AI-Powered Chatbot with Streamlit

Python Internship Project

Blend Vidya Edtech

A Dynamic and leading force in the Educational Technology (Edtech) sector,
Approved by AICTE New Delhi,
Kadubeesanahalli, Panathur, Bengaluru, Karnataka 560103

Submitted by-
Sree

Table of contents

[1] Abstract:

This project centers around the development of an AI-powered chatbot that combines modern natural language processing techniques with a user-friendly interface. The chatbot is built using **Streamlit** for the front-end user interface, providing an easy-to-navigate platform for users to interact with the system. For processing and interpreting user inputs, the chatbot leverages **Ollama's Llama3.1 model**, a state-of-the-art large language model (LLM) known for its ability to handle natural language queries with a high degree of accuracy. This integration enables the chatbot to offer intelligent, contextually relevant responses to a wide range of user questions, making it a highly adaptable and dynamic tool.

The primary goal of the project is to create a seamless interaction experience for users by combining advanced language understanding with a straightforward interface. The chatbot is designed to not only answer queries but also provide meaningful, conversational responses, mimicking human interaction. By incorporating a robust language model and an intuitive UI, the project aims to bridge the gap between complex AI technologies and end-user accessibility, ensuring that users with varying technical skills can easily engage with the chatbot.

Beyond offering simple responses, this chatbot is also designed to be flexible and scalable, capable of being expanded with additional features such as personalized learning, advanced querying, or integration with other systems. The project focuses heavily on the optimization of both the AI's response generation and the efficiency of the interface, ensuring that interactions are swift and accurate. Key objectives include the construction of a clean and responsive chatbot interface, the effective utilization of advanced language models, and the enhancement of user experience by ensuring fast and smooth interactions.
The project's deliverables are outlined in a comprehensive report, which details the methodology employed in the development process, the code structure that supports the chatbot's functionality, and the results achieved through its implementation. Additionally, the report highlights potential improvements, such as expanding the model's capabilities, optimizing performance for real-time interactions, and incorporating feedback mechanisms. Future improvements might also include expanding the chatbot's functionality to cater to more specialized use cases or improving the model's ability to handle more complex conversational flows.

[2] Introduction:

In today's digital world, chatbots play a vital role in customer support, virtual assistance, and automated conversations. The primary goal of this project is to develop a chatbot that can respond to a variety of user queries, offering help and information on various topics. While AI chatbots have become common, developing a user-friendly, responsive interface integrated with a powerful language model can greatly enhance the user experience.

This project utilizes Streamlit to build an interactive front-end and Ollama's llama3.1 model as the core natural language processing engine. Langchain is employed to manage prompt chaining and output parsing. The chatbot is designed for general-purpose use and can be adapted for more specific domains like customer service, information retrieval, and educational assistance.

[2.1] Problem Statement:

There is a growing need for simple, effective, and interactive AI systems that can be deployed across various platforms for answering user queries. This project aims to solve this by:

1. Creating a responsive chatbot interface.
2. Integrating a powerful LLM to handle complex queries.
3. Ensuring that the system is easy to deploy and scale for various use cases.

[2.2] Objectives:

- Develop a chatbot interface using Streamlit.
- Integrate Ollama's LLM for query processing and intelligent responses.
- Implement a system that is user-friendly, interactive, and extendable.

This project not only focuses on creating a functional chatbot but also emphasizes the importance of user experience and scalability. By utilizing Streamlit for the front-end and Ollama's advanced LLM for natural language processing, the system is designed to be easily deployable across multiple platforms. The chatbot is highly adaptable, capable of handling a broad spectrum of queries, and can be customized for different domains such as customer service, educational assistance, or even industry-specific support. This flexibility ensures that the project can serve as a foundation for more complex AI-driven applications in the future.

[3] Methodology:

This section outlines the technical approach and tools used to build the chatbot, as well as the detailed steps followed during the development process.
Technologies and Libraries Used:

- **Streamlit**: A Python library used to create a simple and interactive web interface for the chatbot.
- **Langchain**: A framework for managing language model prompts and chaining LLMs with other tools, such as parsers and APIs.
- **Ollama LLM (llama3.1)**: A large language model used for generating responses to user queries.
- **StrOutputParser**: A parser used to handle and process output from the LLM.

Steps:

- **Frontend (User Interface):**
  - Streamlit provides the user with a clean and simple interface where they can input their query via a text box.
  - st.text_input is used to capture the user input, and once the user presses Enter, the query is passed into the backend system.
- **Prompt Template:**
  - A ChatPromptTemplate is defined to structure the conversation. This template consists of system messages (instructions to the chatbot) and user messages (the query).
  - The system message instructs the chatbot to act as a helpful AI assistant.
  - The user message dynamically injects the input query into the conversation template.
- **LLM (Ollama Model):**
  - The prompt is passed to the Ollama LLM, which is tasked with processing the input using its llama3.1 model. This model is a large language model designed for generating human-like responses.
- **Output Parsing:**
  - The LLM response is captured and passed through the StrOutputParser, which extracts the string response from the model's output.
  - The final parsed output is displayed back to the user using st.write in the Streamlit interface.

The development of the chatbot follows a systematic process involving several key components. These include the user interface built with Streamlit, the integration of Ollama's Llama3.1 model for processing natural language inputs, and the utilization of Langchain for prompt chaining and output parsing.

The flow of data between these components is illustrated in the EER diagram below (Figure 1). The diagram outlines the relationship between the user queries, the language model's processing, and the final response generation.

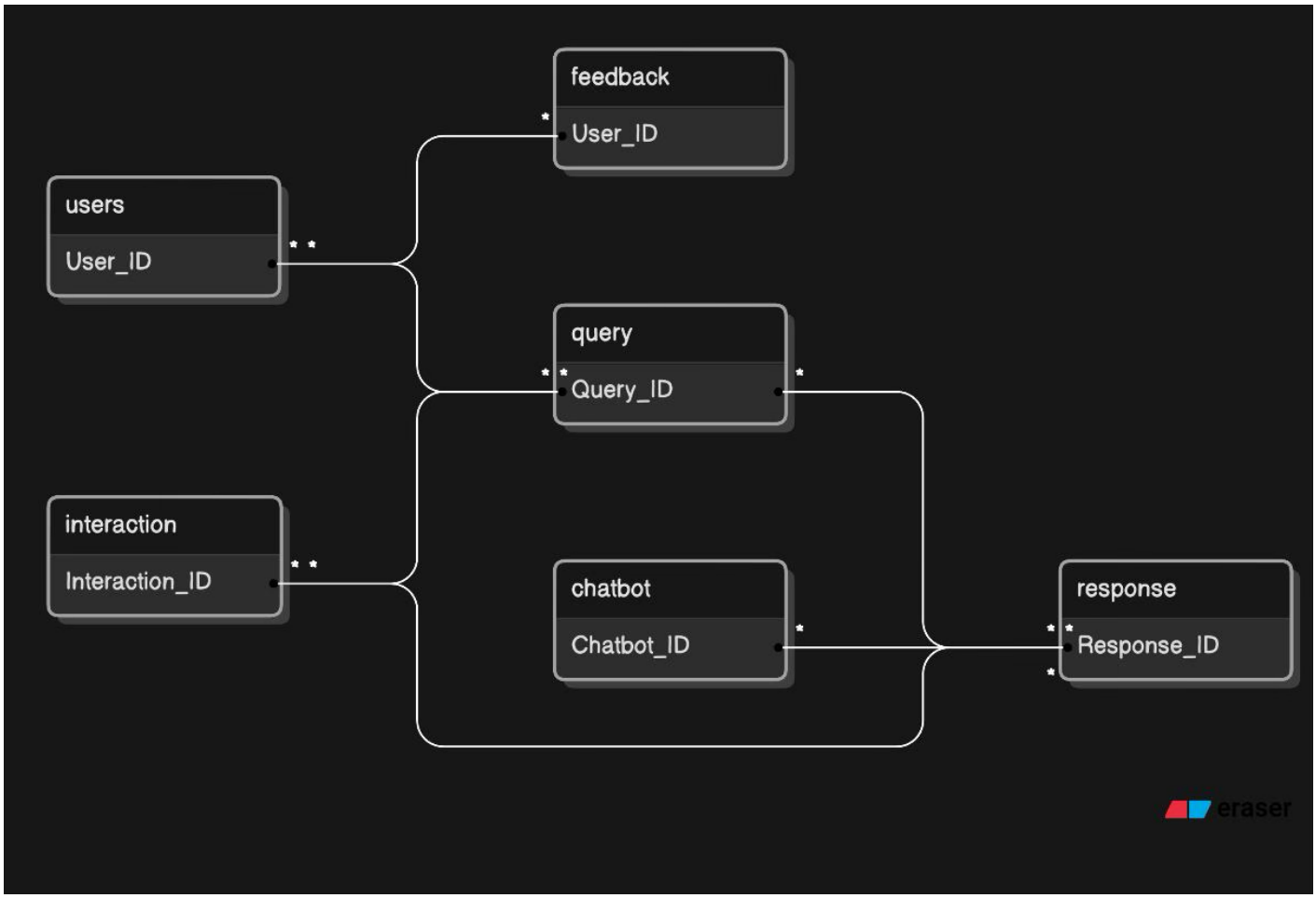

Fig: 1 Flow chat

This approach will help provide a clear, structured view of our system.

[4] Code Explanation:

```
from langchain_core.prompts import ChatPromptTemplate
from langchain_core.output_parsers import StrOutputParser
from langchain_community.llms import Ollama
import streamlit as st

# Title of the Streamlit app
st.title("Sree's Chat Bot")

# Input field for the user to type in their query
input_txt = st.text_input("Please enter your queries here…")

# Template to structure the conversation
prompt = ChatPromptTemplate.from_messages([
   ("system", "you are a helpful AI assistant."),
   ("user", "user query: {query}")
])

# Load the Llama model using Ollama package
llm = Ollama(model="llama3.1")

# Parse the output of the LLM into a simple string format
output_parser = StrOutputParser()

# Chain of actions: prompt -> LLM -> output parser
chain = prompt | llm | output_parser

# If there's any input from the user, process the query and display the response
if input_txt:
   st.write(chain.invoke({"query": input_txt}))
```

Explanation of Key Functions:

- **st.text_input**: Creates a text input box where users can enter their query.
- **ChatPromptTemplate**: Creates a structured conversation format where a system message and user message are passed to the LLM. The system message defines the chatbot's role.
- **Ollama LLM**: This is where the LLM (llama3.1) processes the structured prompt and generates a response based on the user query.

StrOutputParser: This parses the response from the LLM into a human-readable string that can be displayed on the Streamlit interface.
chain.invoke(): The final step of processing the query by passing it through the defined prompt, LLM, and output parser.

[4.1] Entities in the Chatbot System:

- **User:**

  - Attributes:
    - User_ID (Primary Key): A unique identifier for the user interacting with the chatbot.
    - Name: Name of the user (optional).
    - Query: The question or statement provided by the user.
    - Timestamp: The time the query was sent.
  - **Description**: This entity represents the users who interact with the chatbot by entering queries.

- **Chatbot:**

  - Attributes:
    - Chatbot_ID (Primary Key): A unique identifier for the chatbot instance.
    - Model: The language model used by the chatbot (e.g., llama3.1).
    - Status: The status of the chatbot (active, inactive).
  - **Description**: This entity represents the AI-powered chatbot that processes queries and provides responses.

- **Query:**

  - Attributes:
    - Query_ID (Primary Key): A unique identifier for each query input by the user.
    - Text: The actual text of the query submitted by the user.
    - Language: The language of the query.
    - Timestamp: The time when the query was submitted.
  - **Description**: This entity holds details about the queries submitted by the user.

Interaction:

- Attributes:
  - Interaction_ID (Primary Key): A unique identifier for each user-chatbot interaction session.
  - Start_Time: The time the interaction started.
  - End_Time: The time the interaction ended.
  - Query_Count: The number of queries exchanged in this interaction.
- **Description**: This entity captures the interaction sessions between the user and the chatbot, including the time of the session and the number of queries.

[4.2] Relationships:

1. **User Submits Query:**
   - **Cardinality**: One user can submit many queries.
   - **Description**: This relationship links the User entity with the Query entity, signifying that a user submits a query to the chatbot.

2. **Query Is Processed By Chatbot:**
   - **Cardinality**: One query is processed by one chatbot instance, but a chatbot can process many queries.
   - **Description**: This relationship represents the connection between the Query and Chatbot entities, indicating that each query is processed by a chatbot instance.

3. **Chatbot Generates Response:**
   - **Cardinality**: One query can result in one response, but one chatbot can generate many responses.
   - **Description**: This relationship shows that the chatbot provides a response for each query submitted by a user.

4. **Interaction Between User and Chatbot:**
   - **Cardinality**: One user can have many interactions with the chatbot, and each interaction session involves many queries and responses.
   - **Description**: This relationship captures the broader interaction between the user and chatbot, encompassing multiple queries and responses in a single session.

[5] Results:

The chatbot was successfully developed and responds intelligently to user queries. Below are some screenshots and examples of interactions with the bot:

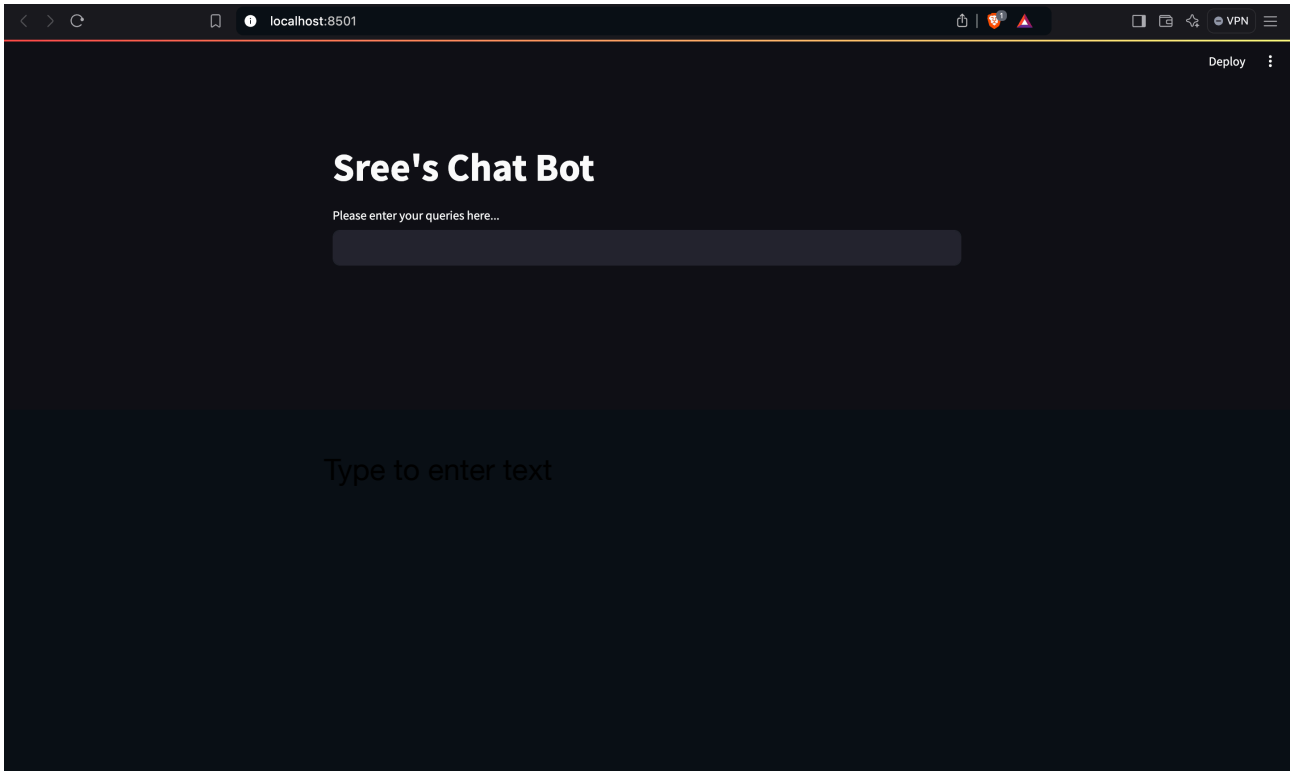

Fig: 2 User Interface

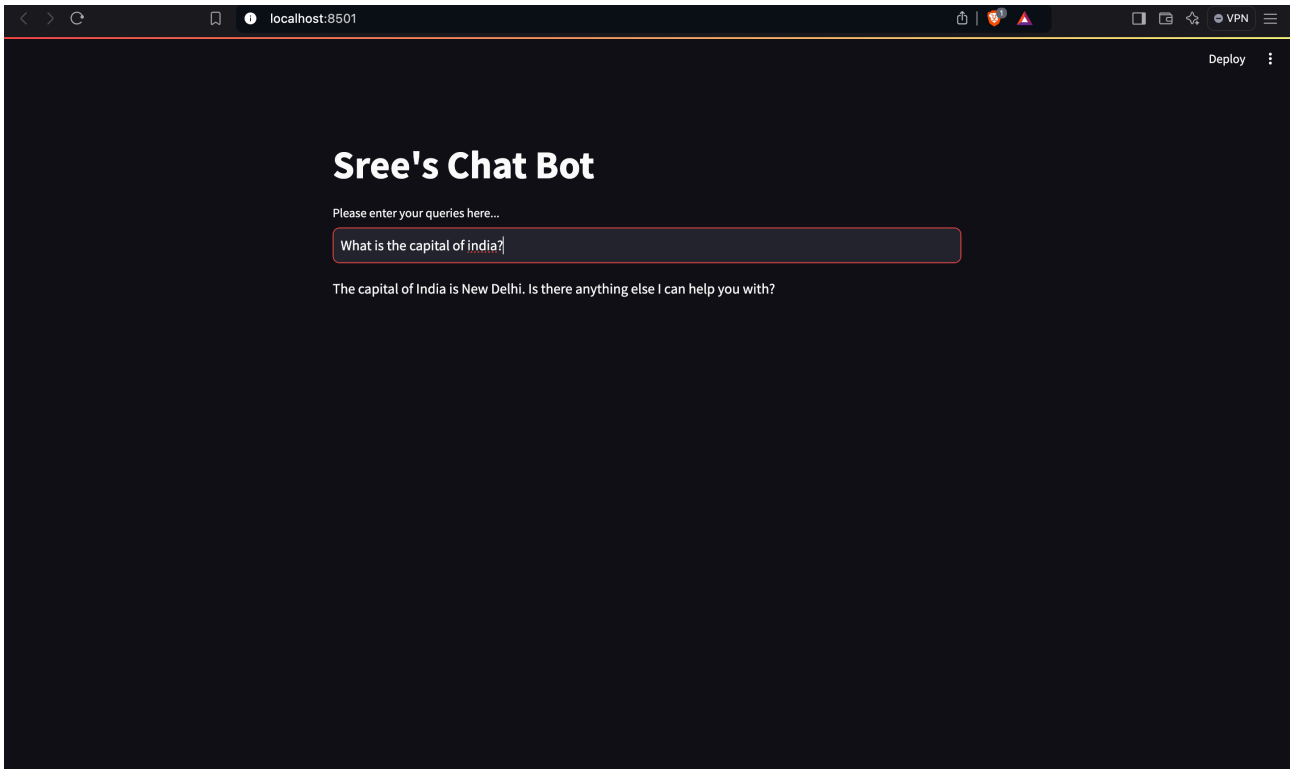

Fig: 3 Test Case 01

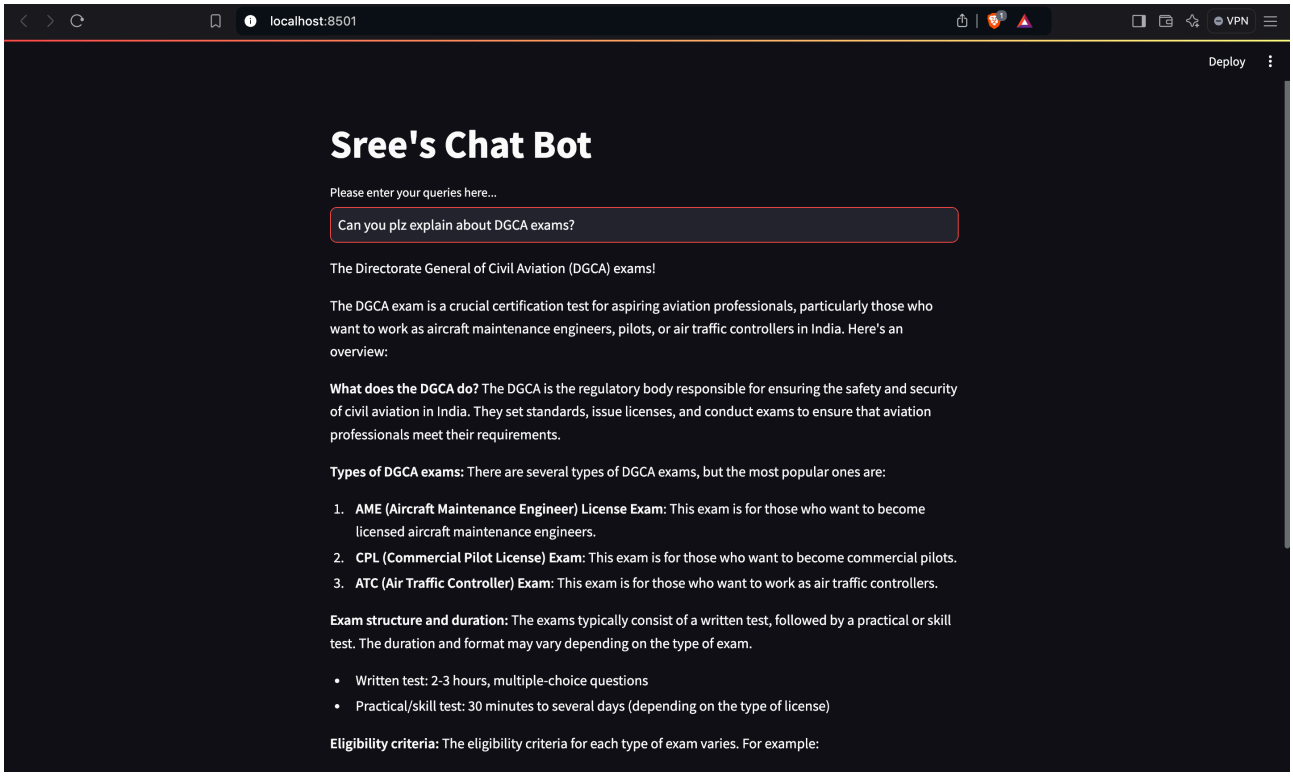

Fig: 4 Test Case 02

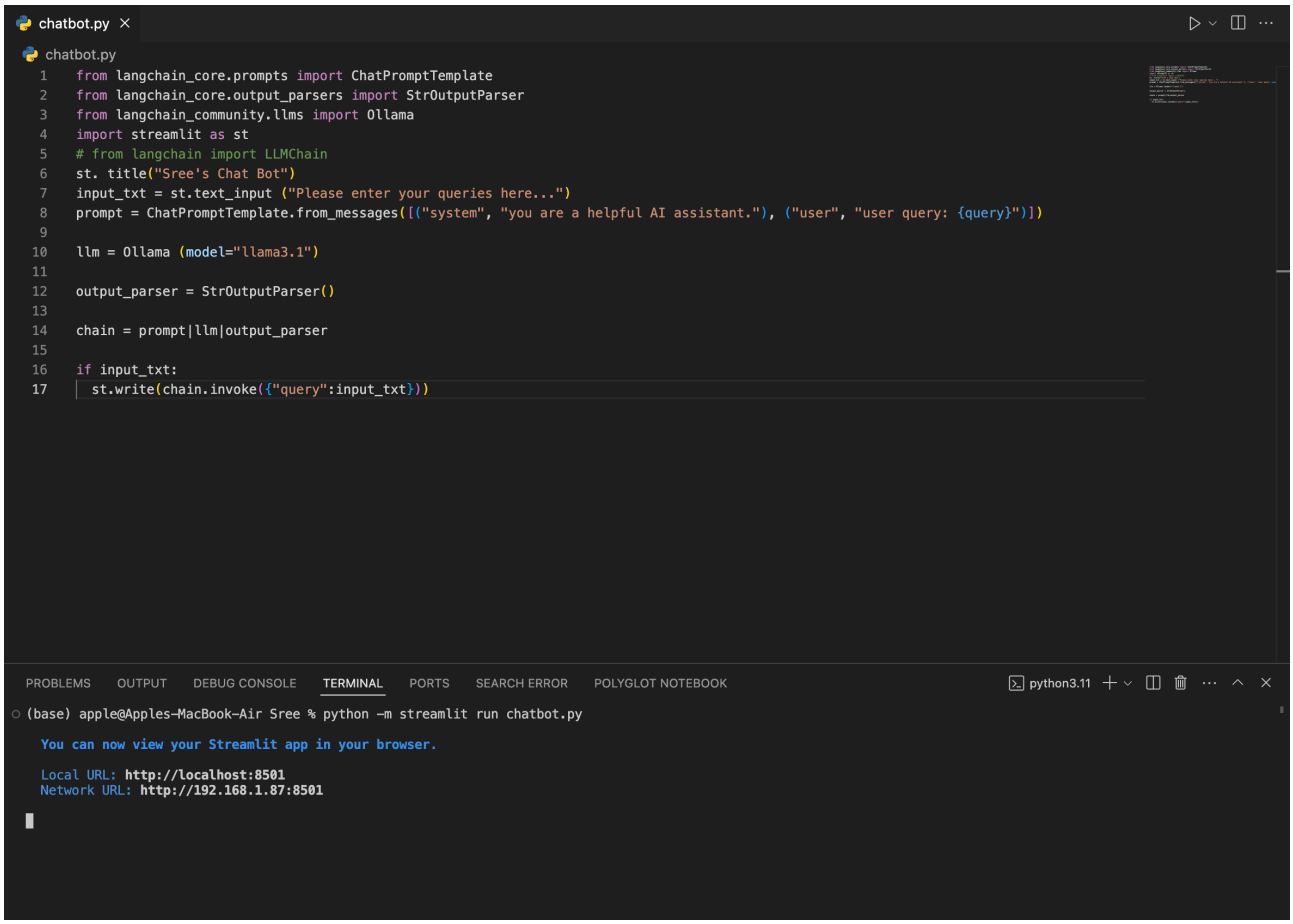

Fig: 5 Code Statement

[6] Conclusion:

The project succeeded in implementing a functional AI chatbot using Streamlit and Ollama. The chatbot is capable of processing and responding to user queries intelligently, thanks to the integration of a powerful LLM model. The system is scalable and can be extended with more features like:

1. Storing user interaction history for better context-aware responses.
2. Adding specialized domains like customer service, healthcare, or technical support.
3. Implementing real-time API calls for queries that require current data, such as weather or stock prices.

[6.1] Future Improvements:

- **Memory and Context**: Adding memory to retain conversation context.
- **Real-time API Integration**: For real-time data like news, stock updates, or weather.
- **Natural Language Understanding**: Improving the model's ability to handle more complex queries and follow-up questions.

[7] References:

1. Streamlit documentation: https://docs.streamlit.io/
2. Langchain documentation: https://langchain.readthedocs.io/
3. Ollama (LLama3.1) documentation
4. Python documentation: https://docs.python.org/3/