High-Level Design (HLD) Document

# 1. Introduction

This document provides the High-Level Design (HLD) of the Retrieval-Augmented Generation (RAG) system. The system is designed to ingest PDF documents (both native and scanned), create embeddings, store them in a vector database, and enable efficient question answering with citations and evidence highlighting. The implementation follows a modular architecture with clean separation of components.

# 2. Architecture Overview

The RAG system consists of the following major components:  
1. PDF Ingestion Pipeline  
2. Text Chunking & Embeddings  
3. Vector Index Storage  
4. Retriever & Generator  
5. Optional Web UI (for chat interface)

# 3. Data Flow

1. The system ingests PDF files (native and scanned).  
2. Text extraction is performed using PyPDF2/PyMuPDF for native PDFs and OCR (Tesseract) for scanned PDFs.  
3. Extracted text is split into smaller chunks using a chunking strategy.  
4. Each chunk is converted into embeddings using a Sentence Transformer model.  
5. Embeddings are stored in a vector database (FAISS).  
6. When a user asks a query, the retriever fetches the most relevant chunks based on semantic similarity.  
7. Retrieved context is passed to a local LLM (e.g., LLaMA/Mistral) to generate an answer.  
8. The system highlights the source and provides citations for transparency.  
9. (Optional) A web UI allows interactive Q&A over selected PDFs.

# 4. Key Design Choices

- \*\*Open-Source Stack\*\*: Ensures cost-effectiveness and flexibility (Tesseract, FAISS, HuggingFace models).  
- \*\*Embeddings Model\*\*: SentenceTransformers (all-MiniLM-L6-v2) chosen for balance between speed and accuracy.  
- \*\*Vector Index\*\*: FAISS selected for efficient similarity search.  
- \*\*LLM\*\*: Local model (e.g., Mistral-7B or LLaMA-2) used for data privacy and offline capability.  
- \*\*Modular Design\*\*: Each component (ingestion, embeddings, retriever, generator) implemented as a separate function/file.  
- \*\*Scanned PDF Support\*\*: Integrated OCR ensures usability across all PDF types.

# 5. Component-wise Description

**app.py** – Provides a simple demo Streamlit app (placeholder, shows sample responses).

**ui\_streamlit.py** – Main Streamlit UI for interactive Q&A over uploaded PDFs.

**build\_faiss.py** – Builds the FAISS index using pre-computed embeddings.

**build\_index.py** – Helper script for index building operations.

**chunk.py** – Splits extracted text into smaller, semantically meaningful chunks.

**embed\_index.py** – Generates embeddings for text chunks and saves them into FAISS index.

**ingest.py** – Extracts text from PDFs (using PyPDF2/PyMuPDF for native and Tesseract OCR for scanned).

**rag\_local.py** – Core RAG pipeline implementation. Handles retrieval from FAISS and generation via LLaMA3.

**requirements.txt** – Lists Python dependencies needed to run the project.

# 6. Quality & Evaluation

The system is evaluated on:  
- \*\*Accuracy\*\*: Correctness of retrieved answers.  
- \*\*Relevance\*\*: Semantic similarity between query and retrieved chunks.  
- \*\*Efficiency\*\*: Speed of retrieval and response generation.  
- \*\*Transparency\*\*: Citation and evidence highlighting ensures trustworthiness.

# 7.Repository Layout:-

End-to-End-RAG/

│

├── data/

│ └── index/ # Stores FAISS index and metadata

│

├── src/

│ ├── app.py # Simple demo Streamlit app (placeholder)

│ ├── ui\_streamlit.py # Main Streamlit UI for PDF Q&A

│ ├── build\_faiss.py # Builds FAISS index from embeddings

│ ├── build\_index.py # Helper script to build index

│ ├── chunk.py # Splits extracted text into chunks

│ ├── embed\_index.py # Generates embeddings and saves index

│ ├── ingest.py # Handles ingestion and text extraction (OCR/native)

│ ├── rag\_local.py # Core RAG pipeline with retriever + generator

│ └── requirements.txt # Python dependencies

│

└── RAG\_HLD.docx # High-Level Design document

# 8. Conclusion

This HLD document describes the design and architecture of the RAG system. The modular approach ensures maintainability, scalability, and adaptability for future improvements, including UI enhancements and integration with additional LLMs.