##

Hugghing face codes :

##

App.py :

from flask import Flask, render\_template,jsonify,request

from src.helper import download\_embedding

from langchain\_pinecone import PineconeVectorStore

#from langchain\_openai import ChatOpenAI

from langchain.chains import create\_retrieval\_chain

from langchain.chains.combine\_documents import create\_stuff\_documents\_chain

from langchain\_core.prompts import ChatPromptTemplate

from langchain\_community.llms import HuggingFaceHub

#from langchain\_huggingface import HuggingFaceHub

from langchain\_community.chat\_models import ChatHuggingFace

from dotenv import load\_dotenv

from src.prompt import \*

import os

app= Flask(\_\_name\_\_)

load\_dotenv()

PINECONE\_API\_KEY = os.getenv("PINECONE\_API\_KEY")

#OPENAI\_API\_KEY = os.getenv("OPENAI\_API\_KEY")

HUGGINGFACEHUB\_API\_TOKEN= os.getenv("HUGGINGFACEHUB\_API\_TOKEN")

os.environ["PINECONE\_API\_KEY"]= PINECONE\_API\_KEY

#os.environ["OPENAI\_API\_KEY"]= OPENAI\_API\_KEY

os.environ["HUGGINGFACEHUB\_API\_TOKEN"]=HUGGINGFACEHUB\_API\_TOKEN

embedding = download\_embedding()

index\_name = "medical-chatbot"

docsearch = PineconeVectorStore.from\_existing\_index(

     embedding = embedding,

     index\_name = index\_name

)

retriever = docsearch.as\_retriever(search\_type="similarity",search\_kwargs={"k":3})

#gives llm based answer in vector form

#chatModelLLM = ChatOpenAI(model="gpt-4o")

prompt = ChatPromptTemplate.from\_messages(

    [

        ("system", system\_prompt),

        ("human", "{input}"),

    ]

)

repo\_id = "medalpaca/medalpaca-7b"

hf\_llm=HuggingFaceHub(

    repo\_id = repo\_id,

    huggingfacehub\_api\_token=HUGGINGFACEHUB\_API\_TOKEN,

    model\_kwargs={"temperature":0.0,"max\_new\_tokens":512}

)

chatModelLLM=ChatHuggingFace(llm=hf\_llm)

#main chaining part

question\_answer\_chain = create\_stuff\_documents\_chain(chatModelLLM,prompt)

rag\_chain = create\_retrieval\_chain(retriever , question\_answer\_chain)

#now answer from vector format --> word format (user understanding)

@app.route("/")

def index():

    return render\_template('chat.html')

@app.route("/get",methods=["GET","POST"])

def chat():

    msg = request.form["msg"]

    input = msg

    print(input)

    response = rag\_chain.invoke({"input":msg})

    print("Response : ", response["answer"])

    return str(response["answer"])

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

    app.run(host="0.0.0.0",port=8080,debug = True)

………..

# app.py

from flask import Flask, render\_template, request

from src.helper import download\_embedding

from langchain\_pinecone import PineconeVectorStore

from langchain.chains import create\_retrieval\_chain, create\_stuff\_documents\_chain

from langchain\_core.prompts import ChatPromptTemplate

from langchain\_community.llms import HuggingFaceHub

from langchain\_community.chat\_models import ChatHuggingFace

from dotenv import load\_dotenv

from src.prompt import system\_prompt

import os

app = Flask(\_\_name\_\_)

load\_dotenv()

PINECONE\_API\_KEY = os.getenv("PINECONE\_API\_KEY")

HUGGINGFACEHUB\_API\_TOKEN = os.getenv("HUGGINGFACEHUB\_API\_TOKEN")

os.environ["PINECONE\_API\_KEY"] = PINECONE\_API\_KEY

os.environ["HUGGINGFACEHUB\_API\_TOKEN"] = HUGGINGFACEHUB\_API\_TOKEN

# -------------------------------

# Embedding & Pinecone

# -------------------------------

embedding = download\_embedding()

index\_name = "medical-chatbot"

docsearch = PineconeVectorStore.from\_existing\_index(

    embedding=embedding,

    index\_name=index\_name

)

retriever = docsearch.as\_retriever(search\_type="similarity", search\_kwargs={"k": 3})

# -------------------------------

# LLM setup

# -------------------------------

repo\_id = "medalpaca/medalpaca-7b"

hf\_llm = HuggingFaceHub(

    repo\_id=repo\_id,

    huggingfacehub\_api\_token=HUGGINGFACEHUB\_API\_TOKEN,

    model\_kwargs={"temperature": 0.0, "max\_new\_tokens": 512}

)

chatModelLLM = ChatHuggingFace(llm=hf\_llm)

# -------------------------------

# RAG chain

# -------------------------------

prompt = ChatPromptTemplate.from\_messages([("system", system\_prompt), ("human", "{input}")])

question\_answer\_chain = create\_stuff\_documents\_chain(chatModelLLM, prompt)

rag\_chain = create\_retrieval\_chain(retriever, question\_answer\_chain)

# -------------------------------

# Flask routes

# -------------------------------

@app.route("/")

def index():

    return render\_template('chat.html')

@app.route("/get", methods=["POST"])

def chat():

    msg = request.form["msg"]

    response = rag\_chain.invoke({"input": msg})

    return str(response["answer"])

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

    app.run(host="0.0.0.0", port=8080, debug=True)

##

Helper.py

from langchain\_community.document\_loaders import PyPDFLoader , DirectoryLoader

from langchain.text\_splitter import RecursiveCharacterTextSplitter

from langchain\_community.embeddings import HuggingFaceHubEmbeddings

from typing import List

from langchain.schema import Document

import os

#now to create function to extract pdf files

def load\_pdf\_file(data):

    loader=DirectoryLoader(

        data,

        glob="\*.pdf",

        loader\_cls=PyPDFLoader

    )

    documents=loader.load()

    return documents

def filter\_to\_minimal\_docs(docs: List[Document]) -> List[Document]:

    """Given a list of documents objects , return a new list of Documents objects containing only 'source'

     in metadata and the original page\_content.

     """

    minimal\_docs: List[Document]=[]

    for doc in docs:

        src=doc.metadata.get("source")

        minimal\_docs.append(

            Document(

                page\_content = doc.page\_content,

                metadata ={"source":src}

            )

        )

    return minimal\_docs

#therefore now lets start with chunking of the data

def text\_split(minimal\_docs):

    text\_splitter=RecursiveCharacterTextSplitter(

        chunk\_size=500,

        chunk\_overlap=20,

    )

    text\_chunk= text\_splitter.split\_documents(minimal\_docs)

    return text\_chunk

def download\_embedding():

   """

   Download and return the HuggingFaceEmbedding Models

   //changes currently using biomedical PubMed embeddings for more relevant retrieval

   """

   #model\_name="sentence-transformers/all-MiniLM-L6-v2"

   model\_name="NeuML/pubmedbert-base-embeddings"

   HUGGINGFACEHUB\_API\_TOKEN= os.getenv("HUGGINGFACEHUB\_API\_TOKEN")

   embedding=HuggingFaceHubEmbeddings(

      model\_name = model\_name,

      ##

      task="feature-extraction",

      huggingfacehub\_api\_token=HUGGINGFACEHUB\_API\_TOKEN

   )

   return embedding

……………………

# src/helper.py

from langchain\_community.document\_loaders import PyPDFLoader, DirectoryLoader

from langchain.text\_splitter import RecursiveCharacterTextSplitter

from langchain\_huggingface import HuggingFaceEndpointEmbeddings

#from langchain\_community.embeddings import HuggingFaceEmbeddings

from typing import List

from langchain.schema import Document

from langchain\_huggingface import HuggingFaceEndpointEmbeddings

import os

def load\_pdf\_file(data\_path):

    loader = DirectoryLoader(

        data\_path,

        glob="\*.pdf",

        loader\_cls=PyPDFLoader

    )

    documents = loader.load()

    return documents

def filter\_to\_minimal\_docs(docs: List[Document]) -> List[Document]:

    minimal\_docs: List[Document] = []

    for doc in docs:

        src = doc.metadata.get("source")

        minimal\_docs.append(

            Document(

                page\_content=doc.page\_content,

                metadata={"source": src}

            )

        )

    return minimal\_docs

def text\_split(minimal\_docs):

    text\_splitter = RecursiveCharacterTextSplitter(

        chunk\_size=500,

        chunk\_overlap=20,

    )

    return text\_splitter.split\_documents(minimal\_docs)

def download\_embedding():

    from langchain\_huggingface import HuggingFaceEndpointEmbeddings

    import os

    model\_name = "sentence-transformers/all-MiniLM-L6-v2"

    hf\_token = os.getenv("HUGGINGFACEHUB\_API\_TOKEN")

    embedding = HuggingFaceEndpointEmbeddings(

        model=model\_name,

        huggingfacehub\_api\_token=hf\_token

    )

    return embedding

##

Prompt.py

#NOW PROMPT TEMPLATE CREATION

"""

system\_prompt=(

    "You are an Medical assistant for question-answering tasks. "

    "Use the following pieces of retrieved context to answer "

    "the question .If you don't know the answer , say that you "

    "dont know . Use three sentences maximum and keep the "

    "answer concise "

    "\n\n"

    "{context}"

)

"""

system\_prompt = (

    "You are a Medical assistant for question-answering tasks. "

    "Use the following pieces of retrieved context to answer "

    "the question. If you don't know the answer, say that you "

    "don't know. Use three sentences maximum and keep the "

    "answer concise.\n\n"

    "{context}"

)

………………

# src/prompt.py

system\_prompt = (

    "You are a medical assistant for question-answering tasks. "

    "Use the following retrieved context to answer the question. "

    "If you don't know the answer, say that you don't know. "

    "Use a maximum of three sentences and keep the answer concise.\n\n"

    "{context}"

)

##

Store\_index.py

from dotenv import load\_dotenv

import os

from src.helper import load\_pdf\_file,filter\_to\_minimal\_docs,text\_split,download\_embedding

from pinecone import Pinecone

from pinecone import ServerlessSpec

from langchain\_pinecone import PineconeVectorStore

load\_dotenv()

PINECONE\_API\_KEY = os.getenv("PINECONE\_API\_KEY")

#OPENAI\_API\_KEY = os.getenv("OPENAI\_API\_KEY")

HUGGINGFACEHUB\_API\_TOKEN= os.getenv("HUGGINGFACEHUB\_API\_TOKEN")

os.environ["PINECONE\_API\_KEY"]= PINECONE\_API\_KEY

#os.environ["OPENAI\_API\_KEY"]= OPENAI\_API\_KEY

os.environ["HUGGINGFACEHUB\_API\_TOKEN"]=HUGGINGFACEHUB\_API\_TOKEN

extracted\_data = load\_pdf\_file(r"C:\Users\Krishna Gupta\Desktop\health\_chatbot\data")

minimal\_docs = filter\_to\_minimal\_docs(extracted\_data)

text\_chunk=text\_split(minimal\_docs)

embedding = download\_embedding()

pinecone\_api\_key = PINECONE\_API\_KEY

pc = Pinecone(api\_key=pinecone\_api\_key)

index\_name = "medical-chatbot"

print("existing indexes:", pc.list\_indexes())

if index\_name not in [i["name"] for i in pc.list\_indexes()]:

    pc.create\_index(

        name=index\_name,

        dimension=768,   #dimension in embedding

        metric="cosine",

        spec=ServerlessSpec(cloud="aws", region="us-east-1")

    )

index = pc.Index(index\_name)

#embed each chunk and upsert the embedding into your pinecone index

docsearch = PineconeVectorStore.from\_existing\_index(

     embedding = embedding,

     index\_name = index\_name

)

…………..

# src/store\_index.py

from dotenv import load\_dotenv

import os

from src.helper import load\_pdf\_file, filter\_to\_minimal\_docs, text\_split, download\_embedding

from pinecone import Pinecone, ServerlessSpec

from langchain\_pinecone import PineconeVectorStore

load\_dotenv()

PINECONE\_API\_KEY = os.getenv("PINECONE\_API\_KEY")

HUGGINGFACEHUB\_API\_TOKEN = os.getenv("HUGGINGFACEHUB\_API\_TOKEN")

os.environ["PINECONE\_API\_KEY"] = PINECONE\_API\_KEY

os.environ["HUGGINGFACEHUB\_API\_TOKEN"] = HUGGINGFACEHUB\_API\_TOKEN

# -------------------------------

# Load & process PDFs

# -------------------------------

extracted\_data = load\_pdf\_file(r"C:\Users\Krishna Gupta\Desktop\health\_chatbot\data")

minimal\_docs = filter\_to\_minimal\_docs(extracted\_data)

text\_chunks = text\_split(minimal\_docs)

embedding = download\_embedding()

# -------------------------------

# Pinecone setup

# -------------------------------

pc = Pinecone(api\_key=PINECONE\_API\_KEY)

index\_name = "medical-chatbot"

print("Existing indexes:", pc.list\_indexes())

if index\_name not in [i["name"] for i in pc.list\_indexes()]:

    pc.create\_index(

        name=index\_name,

        dimension=768,  # Embedding dimension

        metric="cosine",

        spec=ServerlessSpec(cloud="aws", region="us-east-1")

    )

index = pc.Index(index\_name)

# -------------------------------

# Upsert chunks into Pinecone

# -------------------------------

docsearch = PineconeVectorStore.from\_documents(

    documents=text\_chunks,

    embedding=embedding,

    index\_name=index\_name

)

print("Indexing complete!")

#chat.html

<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8" />

  <title>Medical Chatbot</title>

  <link rel="stylesheet" href="{{ url\_for('static', filename='style.css') }}" />

</head>

<body>

  <div class="chat-container">

    <div class="chat-header">

      <h2>Health Assistant</h2>

    </div>

    <div id="chat-window" class="chat-window">

      <!-- Messages will appear here -->

    </div>

    <form id="chat-form" class="chat-form">

      <input

        type="text"

        id="user-input"

        placeholder="Type symptoms or questions..."

        autocomplete="off"

        required

      />

      <button type="submit">Send</button>

    </form>

  </div>

  <script>

    const form = document.getElementById('chat-form');

    const inputField = document.getElementById('user-input');

    const chatWindow = document.getElementById('chat-window');

    form.addEventListener('submit', async (e) => {

      e.preventDefault();

      const userText = inputField.value.trim();

      if (!userText) return;

      addMessage('user', userText);

      inputField.value = '';

      try {

        const response = await fetch('/get', {

          method: 'POST',

          headers: {

            'Content-Type': 'application/x-www-form-urlencoded'

          },

          body: new URLSearchParams({ msg: userText })

        });

        if (!response.ok) {

          throw new Error('Network response was not ok');

        }

        const botReply = await response.text();

        addMessage('bot', botReply);

      } catch (err) {

        console.error('Error:', err);

        addMessage('bot', 'Sorry, something went wrong.');

      }

    });

    function addMessage(sender, text) {

      const msgDiv = document.createElement('div');

      msgDiv.classList.add('message', sender);

      msgDiv.innerText = text;

      chatWindow.appendChild(msgDiv);

      chatWindow.scrollTop = chatWindow.scrollHeight;

    }

  </script>

</body>

</html>

#style.css

/\* Reset some defaults \*/

\* {

  box-sizing: border-box;

  margin: 0;

  padding: 0;

  font-family: Arial, sans-serif;

}

body {

  background-color: #f5f7fa;

  display: flex;

  justify-content: center;

  align-items: center;

  height: 100vh;

}

.chat-container {

  width: 400px;

  max-width: 95%;

  background-color: #ffffff;

  border-radius: 10px;

  box-shadow: 0 4px 10px rgba(0,0,0,0.1);

  display: flex;

  flex-direction: column;

  overflow: hidden;

}

.chat-header {

  background-color: #3498db;

  color: white;

  padding: 15px;

  text-align: center;

}

.chat-window {

  flex: 1;

  padding: 15px;

  overflow-y: auto;

}

.chat-form {

  display: flex;

  border-top: 1px solid #ddd;

}

.chat-form input[type="text"] {

  flex: 1;

  padding: 10px;

  border: none;

  border-top-left-radius: 0;

  border-bottom-left-radius: 0;

  font-size: 16px;

}

.chat-form button {

  padding: 10px 15px;

  border: none;

  background-color: #3498db;

  color: white;

  cursor: pointer;

  font-size: 16px;

}

.chat-form button:hover {

  background-color: #2980b9;

}

.message {

  margin: 10px 0;

  padding: 10px 15px;

  border-radius: 20px;

  max-width: 80%;

  word-wrap: break-word;

}

.message.user {

  background-color: #dcf8c6;

  align-self: flex-end;

}

.message.bot {

  background-color: #eee;

  align-self: flex-start;

}