Langchain Project for Customer Support App in Python
Project Overview

Overview
The Langchain project is a customer support application leveraging Large Language Models (LLMs) to enhance the customer service experience. Customer support is pivotal in any business, serving as a direct channel for users to seek assistance, resolve issues, and gain valuable information. In this project, we harness the power of LLMs to streamline and optimize the customer support process.

Large Language Models (LLMs) have emerged as a game-changing technology in natural language processing. They can understand, process, and generate human-like text, enabling applications to engage in complex conversations with users. In customer support, LLMs play a crucial role by providing intelligent, context-aware responses, thereby significantly improving the efficiency and effectiveness of customer interactions.

The Langchain project is a multi-agent customer support application powered by LLM. It utilizes a script to greet users, employs a PDF knowledge base to answer queries, and connects to a user database for additional information. If an issue cannot be resolved through chat, it provides a "Submit Ticket" button. The app also includes two additional LLM agents for audio transcription and summarization. It simulates a user's phone call during the chat, where the MP3 of the call is uploaded into the system. The app uses LLM to translate the call into text and summarize the call, adding the information to the ticket.

Prerequisite Project: Kindly ensure the completion of the LLM Project for building and fine-tuning a large language model before proceeding with this project.

Note: Please note that the use of the OpenAI API may require the utilization of allocated free credits or additional purchases (depending on the account status) for implementing the project. Kindly take note of the free credits limit provided for your usage.

Aim
The Langchain project aims to leverage Large Language Models (LLMs) to enhance customer support interactions by providing intelligent, context-aware responses. It seeks to seamlessly integrate LLM technology with databases, PDF knowledge bases, and audio processing agents, creating a comprehensive customer support application.

Data Description

The project utilizes diverse datasets, including user interaction logs, PDF knowledge bases, and user information stored in databases for demonstration purposes.

Tech Stack

➔ Language: Python 3.10.4

➔ Libraries: sentence transformers, openai, langchain, chromadb, streamlit

Approach

- Greeting and Authenticating the Customer:
  - The system starts with a greeting message, welcoming the user to the customer support service.
  - Prompt the user to provide their full email address or phone number for authentication.
  - Implement an LLM-based edge to validate and authenticate the user input.
  - Upon successful authentication, retrieve user details and subscription information.
- LLMs as Graphs:
  - Define utility functions for handling system output, user input, and parsing information.
  - Create classes for edges and nodes to structure the customer support flow as a graph.
  - Implement an edge that checks if the input meets specific conditions (e.g., valid email, phone call request).
  - Parse the validated input to extract relevant information (e.g., email, phone number).
- Node Orchestration:
  - Design a node to manage the flow of conversation and orchestrate the edges.
  - Gather user input and identify which edge(s) should be followed based on conditions.
  - Implement logic to run to continue with the appropriate edge based on user input.
- Edge Implementation:
  - Define various edges with specific conditions and parsing queries.
  - Use LLMs to perform checks and parse user input to extract relevant information.

- Retrieval Augmented on Health Center Data:
    - Create knowledge bases (e.g., premium subscription knowledge base, free subscription knowledge base) for context-based retrieval.
    - Implement a multi-retrieval chain to select the appropriate knowledge base based on the user's subscription type.
    - Use the chain to retrieve information relevant to user queries.
- Call Handling and Summary Generation:
    - Set up a call edge to simulate phone call interactions with the user.
    - Retrieve user information and subscription details from history to initiate the call.
    - Use LLMs to handle the call and extract a summary of the conversation for ticket generation.
- Final App Integration:
    - Integrate all components into a comprehensive customer support application.
    - Establish a user interface for interaction, allowing users to engage with the application.

Modular code overview:

Once you unzip the modular_code.zip file, you can find the following folders.

```
├─ agents
│  ├─ support.py
├─ assets
│  ├─ audio
│  │  └─ customer_support.wav
│  ├─ free
│  │  ├─ compliance.txt
│  │  ├─ locations.txt
│  │  ├─ payments.txt
│  │  └─ pos.txt
│  └─ paid
│  |  ├─ compliance.txt
│  |  ├─ locations.txt
│  |  ├─ payments.txt
│  |  └─ pos.txt
|_ |_ tools
|      └─ audio_transcribe.py
|      |_ rag_responder.py
|      |_ user_info_db.py
|  |_ ui
|    |_ graph_renderer.py
|  graph
|  ├─ chain_based_edge.py
|  ├─ chain_based_node.py
|  ├─ edge.py
|  ├─ node.py
|  ├─ static_text_node.py
|  └─ text_based_edge.py
├─ customer_support.ipynb
├─ customer_support.py
|_ llm_app.py
├─ data
|  ├─ chat.py
|  ├─ graph.py
|  ├─ validation.py
├─ readme.md
└─ requirements.txt
```

The code folder includes the following folders:

agents:

- Contains the support.py module, which includes code related to customer support agents.

assets:

- Contains subdirectories:
  - audio: Holds the customer_support.wav audio file.
  - free: Contains text files (compliance.txt, locations.txt, payments.txt, pos.txt) related to free.
  - paid: Contains text files (compliance.txt, locations.txt, payments.txt, pos.txt) related to paid.

data:

- Contains Python modules (chat.py, graph.py, validation.py) related to data handling, chat, graph-related functionality, and data validation.

tools:

- Contains the following Python scripts (audio_transcribe.py, rag_responder.py, user_info_db.py) related to transcribing audio files, for responding to users, and managing user information in a database.

ui:

- Contains the graph_renderer.py module, which includes code for rendering graphs in the user interface.

graph:

- Contains the following Python modules (chain_based_edge.py, chain_based_node.py, edge.py, node.py, static_text_node.py, text_based_edge.py) related to graph-based structures.

Root Level:

- customer_support.ipynb: Jupyter Notebook file
- customer_support.py: Python script, serving as the main implementation for customer support functionalities.
- readme.md: Markdown file containing documentation or information about the project.
- requirements.txt: Text file specifying project dependencies and their respective versions.
- llm_app.py: Streamlit application

Project Takeaways

1. Understanding Large Language Models (LLMs) and their significance in automating customer support processes
2. What is the LangChain Framework?
3. Exploring LangChains and how they structure multi-step customer support flows
4. What are custom output parsers?
5. How is ReACT prompt engineering performed?
6. What are Directed Acyclic Graphs?
7. How are LLMs as graphs used to create workflows?
8. Implementing Retrieval augmented generation for context-based information retrieval
9. Implementing multi-retrieval chains for interacting with external databases to retrieve relevant information
10. Utilizing the Whisper open AI model for speech-to-text functionality in voice-based interactions