Emotional Sentiment Analysis and Adaptive Response System

Overview:

This project implements a conversational chatbot using LangChain, Streamlit, and Ollama's Llama model. The chatbot can engage in various topics, making it versatile and adaptable for different conversational contexts. The model uses **Llama** from Ollama as the backbone of its natural language understanding.**LangChain** is used to orchestrate interactions and integrate with multiple data sources and APIs. **Streamlit** serves as the user interface (UI) for easy and interactive usage of the chatbot.

This project is a conversational chatbot that integrates the powerful **Llama model from Ollama**, **LangChain** for orchestrating data flow, and **Streamlit** for creating an interactive user interface. The chatbot is designed to handle a wide variety of topics, such as technology, weather, entertainment, and more, while engaging in dynamic, context-aware conversations. By leveraging **LangChain**, the chatbot can integrate with external APIs for real-time information, maintain memory for personalized interactions, and adapt to various conversation scenarios. With **Streamlit**, users can easily interact with the chatbot through a simple, responsive interface. The combination of these technologies allows for a versatile, engaging, and intelligent chatbot experience, ideal for applications like customer support, personal assistance, education, and entertainment.

Technology Stack:

- **Ollama Llama**: A high-performance language model for various NLP tasks.
- **LangChain**: A framework for developing applications with large language models (LLMs). LangChain helps integrate the chatbot with data sources, APIs, and other tools.
- **Streamlit**: A Python framework to quickly build interactive web applications. Streamlit will provide a frontend interface for users to interact with the chatbot.
- **Pycharm**: IDE used for the implementation of the code.

Installation

1. **Install Python and Dependencies:**

Ensure you have Python installed (preferably Python 3.8 or later). Install the required libraries

- langchain_openai
- langchain_core
- python-dotenv
- streamlit
- langchain_community

Command to install:

pip install langchain

pip install streamlit

pip install python-dotenv

pip install langchain_community

import os

2. **Set up the Ollama Llama Model:**

- Download and install the Ollama Llama model following the instructions provided by Ollama.
- Make sure the Ollama client is properly set up on your machine.

Run the Streamlit App:

After installing the necessary packages and setting up your model, you can run the

Streamlit app:

streamlit run app.py

.

Project Structure

```
├── app.py              # Main Streamlit application script
├── chatbot.py          # Logic for chatbot interactions (using LangChain and Ollama)
├── requirements.txt    # List of dependencies
├── assets/             # Folder for static files
└── config.py           # Configuration for Ollama, LangChain, and API key
```

Main Components

1. Ollama Llama Model

Ollama's Llama is the core model for conversational processing. Llama is a powerful language model optimized for diverse conversational contexts, and it supports multitasking and long-form interactions.

2. LangChain Integration

LangChain acts as the integration layer that manages data flow between the Llama model and external sources (APIs, databases, etc.), and ensures smooth context switching between various topics.

Key LangChain Concepts in This Project:

- **Chains**: A sequence of actions where each step may involve calling an API, processing the result, and passing it on to the next step.
- **Agents**: LangChain's agents can be used to dynamically choose the right tool (like an API, database, or other services) based on the conversation's context.
- **Memory**: LangChain can help store conversation history or session data for enhanced user interaction, allowing the model to remember past interactions.

Code Implementation:

```
from langchain_openai import ChatOpenAI

from langchain_core.prompts import ChatPromptTemplate

from langchain_core.output_parsers import StrOutputParser

from langchain_community.llms import Ollama

import streamlit as st

import os

from dotenv import load_dotenv

load_dotenv()

os.environ["LANGCHAIN_TRACING_V2"] = "true"

os.environ["LANGCHAIN_API_KEY"] = os.getenv("LANGCHAIN_API_KEY")

prompt = ChatPromptTemplate.from_messages(

   [

      ("system", "You are a helpful assistant. Please response to the user queries"),

      ("user", "Question:{question}")

   ]

)

#streamlit framework
```

```
st.title('CHAT BOT')

input_text = st.text_input("HOW CAN I HELP YOU")

llm = Ollama(model="gemma2:2b")

output_parser = StrOutputParser()

chain = prompt | llm | output_parser

if input_text:

    st.write(chain.invoke({"question": input_text}))
```

output:

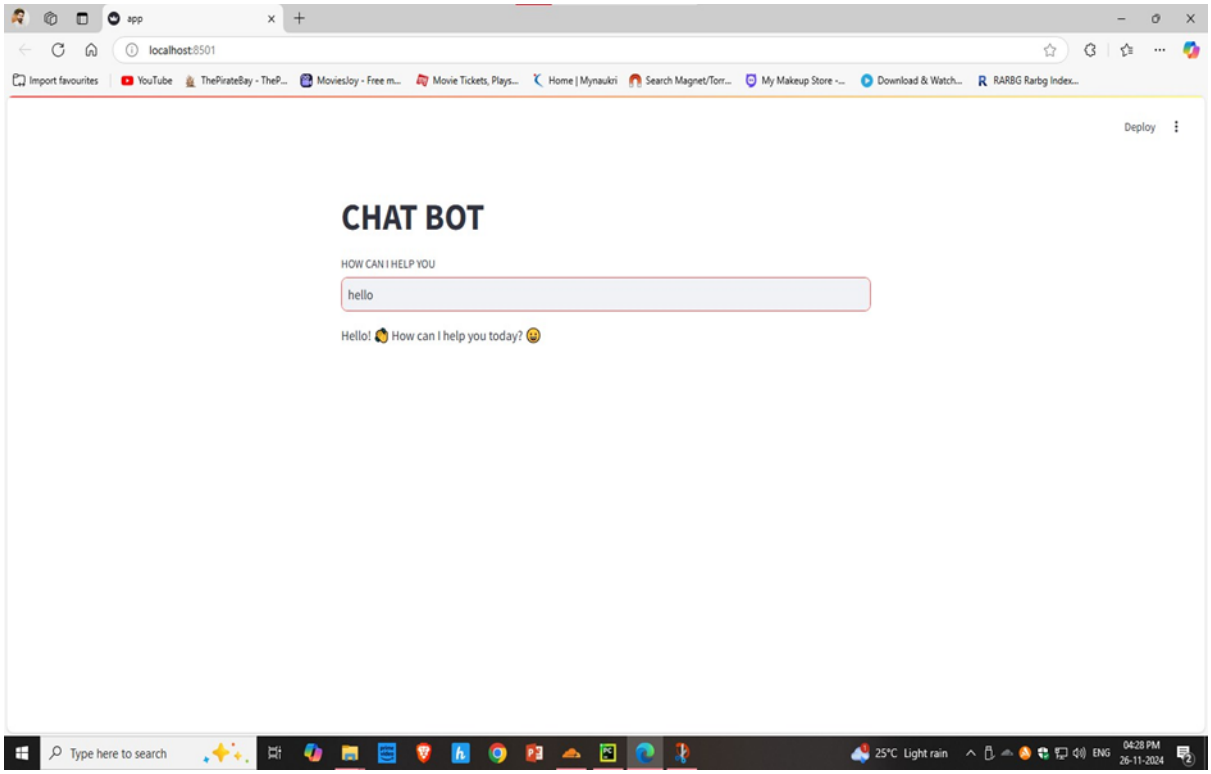

figure:01

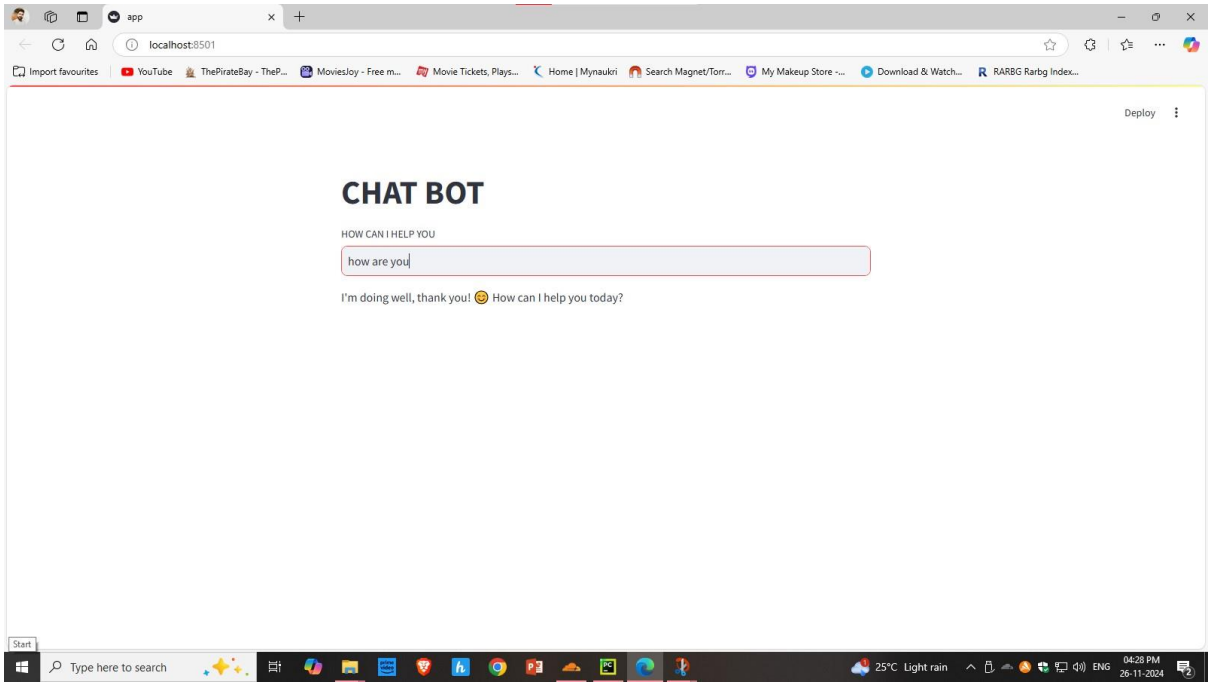

Figure 02

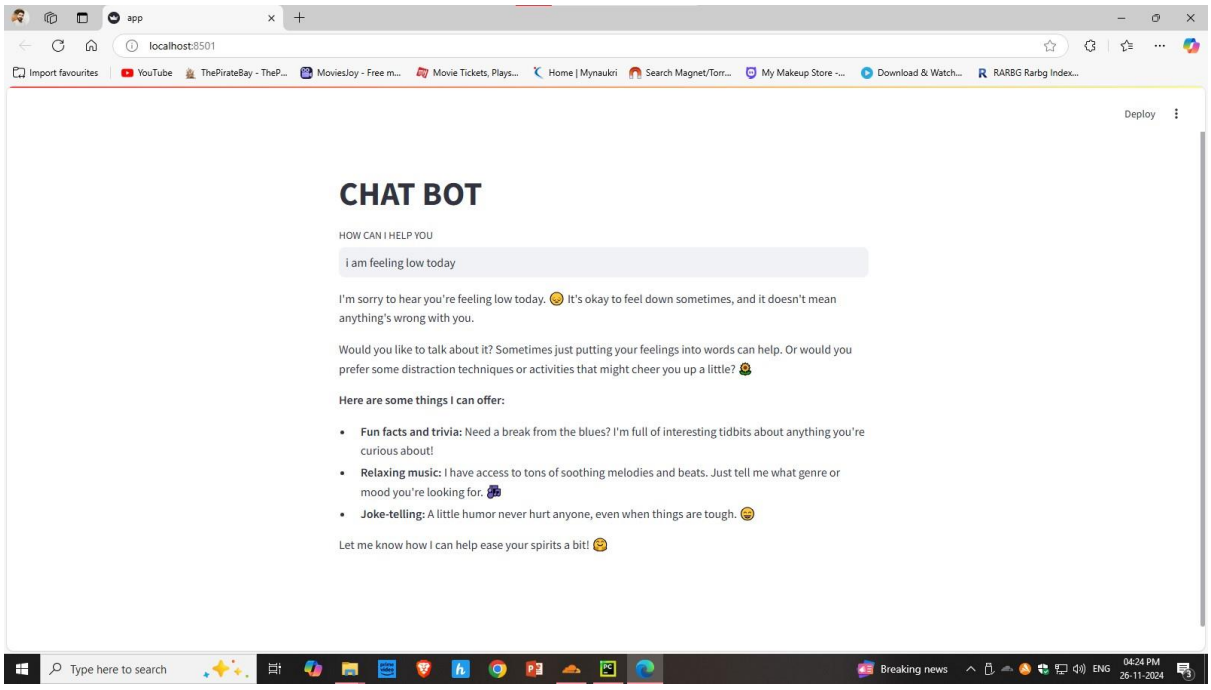

Figure 03

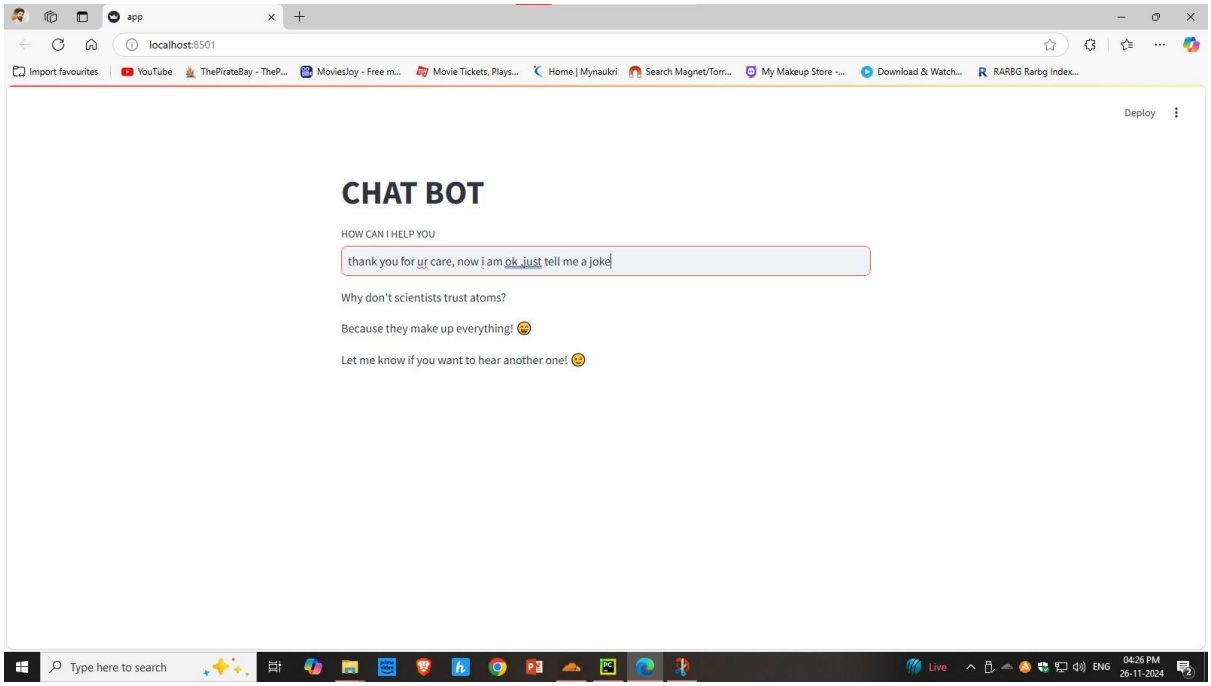

Figure 04

Running the Application

Once the dependencies are installed and the configuration is set up, you can run the chatbot by executing the following Streamlit command:

```
streamlit run app.py
```

This will start a local web server, and you can access the chatbot via your browser.

Conclusion

This project demonstrates how to integrate powerful AI models like Ollama's Llama, LangChain, and Streamlit to create a versatile conversational chatbot. With this setup, you can easily extend the bot's capabilities to handle multiple topics, integrate real-time data from APIs, and create an interactive UI for users.