RAG-Based Information Retrieval System for PDF Documents

1. Overview

This project implements a Retrieval-Augmented Generation (RAG) based conversational question-answering system tailored for PDF documents. The system is designed to extract information from complex PDFs containing unstructured text, multi-page tables, and figures, and respond accurately to user queries about any part of the document.

The solution consists of three core components:

- Document ingestion and processing pipeline
- Backend API service for RAG-based querying
- Frontend conversational UI for interactive Q&A

2. Frontend conversational UI for interactive Q&A

- Enable ingestion of PDFs with granular extraction of textual paragraphs, tables (split into structured rows), and figures.
- Store extracted data in a vector database to enable efficient semantic search.
- Implement a Conversational Retrieval Chain to support:
  - Contextual follow-up questions
  - Chat history management
  - Accurate responses referencing the document source

3. Detailed Methodology

3.1 Document Ingestion and Preprocessing

- The system accepts PDF uploads and parses them using tools like PyMuPDF or pdfplumber.
- For textual content, the document is split into manageable chunks (e.g., paragraphs or sections).
- For tables spanning multiple pages, the ingestion pipeline extracts them as structured data:
- Multi-page tables are concatenated and parsed into individual rows.
- Each row is treated as an independent document chunk to allow fine-grained retrieval.

- For figures, captions and associated metadata are extracted and stored as separate text chunks.
- Each chunk is enriched with metadata including doc_id, page number, and type (text/table/figure).
- Text chunks are embedded using pre-trained embedding models from huggingface i.e. sentence-transformers/all-MiniLM-L6-v2
- All embeddings are stored in a vector database (e.g., ChromaDB) for semantic similarity search.

3.2 Retrieval-Augmented Generation (RAG) Chain

- When a user submits a query, the system:
- Converts the query into an embedding.
- Searches the vector store for the top_k most relevant chunks based on semantic similarity.
- Passes these chunks along with the query to a large language model (LLM) prompt.
- A ConversationalRetrievalChain is implemented using LangChain:
- Maintains chat history to add context from previous interactions.
- Handles follow-up questions by reformulating queries using prior context.
- The LLM then generates an answer using the retrieved chunks as reference context.
- The system returns both the answer and the source chunks used, including page numbers and snippet highlights.

4. Design Decisions and Reasoning

- Fine-grained ingestion of tables as rows : Splitting tables into rows rather than ingesting them as a single blob allows precise retrieval of specific table data relevant to the query, improving answer accuracy.
- Conversational chain with chat history : Enables natural multi-turn Q&A, where follow-up questions leverage previous context, enhancing user experience.
- Vector store filtering by document ID : Supports multi-document environments and user-specific isolation to avoid data leakage.
- Use of LangChain : Provides modular, extensible primitives for chaining retrieval and generation, accelerating development while maintaining production readiness.
- Separation of backend and frontend : Allows independent scaling and easier testing of API and UI components.