# **RAG Application Documentation**

### **Overview**

This application allows users to **upload a PDF file** and **ask questions** related to its content.  
 It uses a **Retrieval-Augmented Generation (RAG)** architecture to fetch the most relevant document chunks and generate context-aware answers — all **locally without any external API**.

## **🧠 How It Works**

The app combines two main components:

1. **Retriever** – Finds relevant text sections from the uploaded PDF using embeddings and FAISS vector search.
2. **Generator (LLM)** – Uses a local text-to-text model (google/flan-t5-small) to generate human-like answers.

The process flow:

1. Upload PDF → extract text.
2. Split text into small overlapping chunks.
3. Convert text chunks into numerical embeddings using **HuggingFace Sentence Transformers**.
4. Store and search these embeddings using **FAISS (Facebook AI Similarity Search)**.
5. Pass relevant chunks to the **Flan-T5 model** for answer generation.
6. Display question–answer history in Streamlit UI.

## **Tech Stack**

| **Component** | **Tool / Library** |
| --- | --- |
| **Frontend/UI** | Streamlit |
| **Document Loader** | LangChain PyPDFLoader |
| **Text Splitter** | RecursiveCharacterTextSplitter |
| **Embeddings** | HuggingFaceEmbeddings (sentence-transformers/all-MiniLM-L6-v2) |
| **Vector Store** | FAISS |
| **LLM (Generator)** | google/flan-t5-small via HuggingFace Transformers |
| **Framework** | LangChain |
| **Environment Management** | Python venv (Virtual Environment) |

## **⚙️ Installation & Setup**

### **1️⃣ Clone or Create the Project**

git clone <repo-url>

cd "RAG Application"

### **2️⃣ (Optional) Create a Virtual Environment**

python -m venv venv

venv\Scripts\activate # for Windows

# OR

source venv/bin/activate # for macOS/Linux

### **3️⃣ Install Required Packages**

pip install -r requirements.txt

## **🏃 Run the Application**

streamlit run app.py

Then open the URL shown in the terminal (usually http://localhost:8501).

## **🧾 Features**

✅ Upload any PDF file (up to 200MB).  
 ✅ Asks natural language questions about the document.  
 ✅ Displays previous Q&A history for conversational flow.  
 ✅ Runs fully offline (no OpenAI API required).  
 ✅ Uses lightweight local models for fast response.

**Example Interaction**

**Upload PDF:** *The Boy Who Could Talk to Crows.pdf*

**Question:**

“To whom is the boy talking?”

**Answer:**

“The boy is talking to the crow.”

## **🧩 Key Code Components**

### **Initialize Embeddings**

embeddings = HuggingFaceEmbeddings(model\_name="sentence-transformers/all-MiniLM-L6-v2")

### **Initialize Local LLM**

pipe = pipeline("text2text-generation", model="google/flan-t5-small", tokenizer="google/flan-t5-small")

llm = HuggingFacePipeline(pipeline=pipe)

### **Conversational RAG Chain**

st.session\_state.qa = ConversationalRetrievalChain.from\_llm(

llm=llm,

retriever=vectorstore.as\_retriever()

)

## **Understanding RAG Architecture**

**Retriever-Augmented Generation (RAG)** enhances LLM responses by combining:

* **Retrieval:** Get relevant facts from the uploaded PDF.
* **Generation:** Use those facts to generate natural answers.

This ensures:

* Contextually correct answers.
* No hallucination.
* Works even with limited local model capacity.

## **Troubleshooting**

| **Issue** | **Cause** | **Fix** |
| --- | --- | --- |
| **App not starting** | Streamlit not installed | pip install streamlit |
| **Model not found** | HuggingFace model missing | Check internet for first run |
| **FAISS import error** | Missing dependency | pip install faiss-cpu |
| **PDF not loading** | Invalid file | Re-upload a valid .pdf |
| **No answer / empty output** | Model too small | Try google/flan-t5-base |