Architectural Design Document for RAG Pipeline

1. Document Ingestion, Chunking, and Representation

The system begins by extracting data from a PDF by using **partition_pdf** function of the "**unstructured"** library. This tool allows for efficient parsing of both text and tables. The partition_pdf method extracts the document into structured chunks of limited character sizes, handling various types of content like text, tables, and images. For text data, chunking is performed **based on titles** and **logical breaks** in content, while table data is handled separately, preserving the structure of tables for better semantic understanding. Text data is processed as chunks of **plain text,** while table data is separated in the **HTML text** format. Most of the LLMs are capable of understanding HTML text and hence tables can be fed easily as context in this form. This dual approach ensures that both types of data are represented effectively for later embedding.

2. Embedding and Retrieval Strategy:

The text extracted from the document is divided into manageable chunks of 10000 characters, with the strategy "**by_title"** ensuring that chunks represent logical sections of the document. The **"sentence-transformers/all-MiniLM-L6-v2"** embedding model is chosen for its balance of performance and accuracy in generating sentence-level embeddings. The embeddings are used to store the text and table summaries in a Chroma vector store, which allows efficient retrieval during query processing. The embeddings are stored in the vectorstore directory for persistent use.

To enhance the quality of document retrieval, we implement a **multi-vector store strategy**. This approach efficiently handles both summarized and raw data to ensure high-quality semantic search.

- **Summarization**

  The process begins with summarizing the raw documents using an LLM. This step condenses large amounts of information into more manageable summaries while retaining essential details relevant for querying. The summarization process occurs in two parts:

  1. **Table Summarization:** For documents containing tables, the table content is summarized to create a more compact, high-level overview. This summary is crafted to preserve key information for semantic search without overloading the vector store with raw data.
  2. **Text Summarization:** For textual content, sections of the document are summarized to ensure that only the most pertinent information is retained for retrieval. This process ensures that irrelevant or overly detailed content does not overwhelm the retrieval system, improving search efficiency and accuracy.

- **Vectorstore Storage**

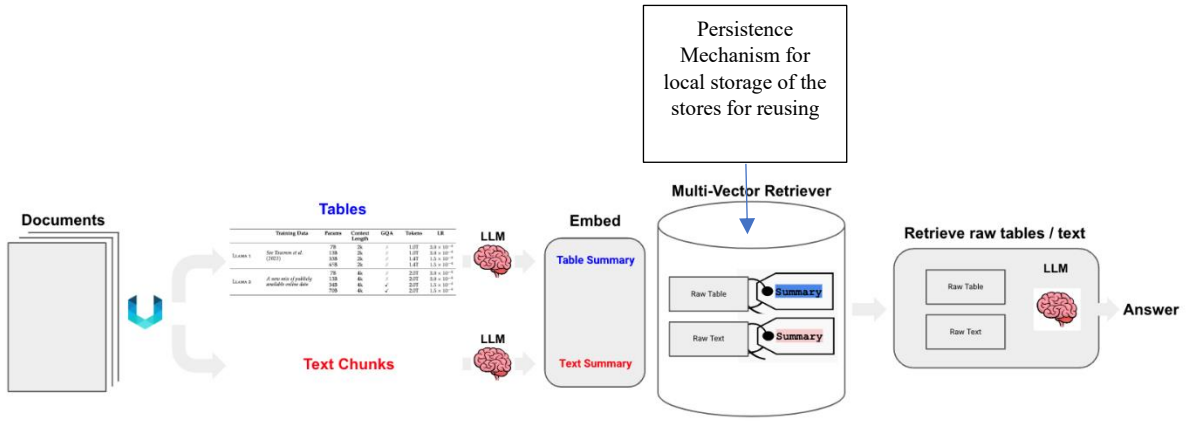
  After summarizing, the next step is embedding both the summarized text and tables into a **Chroma vectorstore**. The summaries are indexed using embeddings from models like **OpenAIEmbeddings** to enable efficient retrieval based on semantic similarity. By storing these embeddings in the vectorstore, we ensure that the system can quickly retrieve content most relevant to the user's query.

- **Raw Data in Docstore**

  In addition to storing the summarized content in the vectorstore, we also maintain the **raw documents** in a **docstore**. This docstore retains the full structure of tables and text, which can later be referenced by the LLM to provide richer context when generating responses. This allows for the retrieval of both condensed summaries and detailed raw data, depending on the query requirements.

3. RAG Pipeline Flow

The flow of the RAG pipeline is visualized by the following diagram:



4. Unanswerable Queries & Hallucinations

To address unanswerable queries and prevent hallucinations, the system employs a simple strategy that ensures the model only answers questions when there is relevant context. If no context is found during the retrieval phase, the model is instructed not to generate an answer. Additionally, the prompt explicitly discourages answers when the relevant information is absent. This approach reduces the likelihood of hallucinations and ensures that responses are grounded in the available data.

5. Scalability, Performance & Maintenance

Scalability and performance are critical aspects of this design. To handle increasing document sizes and user queries:

- **Efficient Retrieval:** The multi vector retriver ensures that embedding-based retrieval is fast and scalable. The persistence mechanism is included to store locally both vectorstore and docstore for reuse, supporting large-scale storage and retrieval operations without recomputing the embeddings.
- **Batch Processing:** By processing documents in batches (e.g., chunking, summarization), the pipeline optimizes memory usage and processing time.
- **Modular Design:** Each component (e.g., embedding, storage, retrieval) is designed as a separate module, making it easy to replace or upgrade components (e.g., embedding model, storage backend) without disrupting the entire pipeline.

6. Continuous Evaluation & Improvement

- **Automated Evaluation:** Automated evaluation scripts can be implemented to test the performance of the model on a test set of queries to identify any potential performance degradation after model updates.
- **Update Mechanism:** New documents can be easily ingested and processed, with updates reflected in the vector store. This modular architecture supports easy updates and model improvements without disrupting service.

Framework Selection Justification

- **HuggingFace Transformers:** Used for both the LLM and embedding models due to its state-of-the-art performance in NLP tasks.
- **Chroma Vector Store:** Provides efficient, scalable storage for embeddings and supports fast retrieval, making it ideal for RAG systems.
- **LangChain:** Facilitates the orchestration of the RAG pipeline by enabling smooth integration between document processing, retrieval, and LLM inference.
- **Streamlit:** Provides an easy-to-use interface for users to interact with the system, enhancing usability and testing.

These frameworks were selected to ensure a balance of performance, scalability, and maintainability while providing a seamless user experience.