import streamlit as st

from streamlit\_chat import message

from langchain.chains import ConversationalRetrievalChain

from langchain.embeddings import HuggingFaceEmbeddings

from langchain.llms import CTransformers

from langchain.llms import Replicate

from langchain.text\_splitter import CharacterTextSplitter

from langchain.vectorstores import FAISS

from langchain.memory import ConversationBufferMemory

from langchain.document\_loaders import PyPDFLoader

from langchain.document\_loaders import TextLoader

from langchain.document\_loaders import Docx2txtLoader

from langchain.callbacks.streaming\_stdout import StreamingStdOutCallbackHandler

import os

from dotenv import load\_dotenv

import tempfile

load\_dotenv()

def initialize\_session\_state():

if 'history' not in st.session\_state:

st.session\_state['history'] = []

if 'generated' not in st.session\_state:

st.session\_state['generated'] = ["Hello! Ask me anything about 🤗"]

if 'past' not in st.session\_state:

st.session\_state['past'] = ["Hey! 👋"]

def conversation\_chat(query, chain, history):

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

history.append((query, result["answer"]))

return result["answer"]

def display\_chat\_history(chain):

reply\_container = st.container()

container = st.container()

with container:

with st.form(key='my\_form', clear\_on\_submit=True):

user\_input = st.text\_input("Question:", placeholder="Ask about your Documents", key='input')

submit\_button = st.form\_submit\_button(label='Send')

if submit\_button and user\_input:

with st.spinner('Generating response...'):

output = conversation\_chat(user\_input, chain, st.session\_state['history'])

st.session\_state['past'].append(user\_input)

st.session\_state['generated'].append(output)

if st.session\_state['generated']:

with reply\_container:

for i in range(len(st.session\_state['generated'])):

message(st.session\_state["past"][i], is\_user=True, key=str(i) + '\_user', avatar\_style="thumbs")

message(st.session\_state["generated"][i], key=str(i), avatar\_style="")

def create\_conversational\_chain(vector\_store):

load\_dotenv()

# Create llm

#llm = CTransformers(model="llama-2-7b-chat.ggmlv3.q4\_0.bin",

#streaming=True,

#callbacks=[StreamingStdOutCallbackHandler()],

#model\_type="llama", config={'max\_new\_tokens': 500, 'temperature': 0.01})

llm = Replicate(

streaming = True,

# model = "replicate/llama-2-70b-chat:58d078176e02c219e11eb4da5a02a7830a283b14cf8f94537af893ccff5ee781",

# model = "a16z-infra/llama-2-7b-chat:d24902e3fa9b698cc208b5e63136c4e26e828659a9f09827ca6ec5bb83014381",

model = "a16z-infra/llama-2-13b-chat:9dff94b1bed5af738655d4a7cbcdcde2bd503aa85c94334fe1f42af7f3dd5ee3",

# model = "replicate/llama-2-70b-chat:2796ee9483c3fd7aa2e171d38f4ca12251a30609463dcfd4cd76703f22e96cdf",

callbacks=[StreamingStdOutCallbackHandler()],

input = {"temperature": 0.01, "max\_length" :500,"top\_p":1})

memory = ConversationBufferMemory(memory\_key="chat\_history", return\_messages=True)

chain = ConversationalRetrievalChain.from\_llm(llm=llm, chain\_type='stuff',

retriever=vector\_store.as\_retriever(search\_kwargs={"k": 2}),

memory=memory)

return chain

def main():

load\_dotenv()

# Initialize session state

initialize\_session\_state()

st.title("SurgeLab QA Chatbot")

# Initialize Streamlit

st.sidebar.title("Document Processing")

uploaded\_files = st.sidebar.file\_uploader("Upload files", accept\_multiple\_files=True)

if uploaded\_files:

text = []

for file in uploaded\_files:

file\_extension = os.path.splitext(file.name)[1]

with tempfile.NamedTemporaryFile(delete=False) as temp\_file:

temp\_file.write(file.read())

temp\_file\_path = temp\_file.name

loader = None

if file\_extension == ".pdf":

loader = PyPDFLoader(temp\_file\_path)

elif file\_extension == ".docx" or file\_extension == ".doc":

loader = Docx2txtLoader(temp\_file\_path)

elif file\_extension == ".txt":

loader = TextLoader(temp\_file\_path)

if loader:

text.extend(loader.load())

os.remove(temp\_file\_path)

text\_splitter = CharacterTextSplitter(separator="\n", chunk\_size=10000, chunk\_overlap=100, length\_function=len)

text\_chunks = text\_splitter.split\_documents(text)

# Create embeddings

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

model\_kwargs={'device': 'cpu'})

# Create vector store

vector\_store = FAISS.from\_documents(text\_chunks, embedding=embeddings)

# Create the chain object

chain = create\_conversational\_chain(vector\_store)

display\_chat\_history(chain)

if \_\_name\_\_ == "\_\_main\_\_":

main()