# 📚 PDF Chatbot

A simple Streamlit web app that lets you chat with a PDF document using natural language. Ask questions and get answers based on the document contents — powered by Facebook Lama LLMs and LangChain.

## 🧠 Code Explanation:

Setup environment:

Firstly, you need to create API variable key exactly like below as we are going to work with Langchain, it requires your environment variable for the GROQ API exactly like this otherwise it will not recognize.

GROQ\_API\_KEY=gsk\_...

Dependencies are store in requirements.txt – install -r requirements.txt

Test your project is run directly in your terminal or not:

def main():

print(‘hello world’)

if \_\_name\_\_ == '\_\_main\_\_':

    main()

#Acess your API key from .env file:

import os# access my api key

from dotenv import load\_dotenv #load api environment

#setting the api key environment and load the api key in environment

def main():

    load\_dotenv()

    # print(os.getenv('GROQ\_API\_KEY'))#checking api key is loading or not

    groq\_api\_key = os.getenv('GROQ\_API\_KEY')

# Creating Graphical Interface

import streamlit as st

st.set\_page\_config(page\_title='Chat with pdf',page\_icon='books')# title of the page

st.header('PDF Chatbot :books:')# header of the page

with st.sidebar:#create a sidebar

        st.subheader('Upload a PDF file and click "Process" to start chat.')# create a header inside sidebar

        #uploading the file

        pdf = st.file\_uploader('',type='pdf')#create a file uploader

        # when clicking button

        if st.button('Process',"Primary Button", type="primary"):#create button

            with st.spinner('processing..'):#create spinner

Logic Part:

Step1: Get text out of your pdf.

from PyPDF2 import PdfReader# #get pdf text

if pdf is not None:# pdf is not null then,

     pdf\_reader = PdfReader(pdf) # create a object pdf\_reader of PdfReader

Pdf reader allow you to extract text only from the single page, so we need to extract by looping:

text = ''

for page in pdf\_reader.pages:

         text += page.extract\_text()

we concat the text by the extracting text from each page. We get whole text of the pdf until this point

# we have a problem now. We have a huge text, which we can’t directly pass to our LLM model.

Step2: Split the text into similar size chunks and look inside the chunks to see which chunks contain the information corresponding to our question and then feed those chunks to the language model.

**#divide text into chunks**

from langchain.text\_splitter import CharacterTextSplitter

 #split into chunks

text\_splitter = CharacterTextSplitter(separator='\n',chunk\_size=1000, chunk\_overlap=200,length\_function = len)

#seperator = new line which separate the chunks, chunk\_size = first 1000 characters and so on

#chunk\_overlap = repeate the 200 characters in every chunks

#length\_function = length of chunks

docs = text\_splitter.split\_text(text)#split text

Step3: Converting chunks into embedding:

Converting chunks to embeddings means turning small pieces of text into meaningful number vectors that represent their meaning, so a computer (or AI model) can understand and compare them.

from langchain.embeddings import HuggingFaceEmbeddings

from langchain\_community.vectorstores import FAISS

# Embed documents

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

vectorstore = FAISS.from\_texts(docs, embeddings)

In simple word

**sentence-transformers/all-MiniLM-L6-v2:**

It's a pre-trained language model that takes a sentence or paragraph and turns it into a vector of numbers (called an embedding) that captures its meaning.

It's a tool that understands the meaning of sentences and turns them into numbers so a computer can compare or search them more easily.

After embedding, we can actually create our object ‘embeddings’ on which we are able to search. So, in order to search in chunks, we use FAISS (Facebook AI Similarity Search)

**FAISS**:

FAISS is a library that helps you search through a lot of text embeddings. It is used to store all the vectors in the vector store or knowledge base. When user ask the question, question is also embedded with same technique and FAISS finds the most similar chunks based on the meaning, not just exact words.

LLM initialized:

Here we initialized our LLM that is llama3 from Facebook using Groq API key.

**retriever=vectorstore.as\_retriever()** - This turns the FAISS vector store into a retriever. It finds similar chunks (based on the user's question).

**RetrievalQA.from\_chain\_type(...)**: This builds a special chain that connects the retriever to the LLM:

First retrieves relevant info, then sends it to the LLM, And gives the answer back.

RetrievalQA = Search + Answer. It combines

Retriever: A system that finds the most relevant chunks from your PDF.

LLM: The brain that reads those chunks and answers your question.

st.session\_state.qa\_chain = ... This saves the entire pipeline in Streamlit's memory, so it stays available across interactions (as long as the app is open).

from langchain.chains.question\_answering import load\_qa\_chain

from langchain\_core.prompts import PromptTemplate

from langchain\_groq import ChatGroq

from langchain.chains import RetrievalQA

llm = ChatGroq(groq\_api\_key=groq\_api\_key, model\_name="llama3-8b-8192")

st.session\_state.qa\_chain = RetrievalQA.from\_chain\_type(llm=llm, retriever=vectorstore.as\_retriever())

User interaction part:

First check qa\_chain is created earlier or not as it creates a chain between retriever and LLM .

If yes then, it will create a question box saying ‘Ask a question from the pdf?’. If no, box is not created as there is no knowledge base to query.

If user ask the question or type something in the box and enter, qa\_chain is run. The question is embedded using same hugging face sentence transformer and retriever search for the similar chunks of that questions in vector store and send it to LLM. LLM use the brain and give it as a response

if "qa\_chain" in st.session\_state:

      user\_question = st.text\_input("Ask a question from the PDF:")

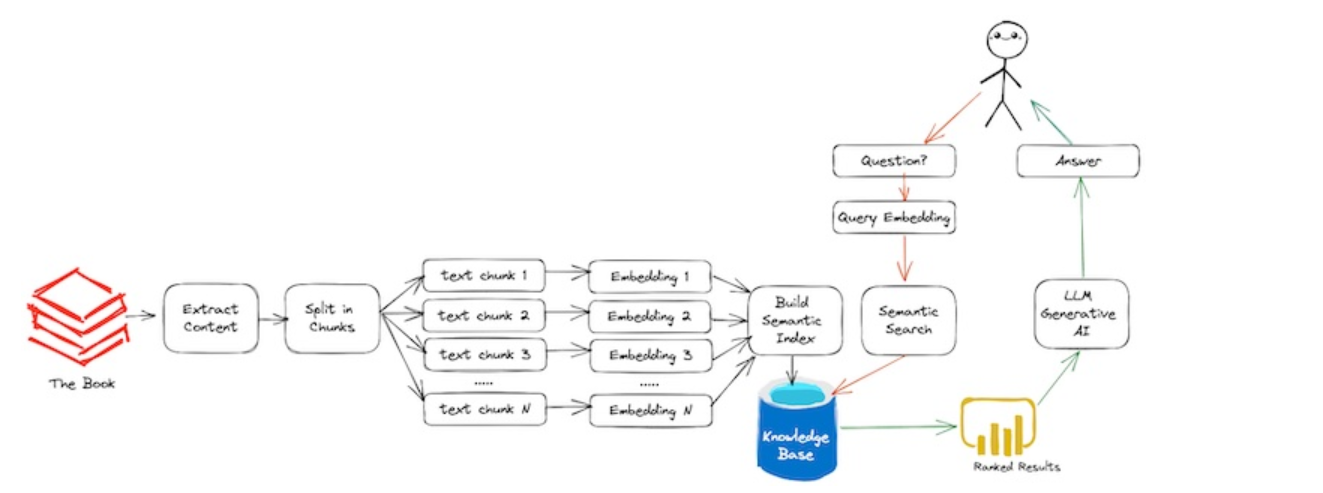
        if user\_question:

            response = st.session\_state.qa\_chain.run(user\_question)

            st.write("### 🧠🤖 Answer:")

            st.write(response)

**Talking to Your PDFs: Behind the Scenes of an LLM Chatbot**



1. Extract the content with pdf reader
2. Split in chunks with same size for example: 1000 chunks
3. Each of chunks are going to convert into embeddings which are the representation of the texts means you can say it as a list of numbers that contain meaning of your text called vector which is the number representation of your meaning of your text
4. Then these vectors are going to be store in your **knowledge base.**
5. When user come and ask the questions, question is going to be embedded using same embedding technique that we use for the text chunks and that is going to allow us to perform the semantic search which is going to find the vectors in knowledge base that are similar to our vector question. This will allow us to find the chunks that actually contain information we need.
6. And those chunks we are going to feed our language model and we are going to get our answers.

## 🚀 Features

- 🧠 Ask questions from any uploaded PDF

- 📎 Upload and process PDF files

- 🤖 Uses LangChain + LLM backend (Groq API)

- 🌐 Streamlit UI for fast, interactive use

- 💬 Supports follow-up questions

---

## 📸 Screenshot

![PDF Chatbot Screenshot](screenshot.png)

---

## 🛠️ Tech Stack

- [Streamlit](https://streamlit.io/)

- [LangChain](https://www.langchain.com/)

- [Groq API](https://console.groq.com/)

- HuggingFace Embeddings

- Python

---

## 📁 Folder Structure

---

## 🔧 How to Run Locally

1. Clone the repository:

```bash

git clone https://github.com/dipen-sherpa/pdf-chatbot.git

cd pdf-chatbot

pip install -r requirements.txt

GROQ\_API\_KEY=your\_key\_here

HUGGINGFACEHUB\_API\_TOKEN=your\_token\_here

streamlit run app.py

This project is open-source and free to use under the MIT License.

yaml

CopyEdit

---

## ✅ How to Use It

1. Save it as `README.md` in your GitHub repo

2. Replace placeholders:

- Screenshot

- API keys

- Your GitHub + LinkedIn

- Streamlit app link

3. Commit and push:

```bash

git add README.md

git commit -m "Add project README"

git push origin main