**TA's 솔루션 해설**

* <https://gist.github.com/ICE0208/8d296c5aabecc9a62fc4c9610d8dc493>

1. ChatOpenAI 모델 초기화 및 설정
   * ChatOpenAI 생성자를 이용하여 OpenAI의 GPT 모델을 초기화합니다.
   * 모델 초기화 시 몇몇 설정을 커스텀할 수 있습니다.
   * 해설 코드에서는 temperature를 0.1로 설정하였습니다.
2. 문서 로드와 쪼개기 / 임베딩 생성 및 캐시
   * TextLoader를 이용하여 텍스트 문서를 로드합니다.
   * CharacterTextSplitter를 사용하여 텍스트를 일정한 크기의 조각으로 나눕니다.
   * OpenAIEmbeddings를 사용하여 텍스트 조각들에 대한 임베딩을 생성합니다.
   * CacheBackedEmbeddings를 이용하여 임베딩을 캐시하여 성능을 향상시킵니다.
   * FAISS를 이용하여 임베딩 벡터들을 저장하고, 효율적으로 검색할 수 있도록 합니다.
3. 메모리 초기화
   * ConversationBufferMemory를 사용하여 대화 기록을 저장하고 관리합니다.
   * return\_messages=True로 설정하여 메모리를 채팅 형식으로 로드할 수 있도록 합니다.
4. 메모리 로드 함수 정의
   * 메모리를 로드하여 대화 기록을 반환하는 함수를 정의합니다.
   * 이 함수는 체인에서 대화 기록을 로드하는 데 사용됩니다.
5. 검색기 설정
   * vectorstore.as\_retriever()를 사용하여 벡터 저장소를 검색기로 설정합니다.
6. 프롬프트 생성
   * ChatPromptTemplate.from\_messages를 이용하여 대화 메시지 템플릿을 생성합니다.
   * 프롬프트는 시스템 메시지, 대화 기록 플레이스홀더, 사용자 질문으로 구성됩니다.
   * 시스템 메시지는 모델에게 주어진 컨텍스트 내에서만 답변하도록 지시합니다.
7. 체인 설정 및 체인 호출 함수 정의
   * RunnablePassthrough와 메모리 로드 함수, 검색기(retriever)를 사용하여 체인을 구성합니다.
   * 체인은 컨텍스트를 검색하고, 사용자 질문과 대화 기록을 결합하여 최종 프롬프트를 생성한 후, LLM을 호출합니다.
   * invoke\_chain 함수를 정의하여 이 함수를 실행하였을 때, 체인을 실행하고 그 결과를 바로 메모리에 저장하도록 합니다.
8. 체인 테스트
   * invoke\_chain 함수를 통해 예시 질문을 입력받아 체인을 실행합니다.
   * 예시 질문에 대한 답변이 정상적으로 반환되는지 확인합니다.
9. 메모리 확인
   * load\_memory 함수를 호출하여 메모리에 저장된 대화 기록이 체인에 적용되었는지 확인합니다.
10. 결론

* 이번 챌린지는 Stuff Documents 체인을 이용하여 완전한 RAG 파이프라인을 수동으로 구현하는 연습을 해보는 것이었습니다. 또한, 체인에 ConversationBufferMemory를 추가하여 대화 기록을 관리하는 방법을 복습했습니다. 이를 통해, 이전에 배운 메모리 관리 기법과 새롭게 배운 RAG 파이프라인의 수동 구현을 결합함으로써 학습 내용을 더욱 잘 이해하고 응용할 수 있었습니다.

**from** langchain.chat\_models **import** ChatOpenAI

**from** langchain.document\_loaders **import** TextLoader

**from** langchain.text\_splitter **import** CharacterTextSplitter

**from** langchain.embeddings **import** OpenAIEmbeddings, CacheBackedEmbeddings

**from** langchain.vectorstores **import** FAISS

**from** langchain.storage **import** LocalFileStore

**from** langchain.schema.runnable **import** RunnablePassthrough

**from** langchain.prompts **import** ChatPromptTemplate, MessagesPlaceholder

**from** langchain.memory **import** ConversationBufferMemory

llm **=** ChatOpenAI(

temperature**=**0.1,

)

**문서 로드와 쪼개기 / 임베딩 생성 및 캐시**

In [8]:

cache\_dir **=** LocalFileStore("./.cache/")

splitter **=** CharacterTextSplitter**.**from\_tiktoken\_encoder(

separator**=**"\n",

chunk\_size**=**600,

chunk\_overlap**=**100,

)

loader **=** TextLoader("./#13-file.txt")

docs **=** loader**.**load\_and\_split(text\_splitter**=**splitter)

embeddings **=** OpenAIEmbeddings()

cached\_embeddings **=** CacheBackedEmbeddings**.**from\_bytes\_store(

embeddings,

cache\_dir,

)

vectorstore **=** FAISS**.**from\_documents(docs, cached\_embeddings)

**메모리와 문서를 이용한 프롬프트**

In [9]:

memory **=** ConversationBufferMemory(

llm**=**llm,

max\_token\_limit**=**20,

return\_messages**=True**,

)

**def** load\_memory(\_):

**return** memory**.**load\_memory\_variables({})["history"]

retriever **=** vectorstore**.**as\_retriever()

prompt **=** ChatPromptTemplate**.**from\_messages(

[

(

"system",

"You are a helpful assistant. Answer questions using only the following context. If you don't know the answer just say you don't know, don't make it up:\n\n{context}",

),

MessagesPlaceholder(variable\_name**=**"history"),

("human", "{question}"),

]

)

**체인 연결 / 체인 호출 함수 정의**

In [10]:

chain **=** (

{

"context": retriever,

"question": RunnablePassthrough(),

"history": load\_memory,

}

**|** prompt

**|** llm

)

**def** invoke\_chain(question):

result **=** chain**.**invoke(question)

print(result)

memory**.**save\_context({"input": question}, {"output": result**.**content})

**체인에 질문하여 테스트**

In [11]:

invoke\_chain("Is Aaronson guilty?")

invoke\_chain("What message did he write in the table?")

invoke\_chain("Who is Julia?")

content='According to the document, Jones, Aaronson, and Rutherford were guilty of the crimes they were charged with.'

content='He wrote "FREEDOM IS SLAVERY" and "TWO AND TWO MAKE FIVE" on the table.'

content='Julia is a character who was involved with Winston in the novel.'

**메모리를 출력하여 메모리가 체인에 적용되었는지 확인**

In [12]:

load\_memory({})

Out[12]:

[HumanMessage(content='Is Aaronson guilty?'),

AIMessage(content='According to the document, Jones, Aaronson, and Rutherford were guilty of the crimes they were charged with.'),

HumanMessage(content='What message did he write in the table?'),

AIMessage(content='He wrote "FREEDOM IS SLAVERY" and "TWO AND TWO MAKE FIVE" on the table.'),

HumanMessage(content='Who is Julia?'),

AIMessage(content='Julia is a character who was involved with Winston in the novel.')]

**Review Quiz!**

* 오늘의 강의: [풀스택 GPT: From #2.0 to #6.10](https://nomadcoders.co/fullstack-gpt/lectures/4542)
* 오늘의 과제: 위의 강의를 시청하신 후, 아래 퀴즈를 풀면 됩니다.
* 제출기간: 익일 오전 6시까지
* 퀴즈는 단 1회만 응할 수 있습니다.

Your score is: 9/10

**Question 1 of 10**

Correct

Why are Prompt Templates useful?

They help us reuse prompts and format them with variables

**Question 2 of 10**

Correct

What is the difference between a PromptTemplate and a ChatPromptTemplate?

ChatPromptTemplate is to format chat messages and PromptTemplate is to format a string

**Question 3 of 10**

Correct

What does LCEL mean?

LangChain Expression Language

**Question 4 of 10**

Incorrect

Why do we use few-shot examples?

To give examples to LangChain on how to format the response

**Your answer**

To give examples to the LLM on how to format the response

**Correct Answer**

**Question 5 of 10**

Correct

What does the | operator do here?

prompt | model | output\_parser

It formats the prompt, sends it to the model and gives the output to the output parser

**Question 6 of 10**

Correct

What does RAG mean?

Retrieval Augmented Generation

**Question 7 of 10**

Correct

When do we need to do RAG?

When we want to use an LLM with data it was not trained with

**Question 8 of 10**

Correct

When doing RAG why do we need to split documents?

To embed the document in small chunks

**Question 9 of 10**

Correct

Why do we need to embed our documents?

To turn words into vectors and perform searches later

**Question 10 of 10**

Correct

Why do we use Vector Stores?

To store and search embeddings

------------------------- Assignment #7

**TA's 솔루션 해설**

* <https://huq8hcz9ktqsatygmrjsdm.streamlit.app/>
* <https://github.com/fullstack-gpt-python/assignment-15/blob/main/app.py>

1. 사용자의 OpenAI API 키 이용하기
   * st.text\_input을 사용하여 사용자의 OpenAI API 키를 입력받습니다.
   * 입력받은 API 키를 ChatOpenAI와 OpenAIEmbeddings 클래스를 사용할 때 openai\_api\_key 매개변수로 넘깁니다.
2. 파일 업로드
   * st.file\_uploader를 사용하여 사용자가 파일을 업로드할 수 있도록 합니다.
   * 업로드할 수 있는 파일의 확장자는 pdf, txt, docx로 지정합니다.
   * 업로드된 파일을 임베딩하고 vectorstore에 저장한 후, 이를 retriever로 변환하여 체인에서 사용합니다.
   * 이전과 같은 파일을 선택했을 때 임베딩 과정을 다시 하지 않도록 하기 위해 embed\_file 함수에 st.cache\_data 데코레이터를 추가하였습니다. ([st.cache\_data 공식 문서](https://docs.streamlit.io/develop/api-reference/caching-and-state/st.cache_data" \t "_blank))
3. 채팅 기록
   * 채팅 기록을 저장하기 위해 Session State를 사용합니다.
   * 솔루션에서는 st.session\_state["messages"]를 리스트로 초기화하고, 메시지를 추가하는 방법으로 구현했습니다. (save\_message 함수 참고)
   * 저장된 채팅 기록을 페이지에 출력하기 위해 st.session\_state["messages"] 리스트에 있는 메시지들을 하나씩 출력합니다. (paint\_history 함수 참고)
4. 결론

* 이전 과제에서 구현한 RAG 파이프라인을 Streamlit을 활용하여 재구현하면서, Streamlit 사용에 익숙해지고 지난 과제 내용을 효과적으로 복습할 수 있었습니다.

import streamlit as st

from langchain.document\_loaders import UnstructuredFileLoader

from langchain.embeddings import CacheBackedEmbeddings, OpenAIEmbeddings

from langchain.storage import LocalFileStore

from langchain.text\_splitter import CharacterTextSplitter

from langchain.vectorstores.faiss import FAISS

from langchain.chat\_models import ChatOpenAI

from langchain.prompts import ChatPromptTemplate

from langchain.schema.runnable import RunnableLambda, RunnablePassthrough

from langchain.callbacks.base import BaseCallbackHandler

from pathlib import Path

st.set\_page\_config(

page\_title="Assignment #15",

page\_icon="📜",

)

class ChatCallbackHandler(BaseCallbackHandler):

message = ""

def on\_llm\_start(self, \*args, \*\*kwargs):

self.message\_box = st.empty()

def on\_llm\_end(self, \*args, \*\*kwargs):

save\_message(self.message, "ai")

def on\_llm\_new\_token(self, token, \*args, \*\*kwargs):

self.message += token

self.message\_box.markdown(self.message)

if "messages" not in st.session\_state:

st.session\_state["messages"] = []

@st.cache\_data(show\_spinner="Embedding file...")

def embed\_file(file):

file\_content = file.read()

file\_path = f"./.cache/files/{file.name}"

Path("./.cache/files").mkdir(parents=True, exist\_ok=True)

with open(file\_path, "wb+") as f:

f.write(file\_content)

cache\_dir = LocalFileStore(f"./.cache/embeddings/{file.name}")

splitter = CharacterTextSplitter.from\_tiktoken\_encoder(

separator="\n",

chunk\_size=600,

chunk\_overlap=100,

)

loader = UnstructuredFileLoader(f"{file\_path}")

docs = loader.load\_and\_split(text\_splitter=splitter)

embeddings = OpenAIEmbeddings(

openai\_api\_key=openai\_api\_key,

)

cached\_embeddings = CacheBackedEmbeddings.from\_bytes\_store(embeddings, cache\_dir)

vectorstore = FAISS.from\_documents(docs, cached\_embeddings)

retriever = vectorstore.as\_retriever()

return retriever

def save\_message(message, role):

st.session\_state["messages"].append({"message": message, "role": role})

def send\_message(message, role, save=True):

with st.chat\_message(role):

st.markdown(message)

if save:

save\_message(message, role)

def paint\_history():

for message in st.session\_state["messages"]:

send\_message(message["message"], message["role"], save=False)

def format\_docs(docs):

return "\n\n".join(document.page\_content for document in docs)

prompt = ChatPromptTemplate.from\_messages(

[

(

"system",

"""

Answer the question using ONLY the following context. If you don't know the answer just say you don't know. DON't make anything up

Context: {context}

""",

),

("human", "{question}"),

]

)

def main():

if not openai\_api\_key:

return

llm = ChatOpenAI(

temperature=0.1,

streaming=True,

openai\_api\_key=openai\_api\_key,

callbacks=[

ChatCallbackHandler(),

],

)

if file:

retriever = embed\_file(file)

send\_message("I'm ready! Ask away!", "ai", save=False)

paint\_history()

message = st.chat\_input("Ask anything about your file.....")

if message:

send\_message(message, "human")

chain = (

{

"context": retriever | RunnableLambda(format\_docs),

"question": RunnablePassthrough(),

}

| prompt

| llm

)

with st.chat\_message("ai"):

chain.invoke(message)

else:

st.session\_state["messages"] = []

return

st.title("Document GPT")

st.markdown(

"""

Welcome!

Use this chatbot to ask questions to an AI about your files!

1. Input your OpenAI API Key on the sidebar

2. Upload your file on the sidebar.

3. Ask questions related to the document.

"""

)

with st.sidebar:

# API Key 입력

openai\_api\_key = st.text\_input("Input your OpenAI API Key")

# 파일 선택

file = st.file\_uploader(

"Upload a. txt .pdf or .docx file",

type=["pdf", "txt", "docx"],

)

try:

main()

except Exception as e:

st.error("Check your OpenAI API Key or File")

st.write(e)