import streamlit as st

from sentence\_transformers import SentenceTransformer

import faiss

import pdfplumber

import re

import groq

import os

import numpy as np

# Set up Groq API key

GROQ\_API\_KEY = "gsk\_tPa1ncWKwbl54mNg9ASJWGdyb3FY043j9xfZznG1dEgXllcnpYz3"

client = groq.Client(api\_key=GROQ\_API\_KEY)

# Define file paths for legal texts

BOOKS = {

"Pakistan Penal Code": "/content/drive/MyDrive/pakistan penal code\_removed.pdf",

"Code of Criminal Procedure": "/content/drive/MyDrive/code of criminal procedure\_removed.pdf",

"Code of Civil Procedure": "/content/drive/MyDrive/code of civil procedure\_removed\_removed.pdf",

"Constitution of Pakistan": "/content/drive/MyDrive/constitution of pakistan\_removed.pdf",

"Rules of Business": "/content/drive/MyDrive/rules of business\_removed.pdf"

}

# Function to extract text from PDF

def extract\_text\_from\_pdf(pdf\_path):

if not os.path.exists(pdf\_path):

st.error(f"Error: PDF file '{pdf\_path}' not found.")

return ""

sections = []

try:

with pdfplumber.open(pdf\_path) as pdf:

for page in pdf.pages:

text = page.extract\_text()

if text:

sections.append(text)

return "\n".join(sections)

except Exception as e:

st.error(f"Error reading PDF: {e}")

return ""

# Function to clean text

def clean\_text(text):

text = re.sub(r'Page\s\*\d+\s\*of\s\*\d+', '', text)

text = re.sub(r'\bPage\s\*\d+\b', '', text)

text = re.sub(r'\b\d+\s\*/\s\*\d+\b', '', text)

return text.strip()

# Function to split text into structured sections

def split\_sections\_by\_number(text):

section\_pattern = r"(\d+[A-Z]?(?:\(\d+\))?)\.\s\*(.\*?)(?=\n\d+[A-Z]?(?:\(\d+\))?\.)"

sections = re.findall(section\_pattern, text, re.DOTALL)

return [{"section\_id": s[0].strip(), "content": s[1].strip()} for s in sections]

# Function to create FAISS index

@st.cache\_data

def create\_faiss\_index(structured\_data):

if not structured\_data:

st.error("Error: No structured data found for indexing.")

return None, None, None

try:

retriever = SentenceTransformer('all-MiniLM-L6-v2')

corpus = [section['content'] for section in structured\_data if section.get('content')]

corpus\_embeddings = retriever.encode(corpus)

dimension = corpus\_embeddings.shape[1]

index = faiss.IndexFlatL2(dimension)

index.add(np.array(corpus\_embeddings))

return retriever, index, structured\_data

except Exception as e:

st.error(f"Error creating FAISS index: {e}")

return None, None, None

# Load structured data

@st.cache\_data

def load\_data():

return {

name: split\_sections\_by\_number(clean\_text(extract\_text\_from\_pdf(path)))

for name, path in BOOKS.items()

}

# Function to get exact section

def get\_exact\_section(section\_number, structured\_data):

for section in structured\_data:

if section["section\_id"].strip() == section\_number.strip():

return section

return None

# Function to find relevant sections using FAISS

def find\_relevant\_section(query, retriever, index, structured\_data, top\_k=3):

if not retriever or not index:

return []

query\_embedding = retriever.encode([query])

distances, indices = index.search(np.array(query\_embedding), top\_k)

return [structured\_data[i] for i in indices[0] if i < len(structured\_data)]

# AI Response Generator

def generate\_response\_with\_groq(prompt, section\_number, book\_name, context):

structured\_prompt = f"According to Section {section\_number} of {book\_name}, {prompt}"

try:

response = client.chat.completions.create(

model="llama3-8b-8192",

messages=[

{"role": "system", "content": "You are a legal assistant providing detailed and comprehensive legal explanations based on Pakistani law. Always provide at least 5-6 sentences per response."},

{"role": "user", "content": f"{structured\_prompt}\n\nContext: {context}"}

],

max\_tokens=5000

)

return response.choices[0].message.content.strip()

except Exception as e:

return f"Error generating response: {e}"

# Load data

legal\_data = load\_data()

faiss\_indices = {}

for law, structured\_data in legal\_data.items():

if structured\_data:

retriever, index, \_ = create\_faiss\_index(structured\_data)

faiss\_indices[law] = (retriever, index, structured\_data)

# Streamlit UI

st.title("Pakistan Legal Code Chatbot 📜🤖")

st.write("Ask a question about Pakistan Penal Code, Code of Criminal Procedure, Code of Civil Procedure, Constitution of Pakistan, or Rules of Business.")

query = st.text\_input("Enter your legal question:")

if st.button("Submit"):

if query.strip():

book\_name, structured\_data, retriever, index = None, None, None, None

for law in legal\_data.keys():

if law.lower() in query.lower():

retriever, index, structured\_data = faiss\_indices[law]

book\_name = law

break

match = re.search(r"(?:section\s\*no\s\*|section|rule|article)\s\*(\d+[A-Z]?(?:\(\d+\))?)", query, re.IGNORECASE)

if match and structured\_data:

section\_number = match.group(1)

exact\_section = get\_exact\_section(section\_number, structured\_data)

if exact\_section:

response = generate\_response\_with\_groq(query, exact\_section['section\_id'], book\_name, exact\_section['content'])

st.write("### AI-Generated Explanation:")

st.write(response)

else:

relevant\_sections = find\_relevant\_section(query, retriever, index, structured\_data)

if relevant\_sections:

response = generate\_response\_with\_groq(query, relevant\_sections[0]['section\_id'], book\_name, relevant\_sections[0]['content'])

st.write("### AI-Generated Explanation:")

st.write(response)

else:

st.warning("No relevant section found.")