Explanation of Implementation: AI Agent-based Deep Research System

Introduction

The goal of this project was to develop a Deep Research AI Agentic System that crawls websites using Tavily for online information gathering. The system was designed with a dual-agent architecture, where one agent is responsible for gathering research data from the web, and the second agent drafts answers based on the gathered information. The system utilizes LangGraph and LangChain frameworks to effectively organize and process the gathered data.

System Architecture

1. Agents

- **Research Agent**: This agent is responsible for querying online sources to gather relevant research data. It uses the Tavily API to search for the information needed based on a user query. It collects the research results from different sources and summarizes them for further use.
- **Drafting Agent**: Once the research data has been collected, the drafting agent takes over. It processes the collected information to draft a final answer or summary in response to the user's query. If no research content is found, it returns an appropriate message.

2. Technologies Used

- **Tavily API**: The Tavily API is used to perform the search queries and fetch research results from a wide range of online sources. It provides search capabilities that are crucial for the data collection phase of the project.
- **LangGraph**: LangGraph is employed to model the different stages of the system using a state graph. It helps organize the workflow of both agents and allows easy tracking of the current state of the system (e.g., research content, final answers).
- **LangChain**: LangChain assists in managing the flow of logic between the agents and helps integrate different tools and frameworks within the system. It ensures that the output from the research agent feeds seamlessly into the drafting agent.

Detailed Implementation

Step 1: Setting Up the State Schema

The system was built on a state machine where the state of the system at any point in time is represented by a ResearchState. This state schema includes:

- query: The research query provided by the user.
- research_content: The content fetched by the research agent.
- final_answer: The answer drafted by the drafting agent.

Step 2: Initializing LangGraph

LangGraph was used to define a flow of tasks that the agents will follow. The system was structured as a series of nodes:

- **Search Node**: Responsible for gathering research from online sources based on the user's query.
- **Draft Node**: Once the research is collected, this node drafts the final answer for the user query.

Step 3: Implementing the Research Agent

The research agent utilizes the Tavily API to perform the search. Upon receiving a query, it sends the query to the Tavily service and retrieves the most relevant results. If results are found, they are formatted into a summary, which will be used by the drafting agent.

Step 4: Implementing the Drafting Agent

After the research data is collected, the drafting agent takes over. The drafted answer is based on the research content fetched by the research agent. If no research data is available, the drafting agent informs the user.

Step 5: Running the System

The system is designed to accept user input for a query. Upon receiving the input, the system invokes the search agent to gather research and then invokes the drafting agent to process the research data into a final answer.

Flow of Execution

1. **User Input**: The user provides a query for research.
2. **Research Agent**: The system queries Tavily for research results based on the user input.
3. **Data Processing**: The research agent compiles the results and stores them in the research_content.
4. **Drafting Agent**: The drafting agent processes the research content and prepares the final answer.

5. **Final Output**: The final drafted answer is returned to the user.

Challenges & Solutions

1. **Data Collection**: Initially, the biggest challenge was ensuring that the research content gathered was relevant and comprehensive. To mitigate this, we limited the number of results fetched and refined the queries.
2. **Integration**: The integration of LangChain and LangGraph was another challenge as both tools need to be synchronized correctly for seamless execution. Proper state management helped resolve this issue.
3. **Error Handling**: Several exceptions could arise during the execution (e.g., API failure, no results returned). These were handled gracefully by providing default error messages and retry mechanisms.

Conclusion

This Deep Research AI Agentic System leverages advanced tools and frameworks to offer an efficient solution for gathering and drafting information based on user queries. The combination of Tavily for research, LangGraph for managing states, and LangChain for logical flow provides a robust and scalable solution. The system has the potential to be expanded further with more agents, additional data sources, or enhanced user interaction features.