# 📘 Full Documentation for Conversational RAG with PDF (LangChain + Groq + Streamlit)

## 🔹 1. Imports & Their Purpose

import os  
import streamlit as st  
from dotenv import load\_dotenv

* **os** → For environment variables & file operations.
* **streamlit** → Builds the web UI.
* **dotenv.load\_dotenv** → Loads API keys & secrets from .env file.

from langchain.memory import ConversationBufferMemory  
from langchain\_chroma import Chroma  
from langchain\_community.document\_loaders import PyMuPDFLoader  
from langchain\_community.embeddings import HuggingFaceEmbeddings  
from langchain\_text\_splitters import RecursiveCharacterTextSplitter  
from langchain.chains import ConversationalRetrievalChain  
from langchain\_groq import ChatGroq

* **ConversationBufferMemory** → Stores chat history across turns.
* **Chroma** → Vector database for storing/retrieving embeddings.
* **PyMuPDFLoader** → Extracts text from PDFs.
* **HuggingFaceEmbeddings** → Converts text chunks into numerical vectors.
* **RecursiveCharacterTextSplitter** → Splits large text into chunks for embedding.
* **ConversationalRetrievalChain** → Core RAG pipeline (retrieval + LLM + memory).
* **ChatGroq** → Wrapper for Groq-hosted LLMs.

## 🔹 2. Environment Variables Setup

load\_dotenv()  
hf\_token = os.getenv("HF\_token")  
os.environ["HK\_Token"] = hf\_token

* Loads **HuggingFace token** from .env.
* Sets it as HK\_Token for embedding models.

## 🔹 3. Embedding Model Initialization

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

* Uses a **sentence transformer** to convert text into vector embeddings.
* These embeddings are stored in Chroma DB for semantic search.

## 🔹 4. Streamlit User Interface

st.title("Conversational RAG with PDF")  
st.write("Upload PDFs and chat with their content")

* Sets **title** and **description** in Streamlit app.

api\_key = st.text\_input("Enter your Groq API key:", type="password")

* Prompts user to enter **Groq API key** securely.

## 🔹 5. LLM Setup (Groq)

if api\_key:  
 llm = ChatGroq(groq\_api\_key=api\_key, model\_name="gemma2-9b-it")

* Initializes **Groq-powered LLM** (gemma2-9b-it).
* Requires valid **Groq API key**.

## 🔹 6. Session Management

session\_id = st.text\_input("Session ID", value="default\_session")  
  
if "store" not in st.session\_state:  
 st.session\_state.store = {}

* Uses session\_state to persist memory across chat sessions.
* Allows multiple sessions via custom session\_id.

## 🔹 7. PDF Upload & Processing

uploaded\_file = st.file\_uploader("Choose a PDF file", type="pdf")  
  
if uploaded\_file:  
 with open("temp.pdf", "wb") as f:  
 f.write(uploaded\_file.getvalue())

* Lets user upload a PDF.
* Saves it temporarily as temp.pdf.

loader = PyMuPDFLoader("temp.pdf")  
documents = loader.load()

* Loads PDF into documents with **metadata + text**.

text\_splitter = RecursiveCharacterTextSplitter(chunk\_size=1000, chunk\_overlap=100)  
split\_docs = text\_splitter.split\_documents(documents)

* Splits documents into chunks of **1000 characters** with **100 overlap**.
* Ensures smooth retrieval.

## 🔹 8. Vector Store Setup

vectorstore = Chroma.from\_documents(documents=split\_docs, embedding=embeddings)  
retriever = vectorstore.as\_retriever()

* Stores chunk embeddings in **Chroma DB**.
* Creates a retriever that fetches relevant chunks during queries.

## 🔹 9. Memory Setup

if session\_id not in st.session\_state.store:  
 st.session\_state.store[session\_id] = ConversationBufferMemory(  
 memory\_key="chat\_history",  
 return\_messages=True  
 )  
memory = st.session\_state.store[session\_id]

* Creates **conversation memory** for each session.
* Stores user + assistant messages for contextual answers.

## 🔹 10. Conversational RAG Chain

qa\_chain = ConversationalRetrievalChain.from\_llm(  
 llm=llm,  
 retriever=retriever,  
 memory=memory,  
 output\_key="answer"  
)

* Combines:
  + **LLM (ChatGroq)**
  + **Retriever (Chroma)**
  + **Memory (ConversationBufferMemory)**
* output\_key="answer" ensures proper output format.

## 🔹 11. Chat Interaction

user\_input = st.text\_input("Your question:")  
if user\_input:  
 response = qa\_chain.invoke({"question": user\_input})

* Takes user input and queries the **RAG chain**.

st.markdown("### Assistant:")  
st.write(response["answer"])

* Displays assistant’s answer.

if "source\_documents" in response:  
 st.write("### Source Documents:")  
 for doc in response["source\_documents"]:  
 st.write(doc.page\_content[:300] + "...")

* Optionally shows **retrieved source documents**.

st.markdown("### Chat History")  
for msg in memory.chat\_memory.messages:  
 st.write(f"\*\*{msg.type.capitalize()}\*\*: {msg.content}")

* Prints **full chat history** (user + assistant messages).

## 🔹 12. Requirements

pip install streamlit langchain langchain-groq langchain-chroma langchain-community langchain-text-splitters python-dotenv pymupdf

## 🔹 13. Run App

streamlit run app.py

## 🔹 14. Example Run

* Upload **Research\_Paper.pdf**.
* Ask: *“Summarize section 3.”*
* Assistant responds with summary from that section.
* Chat history keeps track of previous Q&A.