Multi-Source RAG Project Assignment

Overview

This assignment involves developing a recommendation system with multi-source retrieval capabilities using a Retrieval Augmented Generation (RAG) architecture. You must deploy your project as a live service with an exposed API for chat interactions. The system should intelligently integrate multiple data sources as follows:

- For product recommendation queries (e.g., "a moisturiser under 1200 for oily skin"), the system should use the provided XLSX file containing product data.
- For blog-related queries, the system should retrieve information from a supplied blogs dataset.
- For general skincare inquiries or any other queries (e.g., "How can I treat acne"), the system should be capable of utilizing an external data source, such as a web search or another relevant API or scraped data.

Candidates have the flexibility to choose any RAG architecture—whether a single-agent or multi-agent approach— to handle the query routing and data integration. You may use frameworks such as LangChain or LangGraph, or develop your solution independently. Innovation in routing, managing multiple data sources, and integrating generative responses is highly encouraged.

Assignment Tasks

1. Data Preparation:

- Load and preprocess the provided XLSX file containing product information. Remove any extraneous fields that are not necessary.
- Load and preprocess the provided blog source dataset.

2. System Implementation:

- Design and implement a RAG architecture that integrates multiple data sources: product data, blog data, and external search data.
- Develop logic to detect the intent of user queries. For instance, if the query is product-specific, use the XLSX data; if it relates to blogs, use the blog dataset; otherwise, route the query to an external data source.
- Implement support for either a single-agent or multi-agent system to manage and integrate responses from the various data sources.
- Ensure that retrieval and generative components are effectively consolidated to produce high-quality, context-aware responses.

3. Deployment:

- Deploy your solution as a backend service and expose an API endpoint for chat interaction.
- You may choose any framework or service for backend development and deployment (e.g., Flask, FastAPI, Node.js, etc.).
- Provide comprehensive documentation or instructions on how to interact with your API.

4. Demonstration:

- Record a screen demonstration explaining your technology choices, system design, query routing mechanism, and any challenges encountered during the implementation and deployment process.
- Clearly demonstrate how your system handles product queries, blog queries, and general inquiries differently.

Bonus Points:

Extra credit if you develop a frontend chat interface that integrates with your API, offering features such as conversation history and a user-friendly chat interface.

Evaluation Criteria

- Clarity and organization of your codebase and project structure.
- Justification for the chosen tech stack, RAG architecture, and query routing logic.
- Effective integration and utilization of the provided product data, blog dataset, and external data sources.
- Innovation in the use of single-agent or multi-agent approaches for data integration.
- Quality and clarity of your screen-recorded demonstration.
- Bonus: Development of a frontend chat interface demonstrating a complete end-to-end solution.

Submission Guidelines

- Submit the link to your deployed API or detailed instructions for local setup, along with the complete source code repository.
- Ensure that your documentation, deployment instructions, and the screen demonstration are thorough and clear.

Good luck, and we look forward to seeing your innovative solutions for multi-source RAG implementations!