Project: Multi-Agent AI Customer Support System

Problem Statement:

We addressed the challenge of providing efficient and comprehensive customer support, particularly when dealing with complex data like order information and product details. Traditional methods often involve manual lookups and fragmented information, leading to slow response times and inconsistent answers. We built an intelligent customer support system that seamlessly accesses and synthesizes data from multiple sources, providing accurate and timely responses to customer inquiries.

Project Abstract:

This project successfully developed a multi-agent AI application using Langflow. The system handles customer support tasks by integrating Retrieval Augmented Generation (RAG) with database queries. It comprises several specialized agents, including an order lookup agent, a product information retriever, and an FAQ responder, all managed by a central orchestrator. We leveraged Astra DB, a vector database, to efficiently store and retrieve relevant information. A user-friendly interface was created using Streamlit to enable seamless interaction with the AI system. The complete project codebase has been pushed to Git.

Project Summary:

The following key steps were undertaken:

1. **Foundation – Basic AI Agent Setup (Langflow):**
   - A foundational Langflow flow was created.
   - "Chat Input," an LLM node (e.g., OpenAI's GPT), and "Chat Output" nodes were implemented.
   - The LLM node was configured with API keys.
   - Basic chat functionality was tested and verified.
2. **Data Integration – Vector Database (Astra DB):**
   - Astra DB was integrated for data retrieval.
   - An "Astra DB" node was added to the flow.
   - A connection to an Astra DB instance was established.
   - A vector-enabled collection was created.
3. **RAG Implementation – FAQ Handling:**
   - The agent was enabled to retrieve information from a PDF FAQ document.
   - "PDF Loader," "Text Splitter," and "Embeddings" nodes were implemented.
   - The PDF was processed, embeddings were created, and stored in Astra DB.
   - The LLM was configured to use retrieved context.
4. **Specialized Tool Development – Order Lookup:**

- An agent was created specifically for order lookups.
- Database query nodes were added.
- SQL queries were written (or database connector nodes were used) to retrieve order details.
- The output was formatted for user-friendly display.

5. **Agent Orchestration – Manager Agent:**
   - A manager agent was built to direct user queries.
   - A "Router" or "Agent Executor" node was added.
   - Rules or prompts were defined to guide the manager's decisions.
   - The manager was connected to the FAQ and order lookup tools.
6. **API Integration – External Access:**
   - The system was enabled to be accessed via an API.
   - The Langflow API endpoint was obtained.
   - API calls were tested with tools like curl or Python's requests.
7. **User Interface – Streamlit Front End:**
   - A user-friendly web interface was created.
   - A Streamlit application was developed.
   - Input fields and display areas were implemented.
   - API calls to the Langflow flow were integrated.

Here's a detailed explanation of the data flow within the multi-agent AI system, from start to finish:

1. **User Query Input:**
   - The process begins when a user enters a question or request into the text input field of the Streamlit application's user interface. This query is typically a customer support request, such as "What is the status of my order #12345?" or "What are the return shipping costs?"
2. **Streamlit to Langflow API:**
   - Streamlit, acting as the front-end, captures the user's query.
   - Streamlit then sends this query to the Langflow Runtime via the Langflow API.
   - This communication involves an API request (likely an HTTP POST request), where the user's query is included in the request payload.
3. **Langflow Runtime Reception:**
   - The Langflow Runtime, which is the execution environment for the defined multi-agent flow, receives the API request from Streamlit.
   - The Langflow Runtime is responsible for orchestrating the execution of the agent flow.
4. **Manager Agent: Query Analysis and Routing:**
   - The Langflow Runtime passes the user's query to the Manager Agent.
   - The Manager Agent's role is crucial: it analyzes the user's query to determine the most appropriate agent(s) to handle the request.

    - This analysis might involve:
        - Keyword extraction: Identifying key terms like "order status" or "return shipping."
        - Intent recognition: Determining the user's intention (e.g., to check order status, to ask a question).
        - Context analysis: Considering any previous interactions or session data.
    - Based on this analysis, the Manager Agent decides which agent or agents should process the query (in this case, the FAQ Agent or the Order Lookup Agent).
5. **FAQ Agent Path (for FAQ-related queries):**
    - If the Manager Agent determines the query is related to frequently asked questions, it routes the query to the FAQ Agent.
    - **5.1 FAQ Agent to Astra DB:** The FAQ Agent formulates a query for the Astra DB vector database. This query is designed to retrieve FAQ entries that are semantically similar to the user's query. The query uses the vector embeddings.
    - **5.2 Astra DB Retrieval:** Astra DB performs a similarity search using the query vector. It identifies the FAQ entries whose embeddings are most similar to the query's embedding. Astra DB returns these relevant FAQ entries to the FAQ Agent.
    - **5.3 LLM Contextualization and Response Generation:** The FAQ Agent receives the relevant FAQ entries from Astra DB. The FAQ Agent then combines these retrieved FAQ entries with the original user query. This combined information is provided as context to the LLM. The LLM uses this context to generate a natural language response that directly answers the user's question, drawing upon the information from the FAQ entries.
    - For example, if the user asks, "How long does shipping take?", the FAQ Agent might retrieve FAQ entries like "Shipping Time: 3-5 business days" and "Shipping Options: Express and Standard." The LLM would then use this information to generate a response like, "Shipping typically takes 3-5 business days. We offer both Express and Standard shipping options."
6. **Order Lookup Agent Path (for order-related queries):**
    - If the Manager Agent determines the query is related to order information, it routes the query to the Order Lookup Agent.
    - **6.1 Order Lookup Agent to Order Database:** The Order Lookup Agent constructs an SQL query to retrieve the relevant order information from the Order Database. The specific SQL query will depend on the user's request (e.g., retrieving order status, order details, or shipping address).
    - **6.2 Order Database Query and Retrieval:** The Order Database executes the SQL query and retrieves the requested order information. The database returns this data to the Order Lookup Agent.
    - **6.3 Order Lookup Agent Response Formatting:** The Order Lookup Agent receives the raw order data from the database. It then formats this data into a user-friendly response. This might involve:
        - Extracting the relevant fields (e.g., order status, shipping date, items

        ordered).
        - Formatting dates and numbers.
        - Constructing a clear and concise message.
    - For example, if the user asks, "What is the status of my order #12345?", the Order Lookup Agent might retrieve data like "order_id: 12345, status: 'shipped', shipping_date: '2024-07-28'". It would then format this into a response like, "Order #12345 has shipped on July 28, 2024."

7. **Response Aggregation and Delivery:**
    - The selected agent (either the FAQ Agent or the Order Lookup Agent) sends its generated response back to the Langflow Runtime.
8. **Langflow Runtime to Streamlit:**
    - The Langflow Runtime receives the response from the agent.
    - The Langflow Runtime then forwards this response to the Streamlit application via the Langflow API.
9. **Streamlit Display:**
    - Streamlit receives the response from the Langflow Runtime.
    - Streamlit displays the response to the user in the user interface. This completes the cycle.

tech stack with detailed explanations:

- **Langflow:** A visual tool for building AI workflows by connecting components, simplifying AI agent creation.
- **Python:** The programming language for Langflow, API interaction, logic, and the Streamlit app.
- **LLMs:** AI models enabling agents to understand and generate human-like text for natural language processing.
- **Astra DB:** A vector database for efficient storage and retrieval of data, crucial for implementing RAG.
- **RAG:** A technique to enhance LLM responses with relevant information from Astra DB.
- **Streamlit:** A Python framework used to create the user interface for interacting with the AI system.