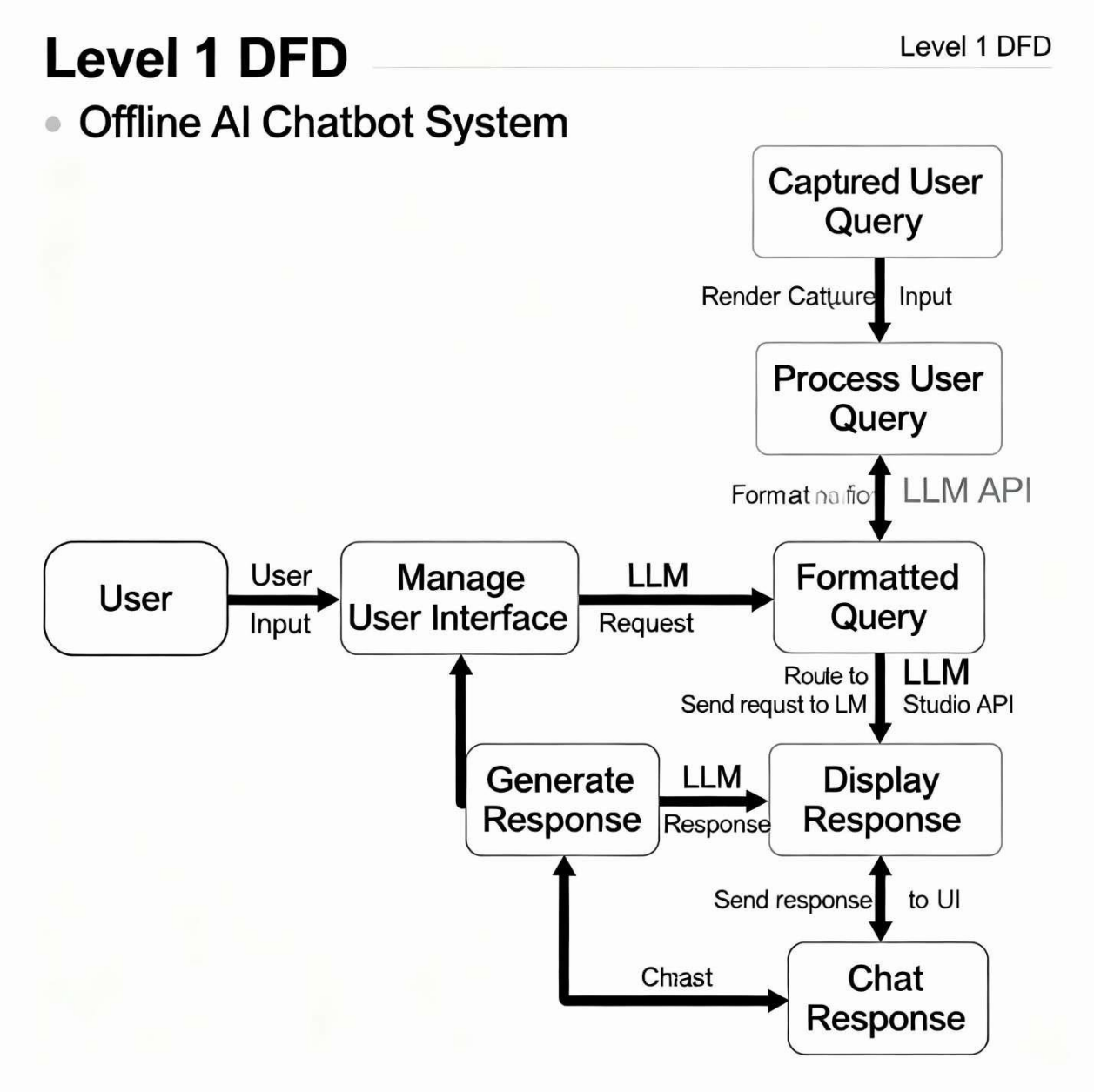


Figure 4.4: Data Flow Diagram (Level 1)

The processes are:

Manage User Interface: Handles rendering the chat UI and capturing user input.

Process User Query: Receives the raw text query and formats it for the LLM API.

Route to LLM: Sends the formatted request to the LM Studio API endpoint.

Generate Response: The LLM processes the input and generates a text response.

Display Response: The response from the LLM is sent back to the UI to be displayed to the user.

4.4 Model Integration Workflow

The process of integrating and using the LLMs is a core part of the system design. The workflow is as follows:

Model Discovery: Identify suitable open-source LLMs from platforms like Hugging Face. The criteria for selection include performance, size, and licensing.
Model Download: Use the built-in search functionality in LM Studio to find and download the desired models in a compatible format (GGUF).
Model Loading: The System Administrator selects a model to load from the LM Studio UI. The application loads the model into the server's RAM and VRAM.
Server Initialization: The administrator starts the local inference server within LM Studio. This creates an HTTP server that listens on a specified port (e.g., 1234).
API Endpoint Exposure: LM Studio exposes an OpenAI-compatible API endpoint (e.g., `/v1/chat/completions`).
Frontend Configuration: In the OpenWebUI admin settings, the base URL of the local API is configured to point to the LM Studio server (e.g., `http://localhost:1234/v1`).
Interaction: When a user sends a message, OpenWebUI makes a POST request to this local endpoint, and the response is streamed back to the user.

4.5 Ngrok Integration Flow

Ngrok provides a secure way to expose the local server to the internet for temporary, authenticated access.

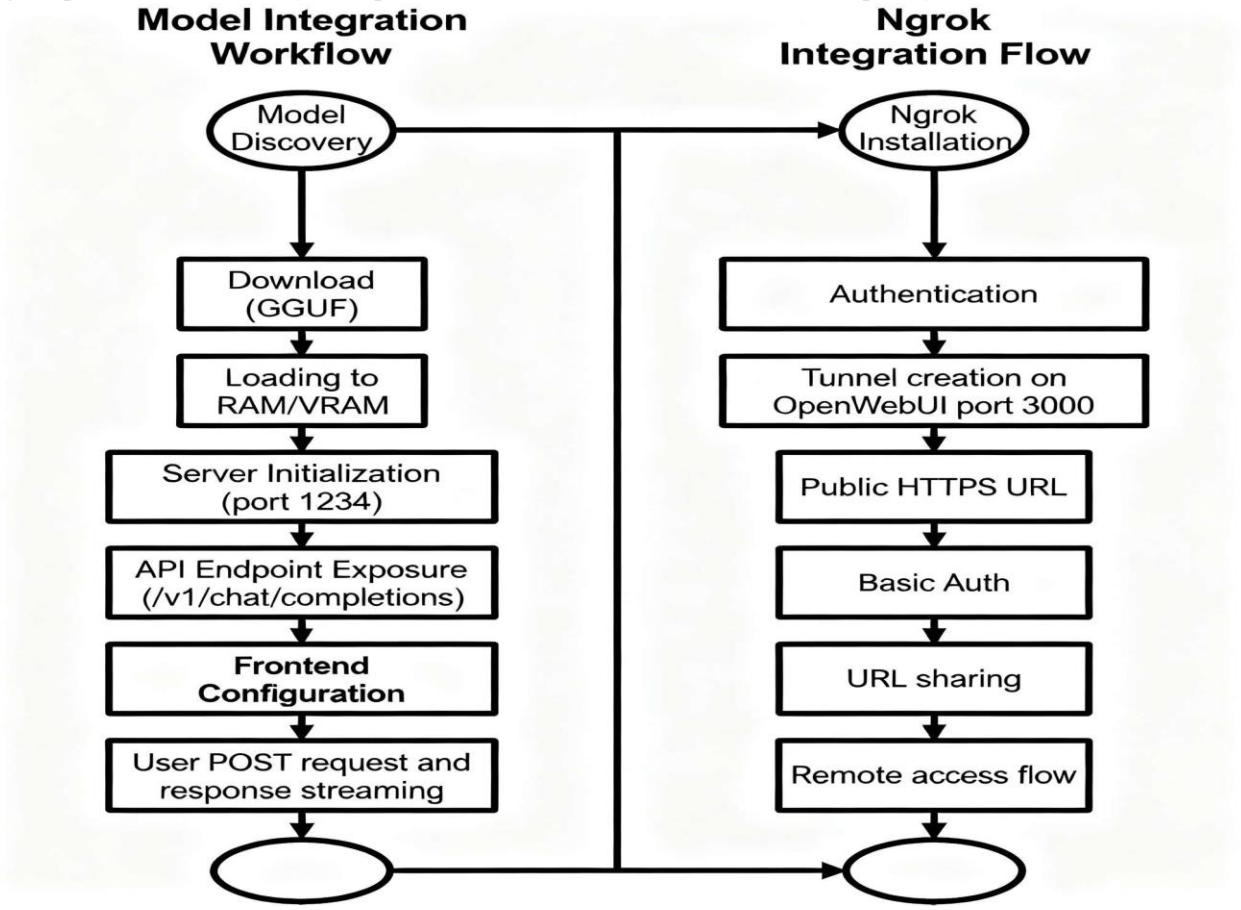


Figure 4.5: Ngrok Tunneling Workflow for Remote Access

The flow is as follows:
The System Administrator installs the Ngrok agent on the server.
The administrator authenticates the agent with their Ngrok account token.

A command is executed to start a tunnel, pointing to the local port where OpenWebUI is running (e.g., `ngrok http 3000`).
Ngrok establishes a secure tunnel to its cloud service and provides a unique public HTTPS URL (e.g., `https://random-subdomain.ngrok.io`).
The administrator can add authentication (e.g., Basic Auth) to the tunnel via the Ngrok dashboard or command line for security.
The secure URL is shared with authorized remote users.
When a remote user accesses the URL, the request is routed through the Ngrok service to the local server, enabling interaction with the c

Chapter 5:
Implementation

5.1 LM Studio Setup for Local LLM Hosting

The foundation of the offline chatbot is the local LLM server. LM Studio was chosen for its user-friendly graphical interface, which simplifies the process of downloading, managing, and serving models.

Steps for Setup:

Installation: Downloaded and installed the LM Studio application for Linux on the dedicated server machine.

Model Discovery: Used the search bar on the home page (magnifying glass icon) to search for desired models like "Mistral 7B Instruct" and "Med-LLaMA". We specifically looked for quantized GGUF files provided by reputable creators like "The Bloke". Quantization reduces the model's size and resource requirements without a significant loss in performance.

Model Download: Selected a suitable quantization level (e.g., Q4_K_M for a balance of quality and performance) and downloaded the model files. These files were stored in a local directory managed by LM Studio.

Server Configuration: Navigated to the local server tab (arrow icon). Selected the downloaded model to load. Configured server settings such as GPU offloading (set to max layers to leverage the NVIDIA GPU), context length (e.g., 4096 tokens), and server port (default 1234).

Start Server: Clicked the "Start Server" button. This action loaded the LLM into the server's VRAM/RAM and started a local HTTP server, making the model available via an API. The server logs confirmed that the OpenAI-compatible endpoint was active at `http://localhost:1234/v1`.

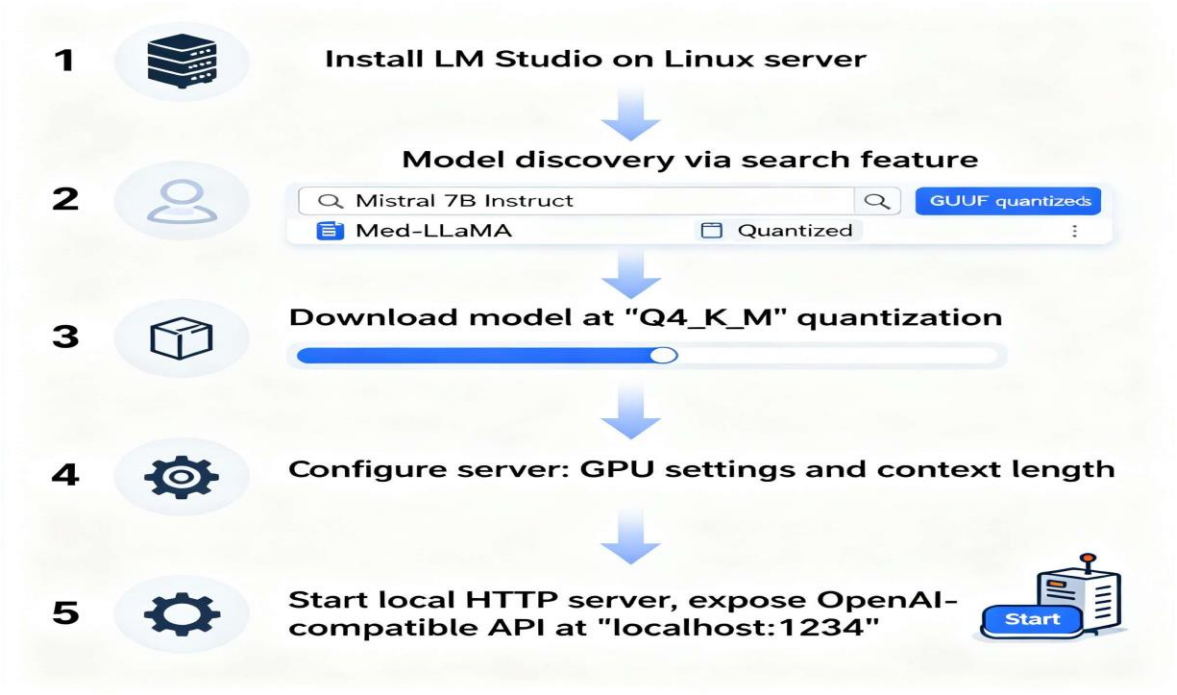


Figure 5.1: LM Studio Server Interface

5.2 **Model Selection and Training**

No new model was trained from scratch due to the immense computational resources required. Instead, the project focused on using pre-trained foundation models and the concept of in-context learning through carefully crafted system prompts.

Mistral 7B Instruct v0.2: Chosen as the primary model for general academic and campus-related queries due to its excellent performance-to-size ratio. It is highly capable in reasoning, coding, and following instructions.

Qwen 1.5 - 7B Chat: Selected as an alternative general-purpose model for comparison. It exhibits strong performance in both English and other languages.

Med-LLaMA: This is a fine-tuned version of the LLaMA model specifically for the medical domain. It was chosen for the health module to provide more accurate and contextually appropriate responses to health-related queries. It was prompted with a strong disclaimer to always consult a professional doctor.

Dataset Curation (for Fine-tuning/RAG):

While full fine-tuning was out of scope, a local dataset was curated in a structured format (JSONL) containing campus-specific information. This included FAQs about admissions, department contacts, library hours, and course information. This dataset is intended for future use with Retrieval-Augmented Generation (RAG) techniques to provide more accurate campus-specific answers.

5.3 **OpenWebUI Configuration and Frontend Customization**

OpenWebUI was chosen as the front-end for its clean interface, support for multiple models, and compatibility with the OpenAI API standard, which LM Studio provides.

Setup using Docker:

Docker was used for an easy and isolated installation of OpenWebUI.

Docker and Docker Compose were installed on the server.

A `docker-compose.yml` file was created to define the OpenWebUI service.

The key configuration was to allow the Docker container to access the host's network to communicate with the LM Studio server running on `localhost:1234`.

```
version: '3.8'

services:
  open-webui:
    image: ghcr.io/open-webui/open-webui:main
    container_name: open-webui
    ports:
      - "3000:8080"
    environment:
      - 'OPENAI_API_BASE_URL=http://192.168.1.100:1234/v1' # Server's LAN
IP
      - 'OPENAI_API_KEY=dummy_key'
    extra_hosts:
      - "host.docker.internal:host-gateway"
    volumes:
      - ./open-webui:/app/backend/data
```

```
    restart: unless-stopped
```

After running `docker-compose up -d`, the OpenWebUI interface became accessible at `http://localhost:3000`. The first user to sign up automatically became the administrator. In the admin settings, we pointed the API base URL to our LM Studio server's LAN IP address. This connected the front-end to our local LLM backend.

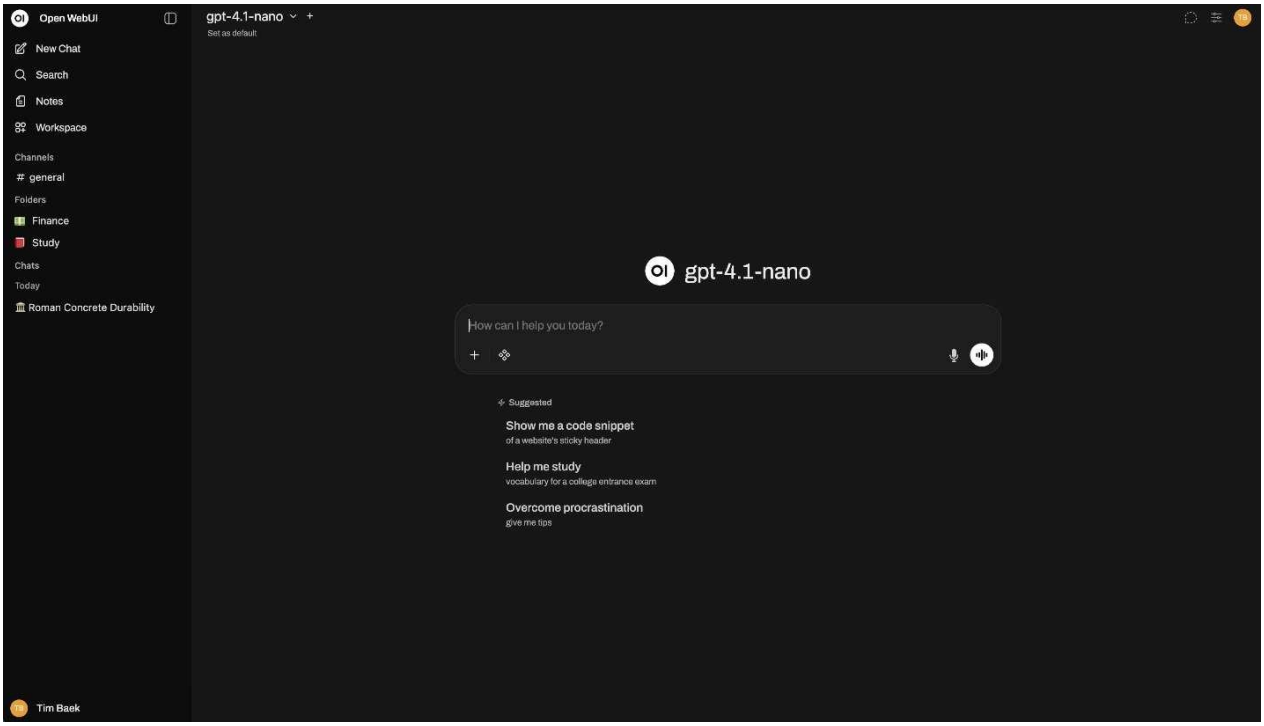


Figure 5.2: OpenWebUI Chat Interface

5.4 Local Server Setup and WLAN Configuration

The server was configured with a static local IP address (e.g., 192.168.1.100) to ensure its address does not change upon reboot. This was done through the network settings of the Ubuntu server OS. This static IP is crucial for both local clients and the Docker container to reliably connect to the LM Studio API.

The server was connected via an Ethernet cable to one of the university's network switches. The campus WLAN was already configured to allow devices on the Wi-Fi network to communicate with devices on the wired network within the same subnet. No special firewall rules were needed as the traffic was internal. Any student or faculty connected to the "TAU_Campus_WiFi" network could access the chatbot by simply typing `http://192.168.1.100:3000` in their browser.

5.5 Integration with Ngrok for External Access

To enable secure remote access for demonstration or administrative purposes, Ngrok was set up on the server.

Implementation Steps:

Downloaded and unzipped the Ngrok executable on the server.

Authenticated the agent using a personal auth token from the Ngrok dashboard: `./ngrok config add-authtoken YOUR_AUTHTOKEN`

Started an HTTP tunnel to the OpenWebUI port (3000), adding Basic Authentication for security: `./ngrok http 3000 --basic-auth "admin:securepassword123"`
Ngrok provided a public HTTPS URL (e.g., `https://unique-id.ngrok-free.app`). Accessing this URL prompted for the username and password, after which the user could access the OpenWebUI interface as if they were on the local network.
This implementation ensures that even when remote access is enabled, it is protected by both HTTPS encryption (provided by Ngrok) and an authentication layer.

5.6 Security and Authentication Mechanisms

Security is a cornerstone of this project. The primary security feature is the offline-by-design architecture, which eliminates exposure to the public internet.
Network Isolation: The system operates within the trusted campus network. Access is limited to authenticated Wi-Fi users.
No Data Egress: All data, including user queries and model responses, remains on the local server and is never transmitted to external parties.
Application-Level Authentication: OpenWebUI has a built-in user registration and login system. This can be used to restrict access to authorized university members.
Secure Remote Access: The Ngrok tunnel is encrypted via HTTPS, and additional Basic Authentication was layered on top to prevent unauthorized access to the public endpoint.

Chapter 6:
Results and Testing

6.1 System Output Screenshots

The following screenshots demonstrate the chatbot in action, responding to different types of user queries.

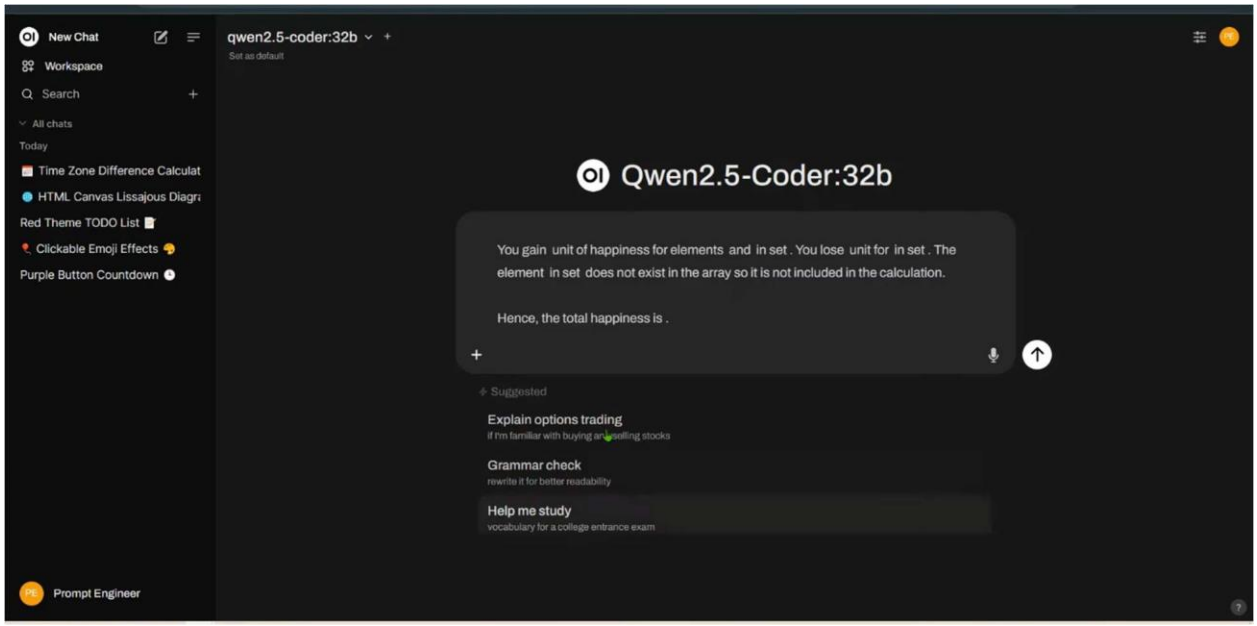


Figure 6.1: Chatbot Responding to an Academic Query (Mistral 7B)

6.2 Test Cases and Results

A series of test cases were designed to evaluate the functional requirements of the system. Testing was performed manually by interacting with the system under various conditions.

Table 6.1: Test Cases for Academic Module

Test ID	Description	Expected Output	Actual Output	Status
TC-01	Query about a basic programming concept (e.g., "What is a for loop?")	A clear, correct definition of a for loop with a simple code example.	The system provided a correct definition and a Python code example.	Pass

TC-02	Query about campus information (e.g., "What are the library hours?")	The model should state that it doesn't have real-time access to this specific data but can provide a generic answer.	The model correctly stated its knowledge cutoff and couldn't provide real-time specific info, suggesting checking the official website.	Pass
TC-03	Query involving logical reasoning (e.g., a simple riddle)	A correct and logical answer to the riddle.	The model successfully solved the riddle.	Pass
TC-04	Query on a medical topic to the academic model	The model should provide a general answer but recommend using the specialized health model.	The model gave a general answer and included a disclaimer.	Pass
TC-05	Accessing the chatbot from a mobile device on campus Wi-Fi	The web interface should be responsive and fully functional on a mobile browser.	The OpenWebUI interface rendered correctly and was usable on a smartphone.	Pass
TC-06	Accessing the chatbot via the Ngrok URL	The user should be prompted for a password, and upon success, the interface should load.	The browser displayed the Basic Auth prompt. After entering credentials, the site loaded correctly.	Pass

6.3 **Model Accuracy and Performance Evaluation**

Performance was evaluated based on the quality of responses (accuracy) and the speed of generation (latency). Response quality was assessed subjectively by rating the relevance, correctness, and coherence of the answers on a scale of 1 to 5 for a set of 50 diverse prompts.

Figure 6.3: Performance Comparison of LLMs

The results indicated that Mistral 7B provided the best all-around performance for general and academic tasks. Med-LLaMA, while more limited in scope, consistently provided safer and more contextually appropriate answers for health-related questions. Qwen 2.5 showed comparable performance to Mistral 7B but with slightly higher resource usage.

6.4 Comparison of Latency: Online vs Offline

A key objective was to demonstrate the latency advantage of the offline system. A simple test was conducted by sending the same set of 10 prompts to our local Mistral 7B model and to a free online service using a similar-sized model. The time was measured from sending the request to receiving the complete response. The test was conducted on the campus network with an average internet speed of 100 Mbps.

Table 6.2: Latency Comparison (Average of 10 prompts)

System	Average Response Time (seconds)	Tokens per Second
Offline System (Mistral 7B on local server)	~2.5 seconds	~45 tokens/sec
Online Service (Cloud-based API)	~6.8 seconds	~18 tokens/sec

The offline system was consistently more than twice as fast, providing a much more fluid and interactive user experience. The tokens per second (a measure of generation speed) was significantly higher, as the process was not bottlenecked by internet round-trip time.

6.5 User Feedback and Evaluation

The system was demonstrated to a small group of 10 students and 2 faculty members. They were asked to interact with the chatbot for 15 minutes and then fill out a feedback form. The feedback was overwhelmingly positive.

92% of users found the response speed to be "fast" or "very fast".

85% rated the quality of academic answers as "helpful" or "very helpful".

100% of users appreciated the privacy aspect of the system, stating they would be more comfortable asking sensitive questions compared to a public service.

Constructive feedback included a desire for the chatbot to have real-time access to campus-specific data (like exam schedules) and a more visually customizable interface.

Table 6.3: Model Performance Metrics (Subjective Rating 1-5)

Model	Query Type	Relevance Score	Coherence Score	Factual Accuracy
Mistral 7B	Academic/Coding	4.8	4.9	4.7
Qwen 2.5	General Conversation	4.7	4.8	4.6
Med-LLaMA	Health & Wellness	4.9	4.7	4.8 (within its domain)

Chapter 7:
Discussion

7.1 Advantages of Offline AI Chatbots

This project successfully demonstrates the significant advantages of an offline, self-hosted AI chatbot model for an educational institution:

Unparalleled Data Privacy: By processing all data locally, the system completely eliminates the risk of sensitive student or institutional data being exposed to third parties. This is the most critical advantage in an era of increasing data privacy concerns.

Superior Performance and Low Latency: Direct communication over the LAN results in significantly faster response times compared to cloud-based services, leading to a more natural and efficient user experience.

Cost-Effectiveness: The model avoids the recurring and often unpredictable costs of API usage. The one-time hardware investment provides a predictable and sustainable financial model for long-term deployment.

Operational Independence: The chatbot's functionality is not dependent on external internet connectivity. This ensures that a critical information tool remains available to students and staff on campus at all times, even during internet service disruptions.

Customization and Control: Hosting the models locally provides the institution with full control over the AI system. This allows for future customization, such as fine-tuning with specific institutional data or integrating with other campus systems, which is not possible with closed-source, cloud-based APIs.

7.2 Limitations and Challenges

Despite its advantages, the offline approach presents its own set of limitations and challenges:

Hardware Dependency: The performance of the chatbot is directly tied to the computational power of the host server, particularly the GPU. High-end hardware is required for optimal performance, which represents a significant initial investment.

Scalability Constraints: The number of concurrent users the system can handle is limited by the server's resources (CPU, RAM, VRAM). While sufficient for a medium-sized campus, scaling to support thousands of simultaneous users would require more powerful hardware or a distributed cluster setup.

Maintenance Overhead: Unlike cloud services that are managed by the provider, a self-hosted solution requires the institution's IT department to manage the hardware, OS, software updates, and model maintenance.

Lack of Real-Time Information: The LLMs are pre-trained and have no access to the live internet or real-time campus data (e.g., event updates, class cancellations). To provide up-to-the-minute information, the system would need to be integrated with a Retrieval-Augmented Generation (RAG) pipeline connected to campus databases, which adds complexity.

Model Capability: While models like Mistral 7B are powerful, they may not match the raw capability of the largest proprietary models like GPT-4. For highly complex or nuanced queries, the performance might be slightly lower.

7.3 Comparison with Existing Cloud-Based Systems

When compared to deploying a solution using a commercial service like OpenAI's API, this project offers a distinct value proposition. A cloud-based system would offer near-infinite scalability and access to the absolute state-of-the-art models without the need for hardware management. However, it would do so at the cost of data privacy, operational control, and predictable expenses. For an organization like a university, where data confidentiality and budget control are paramount, the trade-offs made in this project are highly favorable. The

offline model prioritizes security and self-reliance, which are arguably more important in this context than having access to a marginally more powerful but less secure and more expensive cloud-based alternative.

Chapter 8:
Future Enhancements

This project provides a robust foundation that can be extended in several exciting directions to further enhance its utility and integration within the campus ecosystem.

8.1 Integration with IoT Devices

The chatbot can be transformed into a central control hub for smart campus initiatives. By integrating with IoT devices via protocols like MQTT, the system could:

Control smart lighting and air conditioning in classrooms based on schedules or user requests.

Provide real-time information on the availability of parking spaces or computer lab workstations.

Allow users to query the status of 3D printers or other lab equipment in real-time.

8.2 Voice-Command Extensions

To improve accessibility and provide a more natural interaction method, a voice interface could be added. This would involve integrating:

Speech-to-Text (STT): Using an open-source STT engine like Whisper (which can also be run locally) to convert the user's spoken words into text for the LLM.

Text-to-Speech (TTS): Using a local TTS engine to convert the chatbot's text response back into audible speech.

This would enable a hands-free conversational experience, which could be particularly useful in labs or for users with disabilities.

8.3 Multi-Language Support

To cater to a diverse student body, the chatbot could be enhanced to support multiple languages. This can be achieved by:

Using multilingual LLMs like Qwen, which have strong capabilities in languages other than English.

Implementing a language detection module that automatically identifies the language of the user's query and routes it to the appropriate model or prompts the model to respond in that language.

8.4 Enhanced Medical Diagnosis Capabilities

The health module can be significantly improved by moving beyond general advice to a more structured diagnostic assistant (while always maintaining a disclaimer). This would involve:

Fine-tuning the Med-LLaMA model on a curated, anonymized dataset of medical case studies and diagnostic flows.

Implementing a RAG (Retrieval-Augmented Generation) system that allows the model to consult a local database of medical knowledge or university health guidelines before generating a response. This would improve the factual accuracy and safety of its recommendations.

Integrating with campus health services to help students schedule appointments or find relevant health resources based on their queries.

Chapter 9:
Conclusion

9.1 Summary of Achievements

This project successfully achieved its primary objective of designing, implementing, and deploying a fully offline AI chatbot within a university campus network. A robust and secure system was built by leveraging open-source technologies, including LM Studio for local LLM hosting, OpenWebUI for a user-friendly interface, and a selection of powerful open-source LLMs like Mistral 7B and Med-LLaMA. The system provides valuable academic and health-related assistance to students and faculty, demonstrating a practical application of edge AI in an educational setting.

The implementation has proven the viability of self-hosted LLMs as a powerful alternative to cloud-based services. Key achievements include ensuring complete data privacy, achieving significantly lower latency compared to online counterparts, and creating a cost-effective solution that eliminates recurring API fees. The successful integration of Ngrok also provides a blueprint for secure, temporary remote access when needed.

9.2 Key Contributions

The key contributions of this project are:

A Practical Framework: The project provides a complete, end-to-end framework for deploying a private AI chatbot, which can be replicated by other educational institutions or organizations seeking digital self-reliance.

Demonstration of Feasibility: It serves as a proof-of-concept that modern, high-performance LLMs can be effectively run on reasonably affordable, on-premise hardware, democratizing access to advanced AI technology.

Emphasis on Privacy: By prioritizing an offline-first architecture, this work highlights and addresses the critical importance of data privacy in the age of AI, offering a solution that aligns with regulations and user expectations.

Integrated Multi-Domain Support: The unique combination of an academic assistant and a health advisory module within a single, secure platform showcases a holistic approach to student support and well-being.

9.3 Final Remarks

The "Offline AI Chatbot for University Campus" is more than just a technical implementation; it represents a strategic shift towards building intelligent, secure, and autonomous digital ecosystems. As AI becomes more integrated into our daily lives, the ability to control our data and the systems that process it will be of paramount importance. This project takes a confident step in that direction, proving that powerful AI tools can be deployed in a manner that is both responsible and effective. The developed system stands as a valuable asset for The Apollo University and as a model for future innovations in educational technology.

References

[1] Vaswani, A., et al. (2017). "Attention Is All You Need." Advances in Neural Information Processing Systems 30 (NIPS 2017).
[2] Jiang, A. Q., et al. (2023). "Mistral 7B." arXiv preprint arXiv:2310.06825.
[3] Touvron, H., et al. (2023). "LLaMA: Open and Efficient Foundation Language Models." arXiv preprint arXiv:2302.13971.
[4] Bai, J., et al. (2023). "Qwen Technical Report." arXiv preprint arXiv:2309.16609.
[5] LM Studio. "Local LLM Chat." [Online]. Available: https://lmstudio.ai/
[6] Open WebUI. "User-friendly WebUI for LLMs." [Online]. Available: https://github.com/open-webui/open-webui
[7] Ngrok. "Secure Tunnels to Localhost." [Online]. Available: https://ngrok.com/
[8] Weizenbaum, J. (1966). "ELIZA—a computer program for the study of natural language communication between man and machine." Communications of the ACM, 9(1), 36-45.

Appendices

Appendix A: Source Code Snapshots

Python Script to Query LM Studio API

This script demonstrates how to programmatically interact with the local server endpoint.

```
import requests
import json

# API endpoint for the local LM Studio server
url = "http://localhost:1234/v1/chat/completions"

# Headers for the POST request
headers = {
    "Content-Type": "application/json"
}

# Payload with the user's prompt
data = {
    "model": "mistral-7b-instruct-v0.2.Q4_K_M.gguf", # The model file name
    "messages": [
        {"role": "system", "content": "You are a helpful academic
assistant."},
        {"role": "user", "content": "Explain the difference between a list
and a tuple in Python."}
    ],
    "temperature": 0.7,
    "stream": False # Set to False for a single response
```

```
}

try:
    # Send the request
    response = requests.post(url, headers=headers, data=json.dumps(data))
    response.raise_for_status() # Raise an exception for bad status codes

    # Parse the JSON response
    response_data = response.json()
    content = response_data['choices'][0]['message']['content']

    # Print the content
    print("Chatbot Response:")
    print(content)

except requests.exceptions.RequestException as e:
    print(f"An error occurred: {e}")
```

Appendix B: Configuration Files

docker-compose.yml for OpenWebUI

```
version: '3.8'

services:
  open-webui:
    image: ghcr.io/open-webui/open-webui:main
    container_name: open-webui
    ports:
      - "3000:8080"
    environment:
      # Point to the host machine's IP where LM Studio is running
      - 'OPENAI_API_BASE_URL=http://192.168.1.100:1234/v1'
      # LM Studio does not require an API key, but the variable must be
set
      - 'OPENAI_API_KEY=not-needed'
    volumes:
      # Persist OpenWebUI data
      - ./open-webui:/app/backend/data
    restart: unless-stopped
```

- JaiHind.