D:\myprojects\nlp\rag\rag_wiki.py

```python
# conda create -n rag transformers
# conda activate rag
# conda install pytorch torchvision torchaudio pytorch-cuda=11.8 -c pytorch -c nvidia
# conda install -c conda-forge faiss-gpu
# pip install langchain pdfplumber datasets
# pip install wikipedia sentence-transformers tiktoken
# pip install --upgrade huggingface_hub
# pip install --upgrade openai

import numpy as np

from transformers import AutoTokenizer
from langchain_community.document_loaders import WikipediaLoader
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain_community.embeddings import HuggingFaceEmbeddings
from langchain_community.vectorstores import FAISS
from langchain_community.vectorstores.utils import DistanceStrategy
from sentence_transformers import CrossEncoder

import os, openai
from openai import OpenAI

SHOW_INSIDE = True
# [1] harrypotter 키워드에 대한 wikipedia page 3개 읽어오기
pages = WikipediaLoader(query="Harry Potter", load_max_docs=3, lang="en").load()

# [2.1] text splitter 정의
text_splitter = RecursiveCharacterTextSplitter.from_huggingface_tokenizer(
    tokenizer=AutoTokenizer.from_pretrained("sentence-transformers/all-MiniLM-L12-v2"),
    chunk_size=256,     # 256 chars / chunk
    chunk_overlap=16,   # 이웃하는 chunk사이 16 chars overlap
    strip_whitespace=True,  # chunk 생성시에 공백 제거
)
# [2.2]불러온 페이지에 대하여 text_splitting하여 chunks 구하기
chunks = text_splitter.split_documents(pages)

if SHOW_INSIDE :
    print ('\nsplit된 chunk의 갯수: ',len(chunks))
    print('\nchunk[0]의 전체내용: \n',chunks[0])
    print('\nchunk[0]의 page_content: \n',chunks[0].page_content)
    print('\nchunk[0]의 metadata: \n)',chunks[0].metadata)
    print('\nchunk[0]의 metadata title: \n',chunks[0].metadata['title'])
    print('\nchunk[0]의 metadta summary: \n',chunks[0].metadata['summary'])
    print('\nchunk[0]의 metadata source: \n',chunks[0].metadata['source'])

# [3] bidirectional encpoder 모델을 이용한 Embedding 정의
bi_encoder = HuggingFaceEmbeddings(
    #model_name="sentence-transformers/all-MiniLM-L12-v2", model_kwargs={"device": "cpu"}
    model_name="sentence-transformers/all-MiniLM-L12-v2", model_kwargs={"device": "cuda"}
)

# question과 첫번째 chunk의 내용 embedding 해보기
'''
embedding1 = bi_encoder.embed_query("Harry Potter")
embedding2 = bi_encoder.embed_query(chunks[0].page_content)
'''
```

```python
# [4] chunks들로부터 bi_encoder를 이용한 임베딩을 사용하고, dot-product거리를 이용하여 faiss index 생성.
# FAISS (Facebook AI Similarity Search)
faiss_db = FAISS.from_documents(
    chunks, bi_encoder, distance_strategy=DistanceStrategy.DOT_PRODUCT
)

# [5] cross encoder 정의
cross_encoder = CrossEncoder(
    "cross-encoder/ms-marco-TinyBERT-L-2-v2", max_length=512, device="cpu"
)
'''
cross_encoder.rank(
    query="Do I like cats?",
    documents=["I like cats", "I like horses", "I like dogs"],
    return_documents=True,
)
'''
client = OpenAI(
    #api_key=os.environ.get("OPENAI_API_KEY"),
    api_key = "sk-proj-P******************kFJFytKn0x7vYAeqXrL18AR"
)
#================================================================================================
default_question = "On what date was the first book in the Harry Potter series originally published?"

while True :
    question = input("\nType 'q' to quit: ")

    if question == 'q' : break
    elif question == '' :
        question = default_question

    # [5] question과 유사도가 가장 가까운 상위 25의 chunk를 추출
    retrieved_chunks = faiss_db.similarity_search(question, k=25)

    reranked_chunks = cross_encoder.rank(
        (question),
        [doc.page_content for doc in retrieved_chunks],
        top_k=3,
        return_documents=True,
    )

    context = "".join(doc["text"] + "\n----------\n" for doc in reranked_chunks)

    print("\n\n***최종 상위 답변 3개 통합문장 : \n",context)

    if SHOW_INSIDE :
        print(f"\n\n*** RAG에 의한 연관된 상위 {len(reranked_chunks)} 개 chunks의 제목과 출처(URL)")
        for doc in reranked_chunks:
            corpus_id = doc["corpus_id"]
            print(
                f"Chunk {corpus_id} from '{retrieved_chunks[corpus_id].metadata['title']}' \n\t(URL: {retrieved_chunks[corpus_id].metadata['source']})"
            )

    #------------------------------------------------------------------------------------------
    # Use OpenAI's LLM to generate a response based on the context
```

```
112
113     prompt=f"You are a helpful assistant. Based on the following details about Harry
    Potter, {context} Answer the question.",
114     try :
115         response = client.chat.completions.create(
116             messages=[
117                     {
118                         "role": "system", "content": prompt,
119                         "role": "user", "content": question,
120                     }
121             ],
122             model="gpt-3.5-turbo-instruct",  # or  "text-davinci-002"  "gpt-3.5-turbo-
    instruct"
123             max_tokens=150,
124             temperature=0.5,
125             stop=["\n"]
126             )
127         print("\nOpenAI LLM의 답변:\n", response.choices[0].text.strip())
128     except Exception as e:
129         print("Error : ",e)
```