**MRAG- upload a pdf and ask any question**

This project aims to build a Multimodal RAG where the user can upload an pdf (any book or user manual) and ask a question around it.

**Research And Development**  
  
This application in build in mainly three ways.  
  
**Approach 1:** **Direct Multimodal Embedding**

- The entire PDF document (text + images) is embedded directly into a multimodal embedding space, creating unified vectors.  
- Embeddings are stored in a vector database for similarity-based retrieval.  
  
Challenges:  
- Scalability issues with very large documents.  
- Limited control over granularity (retrieving fine-grained elements like tables, figures, etc.).

**Approach 2: Separate Processing of Modalities**

- PDFs are preprocessed to individually extract text, images, and tables.  
- Text and images are separately embedded using respective embedding models.  
- Resulting embeddings are stored in distinct vector databases.  
  
Challenges:  
- Complex retrieval orchestration.  
- Requires careful synchronization between text and image retrieval results.

**Approach 3: Summarization and Descriptive Embedding (Implemented)**

* The PDF is **preprocessed** to extract **text**, **images**, and **tables**.
* A **multimodal LLM** (Sonnet Claude 3.7 via AWS Bedrock) is used to:
  + Generate **summaries** for extracted text.
  + Generate **detailed descriptions** for extracted images.
* These summaries and descriptions are **embedded** into a **single vector database**.
* Each embedding is **mapped to a unique ID** linking back to the original text or image stored in a **separate datastore**.
* During inference:
  + The user’s query is **vectorized**.
  + A **similarity search** retrieves the closest matching vectors.
  + The corresponding actual text and/or image are fetched using the unique ID and provided as **context** to the LLM to generate the final answer.

**Approach 3** was selected for implementation due to its **scalability**, **modularity**, and ability to provide **highly relevant contextual information** during retrieval and generation.

**System Architecture**

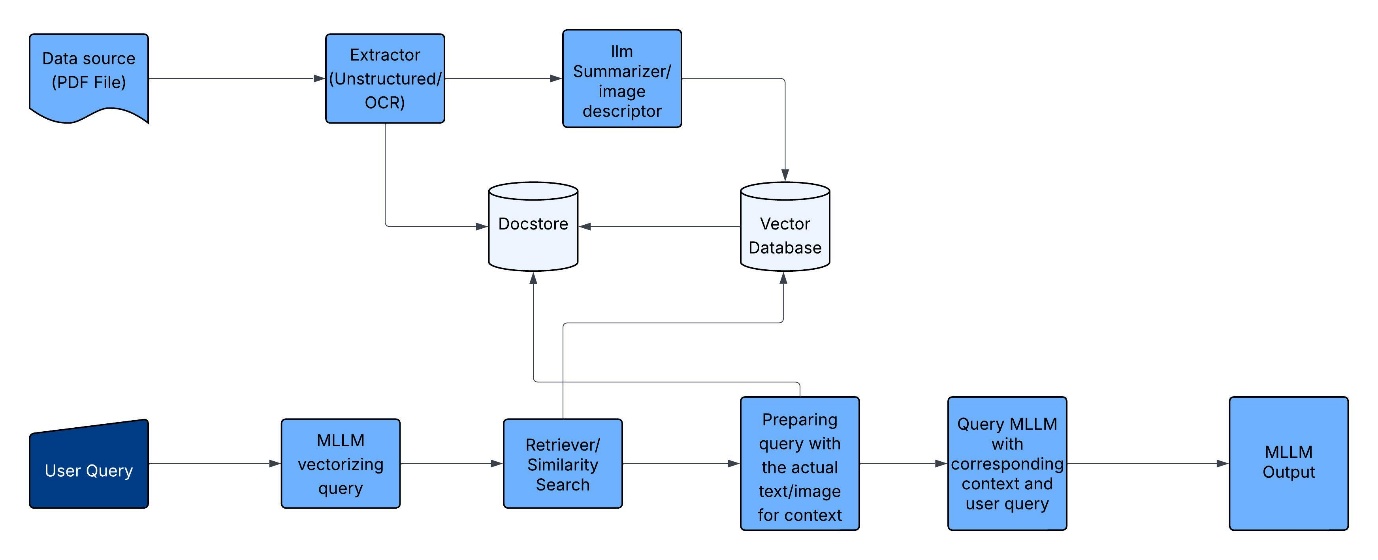


Figure : Architecture of Application

The architecture consists of the following major components:

* Web interface built with FastAPI.
* **Preprocessing Pipeline**: Extraction of multimodal elements from PDFs.
* **Summarization/Description Module**: Generating textual summaries and image descriptions via MLLM.
* **Vector Storage**: Storing multimodal embeddings in ChromaDB.
* **Metadata Storage**: Storing actual extracted text/images mapped by unique IDs (in memory).
* **Retrieval Engine**: Similarity search using embeddings.
* **Answer Generation**: Query answering via AWS Bedrock’s Claude 3.7 model with retrieved context.

**Technology Stack**  
  
- **Programming Language**: Python

- **Web Framework**: FastAPI

- **Large Multimodal Language Model**: Claude Sonnet 3.7 (via AWS Bedrock)

- **Vector Database**: ChromaDB

- **Key Libraries**:

* **Unstructured**: For extracting elements (text, tables, images) from PDFs.
* **LangChain**: For streamlined implementation of retrieval pipelines with ChromaDB.

**Future Improvements**  
1. Implement an **ensembled extraction pipeline** combining tools like [**Pix2Text**](https://github.com/breezedeus/Pix2Text) for more accurate extraction of images, tables, and mathematical equations.   
  
2. Introduce a **dedicated datastore** such as **MongoDB** or **PostgreSQL** to persist extracted raw text, images, and metadata, improving retrieval efficiency and system robustness.

3. Develop a **better ranking mechanism** for retrieval, possibly using re-ranking models or hybrid retrieval techniques (combining semantic and keyword search).  
  
  
  
Reference:  
[RAG Agents in Prod: 10 Lessons We Learned — Douwe Kiela, creator of RAG](https://www.youtube.com/watch?v=kPL-6-9MVyA&ab_channel=AIEngineer)