Python Fast API Developer with Open AI/LLMs

**Aim**:

1. Create a conversational PDF chatbot using Opne AI APIs and Streamlit. Allow the user to upload a PDF file and chat with the PDF contents

1. Set Up Your Environment: Make sure you have Python installed along with necessary libraries like Streamlit, OpenAI API client, and any other dependencies you might need.

2. Develop the Streamlit App: Create a Streamlit web application where users can upload their PDF files. You can use Streamlit's file\_uploader widget for this.

3. Extract Text from PDF: Utilize a library like PyPDF2 or pdfplumber to extract text from the uploaded PDF file. Process the text to remove any unnecessary characters or formatting.

4. Integrate OpenAI API: Set up your OpenAI API client and utilize it to generate responses based on user queries. You may need to preprocess the text from the PDF to provide it as input to the API.

5. Display Conversational Interface: Use Streamlit's text input or other interactive components to allow users to ask questions or provide prompts. Display the responses generated by the OpenAI API in a conversational format.

6. Handle Edge Cases: Account for edge cases such as when the PDF contains images or complex formatting that cannot be easily converted to text.

7. Testing and Refinement: Test your chatbot with various PDF files to ensure it works smoothly and handles different types of content effectively. Gather feedback from users and refine your app accordingly.

8. Deployment: Once you're satisfied with your chatbot, consider deploying it using platforms like Heroku or Streamlit sharing so others can access it.

# RUN THIS CELL FIRST!

!pip install -q langchain==0.0.150 pypdf pandas matplotlib tiktoken textract transformers openai faiss-cpu

import os

import pandas as pd

import matplotlib.pyplot as plt

from transformers import GPT2TokenizerFast

from langchain.document\_loaders import PyPDFLoader

from langchain.text\_splitter import RecursiveCharacterTextSplitter

from langchain.embeddings import OpenAIEmbeddings

from langchain.vectorstores import FAISS

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

from langchain.llms import OpenAI

from langchain.chains import ConversationalRetrievalChain

# You MUST add your PDF to local files in this notebook (folder icon on left hand side of screen)

# Simple method - Split by pages

loader = PyPDFLoader("./attention\_is\_all\_you\_need.pdf")

pages = loader.load\_and\_split()

print(pages[0])

# SKIP TO STEP 2 IF YOU'RE USING THIS METHOD

chunks = pages

# Quick data visualization to ensure chunking was successful

# Create a list of token counts

token\_counts = [count\_tokens(chunk.page\_content) for chunk in chunks]

# Create a DataFrame from the token counts

df = pd.DataFrame({'Token Count': token\_counts})

# Create a histogram of the token count distribution

df.hist(bins=40, )

# Show the plot

plt.show()# Get embedding model

embeddings = OpenAIEmbeddings()

# Create vector database

db = FAISS.from\_documents(chunks, embeddings)

# Create QA chain to integrate similarity search with user queries (answer query from knowledge base)

chain = load\_qa\_chain(OpenAI(temperature=0), chain\_type="stuff")

query = "Who created transformers?"

docs = db.similarity\_search(query)

chain.run(input\_documents=docs, question=query)

chat\_history = []

def on\_submit(\_):

    query = input\_box.value

    input\_box.value = ""

    if query.lower() == 'exit':

        print("Thank you for using the State of the Union chatbot!")

        return

    result = qa({"question": query, "chat\_history": chat\_history})

    chat\_history.append((query, result['answer']))

    display(widgets.HTML(f'<b>User:</b> {query}'))

    display(widgets.HTML(f'<b><font color="blue">Chatbot:</font></b> {result["answer"]}'))

print("Welcome to the Transformers chatbot! Type 'exit' to stop.")

input\_box = widgets.Text(placeholder='Please enter your question:')

input\_box.on\_submit(on\_submit)

display(input\_box)

sssx

