```
!pip install streamlit
!pip install PyPDF2
!pip install langchain
!pip install -U langchain-community
!pip install faiss-cpu
!pip install openai
```

Show hidden output

```
%%writefile app.py
import streamlit as st
from PyPDF2 import PdfReader
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain.embeddings.openai import OpenAIEmbeddings
from langchain_community.vectorstores import FAISS
from langchain.chains.question_answering import load_qa_chain
from langchain_community.chat_models import ChatOpenAI

OPENAI_API_KEY=" your API KEY"

#upload pdf files
st.header("ChatBot")
with st.sidebar:
  st.title("File Upload")
  file=st.file_uploader("Choose a pdf file", type="pdf")

# #xtracting the text from pdf files.
text = ""
if file is not None:
  pdf_reader=PdfReader(file)
  for page in pdf_reader.pages:
    text+=page.extract_text()
else:
  st.warning("Please upload a pdf file")
'''
breaking the text into chunks
text splitter will be used from langchain.
'''
text_splitter=RecursiveCharacterTextSplitter(
  separators="\n",
  chunk_size=1000,
  chunk_overlap=150, # so that the relation and the meaning of the chunks is preserved, to save incomplete sentences.
  length_function=len # breaking based on 1000 characters as length is given
)
chunks=text_splitter.split_text(text)
#-----------------------------------------------------------------------
'''
post geberaton of the chunks, we will create those embeddings and then store them in a vector store
this is like a database to store the embeddings.
'''
#-----------------------------------------------------------------------
# using openai services with langchain to do so
embeddings=OpenAIEmbeddings(openai_api_key=OPENAI_API_KEY)
# creating our vector store using FAISS - Facebook AI Semantic Search, Again Langchain will be used.
vector_store=FAISS.from_texts(chunks, embeddings) # it will take in the chunks and the embeddings.
'''
# - generated embeddings with OpenAI services
# - created/initialized the FAISS vector store
# - stores the chunks and embeddings
'''
#-----------------------------------------------------------------------
'''
We will proceed to Finetuning sectiom
1. get user's question
2. do similarity search
3. output results
'''
#defining our LLM:
llm = ChatOpenAI(
    open_api_key=OPENAI_API_KEY,
    temperature=0, # reducing the randomness, lower the value more specific the answer and not random
    max_tokens=1000,
    model_name="gpt-3.5-turbo"
)
user_question=st.text_input("Ask your Question")
if user_question:
  matches=vector_store.similarity_search(user_question)
  chain=load_qa_chain=load_qa_chain(llm, chain_type="stuff")
  response=chain.run(input_documents=matches, question=user_question)
  st.write(response)
```

```
Overwriting app.py
```

```
!npm install localtunnel
```

```
!streamlit run app.py &> logs.txt & curl ipv4.icanhazip.com
```

```
!npx localtunnel --port (port)
```