Development of a Multimodal Retrieval-Augmented Generation (RAG) Pipeline for PDF Question Answering

Objective

Develop a robust Retrieval-Augmented Generation (RAG) pipeline using LangChain and Ollama, leveraging open-source large language models (LLMs) of your choice. The system should facilitate question answering from PDF documents, incorporating both textual and visual content. The user interface should be developed using Streamlit, Gradio, Plotly Dash, or a backend framework of your choice (Django, Flask, or FastAPI), allowing users to upload PDFs and interact with the system seamlessly.

Functional Requirements

1. **PDF Upload and Parsing**
    - Implement a user interface component to upload PDF files.
    - Extract and parse text from the uploaded PDFs, handling various structures such as tables and images.
    - Utilize libraries like PyPDF2, pdfminer.six, pymudf (fitz) or unstructured for text extraction. (Or any other libraries you may know)
2. **Text Chunking and Embedding**
    - Develop a professional-grade chunking mechanism to segment the extracted text into coherent units.
    - Handle different types of content (e.g., paragraphs, tables, images) appropriately during chunking.
    - Use embedding models from Hugging Face or Ollama to convert text chunks into vector representations.
3. **Vector Database Integration**
    - Store the embeddings in a vector database (e.g., Chroma, FAISS, Qdrant) for efficient retrieval.
    - Implement a retrieval mechanism to fetch relevant chunks based on user queries.

4. **Question Answering Mechanism**
    - Utilize LangChain's RetrievalQA chain to generate answers by combining retrieved context with the user's question.
    - Ensure the system provides accurate and contextually relevant answers.
5. **Visual Question Answering (VQA)**
    - Implement functionality to process images (e.g., screenshots) from the PDF.
    - Use appropriate Vision-Language Models (VLMs) to extract information from images and integrate it into the answer generation process.
6. **User Interface Development**
    - Develop a user-friendly interface using Streamlit, Gradio, Plotly Dash, or a backend framework of your choice (Django, Flask, or FastAPI).
    - The interface should allow users to:
        - Upload PDF files.
        - Input text queries.
        - Upload images for visual question answering.
        - Display answers alongside relevant excerpts from the PDF.
    - Ensure the UI is responsive and intuitive.
7. **Performance and Scalability**
    - Optimize the pipeline for performance, ensuring fast response times even with large PDF documents.
    - Consider scalability to handle multiple concurrent users.
8. **Documentation and Testing**
    - Provide comprehensive documentation covering:
        - System architecture.
        - Set-up and installation instructions.
        - API usage (if applicable).

    - Implement unit and integration tests to ensure the reliability of the system.

Evaluation Criteria

- **Functionality**: The system should meet all specified functional requirements.
- **Code Quality**: Code should be clean, well-organized, and adhere to best practices.
- **User Experience**: The user interface should be intuitive and responsive.
- **Performance**: The system should handle large PDFs efficiently and provide quick responses.
- **Documentation**: Clear and comprehensive documentation should be provided.
- **Testing**: Adequate test coverage should be implemented.

Submission Guidelines

- Provide access to the complete source code repository.
- Include a README file with setup instructions and usage examples.
- Submit a demo video showcasing the system's capabilities.
- Provide any additional resources or documentation that would help in evaluating the system.