고급 기능이 탑재된 Streamlit 헌법 Q&A 챗봇 구현

202000826 김연범

1. 고급 기능의 종류

a) 파일 업로드 기능

a-1) st.file_uploader() 컴포넌트는 Streamlit 상에서 사용자가 로컬 디스크의 파일을 업로드할 수 있도록 해줌.

a-2) 우리의 목표는 업로드된 파일을 기반으로 RAG 체인을 구축하고, 이를 통해 RAG 챗봇을 사용할 수 있도록 하는 것임.

b) 메모리 기능 및 참고 문서 표시 기능

b-1) 메모리 기능을 통해 AI의 대답에 대한 추가 질문이나 연관 질문 기능 구현 및 참고 문서 표시 기능을 통해 환각 현상 점검책 구현

b-2) 우리의 목표는 메모리 기능 및 참고 문서 표시 기능이 추가된 Streamlit 헌법 Q&A 챗봇을 구현하는 것임.

2. 세부사항

a) 파일 업로드 기능

a-1) 업로드된 PDF 문서 처리 함수 : Streamlit은 업로드된 파일의 경로를 저장하는 것이 아니라 파일 자체를 저장하기 때문에 임시파일을 생성해 PDF내용을 기록하고, 이 파일의 경로를 변수에 저장하는 방식으로 처리해야 함.

a-2) Streamlit UI 수정 : 업르드된 파일의 존재 여부에 따라 조건절을 설정해줌.

b) 메모리 기능

b-1) chat history를 고려하도록 시스템 프롬프트 수정 및 프롬프트 사이에 MessagePlaceholder(“history”)를 삽입해 메모리 기능 구현

b-2) history_aware_retriever를 통해 채팅 히스토리와 사용자의 마지막 질문을 바탕으로 하나의 독립된 질문 생성 -> 유사 문서 검색 기능 구현

b-3) 속도 개선을 위한 LLM 모델 선택 기능을 GUI에 추가

c) 참고 문서 표시 기능

c-1) st.expander() 컴포넌트를 활용해 드롭다운 형태로 참고 문서 표시 기능 구현

3. 코드 및 실행 결과

a) 파일 업로드 기능

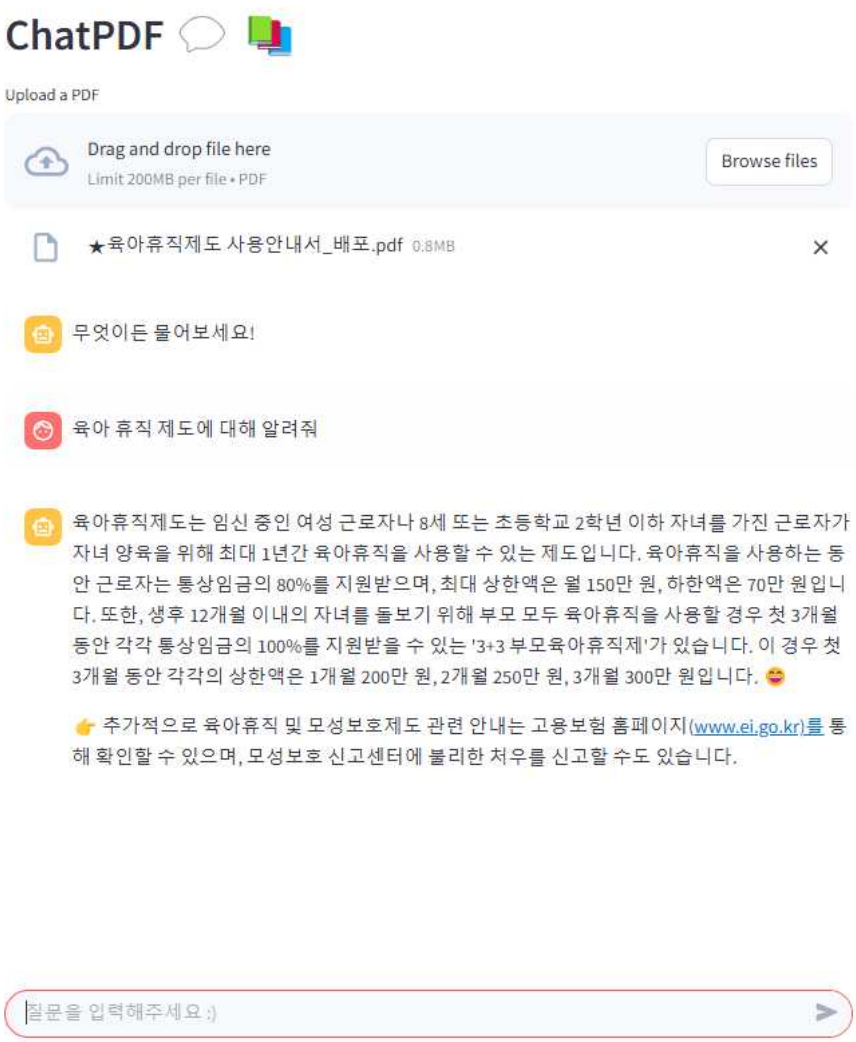
a-1) 코드

```python
import os
import streamlit as st
import tempfile
from langchain.document_loaders import PyPDFLoader
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain_openai import OpenAIEmbeddings, ChatOpenAI
from langchain_chroma import Chroma
from langchain_core.prompts import ChatPromptTemplate
from langchain_core.runnables import RunnablePassthrough
from langchain_core.output_parsers import StrOutputParser

#Chroma tenant 오류 방지 위한 코드
import chromadb
chromadb.api.client.SharedSystemClient.clear_system_cache()

#오픈AI API 키 설정
os.environ["OPENAI_API_KEY"] = "sk-proj-950hCG9R3OttgJ5609m7AAN_Le2ij20LHmv-Doh2cU49Al5zKO6xrsaBGA-HtBt2jmlbjxUapbT3BlbkFJG1wVXnpdID5eAcdI

#cache_resource로 한번 실행한 결과 캐싱해두기
@st.cache_resource
def load_pdf(_file):
    # 임시 파일을 생성해 업로드된 PDF 파일의 데이터 저장
    with tempfile.NamedTemporaryFile(mode="wb", delete=False) as tmp_file:
        tmp_file.write(_file.getvalue())
        tmp_file_path = tmp_file.name
        #PDF 파일 업로드
        loader = PyPDFLoader(file_path=tmp_file_path)
        pages = loader.load_and_split()
    return pages

#텍스트 청크들을 Chroma 안에 임베딩 벡터로 저장
@st.cache_resource
def create_vector_store(_docs):
    text_splitter = RecursiveCharacterTextSplitter(chunk_size=1000, chunk_overlap=0)
    split_docs = text_splitter.split_documents(_docs)
    vectorstore = Chroma.from_documents(split_docs, OpenAIEmbeddings(model='text-embedding-3-small'), persist_directory="./upload_data")
    return vectorstore

#검색된 문서를 하나의 텍스트로 합치는 헬퍼 함수
def format_docs(docs):
        return "\n\n".join(doc.page_content for doc in docs)
```

```python
#PDF 문서 기반 RAG 체인 구축
@st.cache_resource
def chaining(_pages):
    vectorstore = create_vector_store(_pages)
    retriever = vectorstore.as_retriever()

    qa_system_prompt = """
    You are an assistant for question-answering tasks. \
    Use the following pieces of retrieved context to answer the question. \
    If you don't know the answer, just say that you don't know. \
    Keep the answer perfect. please use imogi with the answer.
    Please answer in Korean and use respectful language.\
    {context}
    """

    qa_prompt = ChatPromptTemplate.from_messages(
        [
            ("system", qa_system_prompt),
            ("human", "{input}"),
        ]
    )

    llm = ChatOpenAI(model="gpt-4o")
    rag_chain = (
        {"context": retriever | format_docs, "input": RunnablePassthrough()}
        | qa_prompt
        | llm
        | StrOutputParser()
    )
    return rag_chain
```

```python
# Streamlit UI
st.header("ChatPDF 💬 📚")
# Streamlit의 파일 업로드 기능
uploaded_file = st.file_uploader("Upload a PDF", type=["pdf"])
if uploaded_file is not None:
    pages = load_pdf(uploaded_file)

    rag_chain = chaining(pages)

    if "messages" not in st.session_state:
        st.session_state["messages"] = [{"role": "assistant", "content": "무엇이든 물어보세요!"}]

    for msg in st.session_state.messages:
        st.chat_message(msg['role']).write(msg['content'])

    if prompt_message := st.chat_input("질문을 입력해주세요 :)"):
        st.chat_message("human").write(prompt_message)
        st.session_state.messages.append({"role": "user", "content": prompt_message})
        with st.chat_message("ai"):
            with st.spinner("Thinking..."):
                response = rag_chain.invoke(prompt_message)
                st.session_state.messages.append({"role": "assistant", "content": response})
                st.write(response)
```

a-2) 실행 결과


ChatPDF
Upload a PDF
Drag and drop file here
Limit 200MB per file • PDF
Browse files
★육아휴직제도 사용안내서_배포.pdf 0.8MB
무엇이든 물어보세요!
육아 휴직 제도에 대해 알려줘
육아휴직제도는 임신 중인 여성 근로자나 8세 또는 초등학교 2학년 이하 자녀를 가진 근로자가 자녀 양육을 위해 최대 1년간 육아휴직을 사용할 수 있는 제도입니다. 육아휴직을 사용하는 동안 근로자는 통상임금의 80%를 지원받으며, 최대 상한액은 월 150만 원, 하한액은 70만 원입니다. 또한, 생후 12개월 이내의 자녀를 돌보기 위해 부모 모두 육아휴직을 사용할 경우 첫 3개월 동안 각각 통상임금의 100%를 지원받을 수 있는 '3+3 부모육아휴직제'가 있습니다. 이 경우 첫 3개월 동안 각각의 상한액은 1개월 200만 원, 2개월 250만 원, 3개월 300만 원입니다.
추가적으로 육아휴직 및 모성보호제도 관련 안내는 고용보험 홈페이지(www.ei.go.kr)를 통해 확인할 수 있으며, 모성보호 신고센터에 불리한 처우를 신고할 수도 있습니다.
질문을 입력해주세요 :)

b) 메모리 기능 및 참고 문서 표현 기능

b-1) 코드

```python
import os
import streamlit as st

from langchain.document_loaders import PyPDFLoader
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain_openai import OpenAIEmbeddings, ChatOpenAI
from langchain.vectorstores import Chroma
from langchain.prompts import ChatPromptTemplate, MessagesPlaceholder
from langchain.chains.combine_documents import create_stuff_documents_chain
from langchain.chains import create_history_aware_retriever, create_retrieval_chain
from langchain_core.runnables.history import RunnableWithMessageHistory
from langchain_community.chat_message_histories.streamlit import StreamlitChatMessageHistory

#오픈AI API 키 설정
os.environ["OPENAI_API_KEY"] = "sk-proj-950hCG9R3OttgJ5609m7AAN_Le2ij20LHmv-Doh2cU49Al5zKO6xrsa

#cache_resource로 한번 실행한 결과 캐싱해두기
@st.cache_resource
def load_and_split_pdf(file_path):
    loader = PyPDFLoader(file_path)
    return loader.load_and_split()

#텍스트 청크들을 Chroma 안에 임베딩 벡터로 저장
@st.cache_resource
def create_vector_store(_docs):
    text_splitter = RecursiveCharacterTextSplitter(chunk_size=1000, chunk_overlap=0)
    split_docs = text_splitter.split_documents(_docs)
    persist_directory = "./chroma_db"
    vectorstore = Chroma.from_documents(
        split_docs,
        OpenAIEmbeddings(model='text-embedding-3-small'),
        persist_directory=persist_directory
    )
    return vectorstore

#만약 기존에 저장해둔 ChromaDB가 있는 경우, 이를 로드
@st.cache_resource
def get_vectorstore(_docs):
    persist_directory = "./chroma_db"
    if os.path.exists(persist_directory):
        return Chroma(
            persist_directory=persist_directory,
            embedding_function=OpenAIEmbeddings(model='text-embedding-3-small')
        )
    else:
        return create_vector_store(_docs)
```

```python
# PDF 문서 로드-벡터 DB 저장-검색기-히스토리 모두 합친 Chain 구축
@st.cache_resource
def initialize_components(selected_model):
    file_path = r"헌법.pdf"
    pages = load_and_split_pdf(file_path)
    vectorstore = get_vectorstore(pages)
    retriever = vectorstore.as_retriever()

    # 채팅 히스토리 요약 시스템 프롬프트
    contextualize_q_system_prompt = """Given a chat history and the latest user question \
    which might reference context in the chat history, formulate a standalone question \
    which can be understood without the chat history. Do NOT answer the question, \
    just reformulate it if needed and otherwise return it as is."""
    contextualize_q_prompt = ChatPromptTemplate.from_messages(
        [
            ("system", contextualize_q_system_prompt),
            MessagesPlaceholder("history"),
            ("human", "{input}"),
        ]
    )

    # 질문-답변 시스템 프롬프트
    qa_system_prompt = """You are an assistant for question-answering tasks. \
    Use the following pieces of retrieved context to answer the question. \
    If you don't know the answer, just say that you don't know. \
    Keep the answer perfect. please use imogi with the answer.
    대답은 한국어로 하고, 존댓말을 써줘.\

    {context}"""
    qa_prompt = ChatPromptTemplate.from_messages(
        [
            ("system", qa_system_prompt),
            MessagesPlaceholder("history"),
            ("human", "{input}"),
        ]
    )

    llm = ChatOpenAI(model=selected_model)
    history_aware_retriever = create_history_aware_retriever(llm, retriever, contextualize_q_prompt)
    question_answer_chain = create_stuff_documents_chain(llm, qa_prompt)
    rag_chain = create_retrieval_chain(history_aware_retriever, question_answer_chain)
    return rag_chain

# Streamlit UI
st.header("헌법 Q&A 챗봇 💬 📚")
# LLM 모델 선택
option = st.selectbox("Select GPT Model", ("gpt-4o-mini", "gpt-3.5-turbo-0125"))
rag_chain = initialize_components(option)
chat_history = StreamlitChatMessageHistory(key="chat_messages")

conversational_rag_chain = RunnableWithMessageHistory(
    rag_chain,
    lambda session_id: chat_history,
    input_messages_key="input",
    history_messages_key="history",
    output_messages_key="answer",
)


if "messages" not in st.session_state:
    st.session_state["messages"] = [{"role": "assistant",
                                     "content": "헌법에 대해 무엇이든 물어보세요!"}]

for msg in chat_history.messages:
    st.chat_message(msg.type).write(msg.content)


if prompt_message := st.chat_input("Your question"):
    st.chat_message("human").write(prompt_message)
    with st.chat_message("ai"):
        with st.spinner("Thinking..."):
            config = {"configurable": {"session_id": "any"}}
            response = conversational_rag_chain.invoke(
                {"input": prompt_message},
                config)

            answer = response['answer']
            st.write(answer)
            with st.expander("참고 문서 확인"):
                for doc in response['context']:
                    st.markdown(doc.metadata['source'], help=doc.page_content)
```

b-2) 실행 결과


헌법 Q&A 챗봇
Select GPT Model
gpt-4o-mini
안녕하세요
안녕하세요! 무엇을 도와드릴까요?
대통령의 임기는?
대통령의 임기는 5년으로 하며, 중임할 수 없습니다.
국회의원은?
국회의원의 임기는 4년입니다.
참고 문서 확인
헌법.pdf
헌법.pdf
헌법.pdf
헌법.pdf
Your question