- 출처: LangChain 공식 문서 또는 해당 교재명
- 원본 URL: https://smith.langchain.com/hub/teddynote/summary-stuff-documents

---

6. `MultiQueryRetriever`

1) 다중 쿼리 검색기

- **벡터 검색의 문제점**
  - **벡터 기반 검색**: 질문(쿼리) - 문서의 내용을 숫자로 표현한 것(임베딩) 사이의 **거리** 측정 → 유사한 문서를 찾음
  - **문제점**: 아래의 경우 **검색 결과가 달라지거나 관련성이 떨어질 수 있음**
    - ① 질문의 아주 미세한 차이
    - ② 임베딩이 내용의 의미를 완벽히 포착하지 못할 경우
  - **해결의 어려움**: 검색 결과를 개선하기 위해 질문을 수동으로 조정하는 작업(프롬프트 엔지니어링/튜닝)은 매우 번거로움

- **`MultiQueryRetriever`**: 다양한 관점의 자동화된 검색
  - → 위의 문제를 해결하고 검색 결과를 풍부하게 만들기 위해 고안됨
  - **아이디어**: **LLM** (대규모 언어 모델)을 사용 → 하나의 사용자 질문에서 **다양한 관점을 가진 여러 개의 새로운 쿼리를 자동으로 생성***
  - **작동 방식**:
    - ① LLM이 사용자 질문을 바탕으로 여러 개의 쿼리를 생성
    - ② 각각의 쿼리로 따로 검색을 수행해 관련 문서를 찾음
    - ③ 이 모든 검색 결과에서 중복되지 않는 고유한 문서들만 모아 최종 결과로 반환함 *(합집합)*
  - **효과**
    - **다양한 쿼리를 사용** → 벡터 검색이 놓칠 수 있는 잠재적으로 **관련성 높은 문서를 더 많이 포착하여 더 풍부하고 포괄적인 검색 결과를 얻게 됨**

---

2) 설정

```
# API 키를 환경변수로 관리하기 위한 설정 파일
from dotenv import load_dotenv

# API 키 정보 로드
load_dotenv()                                    # True
```

```
from langsmith import Client
from langsmith import traceable

import os

# LangSmith 환경 변수 확인

print("\n--- LangSmith 환경 변수 확인 ---")
langchain_tracing_v2 = os.getenv('LANGCHAIN_TRACING_V2')
langchain_project = os.getenv('LANGCHAIN_PROJECT')
langchain_api_key_status = "설정됨" if os.getenv('LANGCHAIN_API_KEY') else "설정되지

if langchain_tracing_v2 == "true" and os.getenv('LANGCHAIN_API_KEY') and langchai
    print(f"✅ LangSmith 추적 활성화됨 (LANGCHAIN_TRACING_V2='{langchain_tracing_v2}
    print(f"✅ LangSmith 프로젝트: '{langchain_project}'")
    print(f"✅ LangSmith API Key: {langchain_api_key_status}")
    print("  -> 이제 LangSmith 대시보드에서 이 프로젝트를 확인해 보세요.")
else:
    print("❌ LangSmith 추적이 완전히 활성화되지 않았습니다. 다음을 확인하세요:")
    if langchain_tracing_v2 != "true":
        print(f"  - LANGCHAIN_TRACING_V2가 'true'로 설정되어 있지 않습니다 (현재: '{langc
    if not os.getenv('LANGCHAIN_API_KEY'):
        print("  - LANGCHAIN_API_KEY가 설정되어 있지 않습니다.")
    if not langchain_project:
        print("  - LANGCHAIN_PROJECT가 설정되어 있지 않습니다.")
```

```
# 샘플 벡터DB 구축
from langchain_community.document_loaders import WebBaseLoader
from langchain.vectorstores import FAISS
from langchain_huggingface import HuggingFaceEmbeddings
from langchain_text_splitters import RecursiveCharacterTextSplitter
import warnings
warnings.filterwarnings("ignore")
```

```
# Embeddings 사용

embeddings = HuggingFaceEmbeddings(
        model_name="sentence-transformers/all-mpnet-base-v2",              # 768
        model_kwargs={'device': 'cpu'},
        encode_kwargs={'normalize_embeddings': True}
    )                                                                      # 6.5
```

```
# 블로그 포스트 로드

loader = WebBaseLoader(
    "https://teddylee777.github.io/openai/openai-assistant-tutorial/", encoding="
)
```

```
print(type(loader))
```

```
# 문서 분할
text_splitter = RecursiveCharacterTextSplitter(chunk_size=500, chunk_overlap=0)
docs = loader.load_and_split(text_splitter)                                    # 0.7
```

```
# 벡터DB 생성
db = FAISS.from_documents(docs, embeddings)                                    # 29.
```

```
# retriever 생성
retriever = db.as_retriever()
```

```
# 문서 검색
query = "OpenAI Assistant API의 Functions 사용법에 대해 알려주세요."
relevant_docs = retriever.invoke(query)
```

```
# 검색된 문서의 개수 출력

len(relevant_docs)                                                             # 4
```

- 검색된 결과 중 문서의 내용을 하나씩 출력해보기

```
# 0번 문서 출력하기

print(relevant_docs[0].page_content)
```

```
# 1번 문서 출력하기

print(relevant_docs[1].page_content)
```

```
# 2번 문서 출력하기

print(relevant_docs[2].page_content)
```

```
# 3번 문서 출력하기

print(relevant_docs[3].page_content)
```

- 0번 문서 출력하기

```
OpenAI의 새로운 Assistants API는 대화와 더불어 강력한 도구 접근성을 제공합니다. 본 튜토리얼은 Ope



주요내용
```

- 1번 문서 출력하기

```
OpenAI는 Assistants API를 지원하기 위해 Python 라이브러리를 업데이트했습니다. 따라서 최신 버전으

이 문서는 openai 라이브러리를 최신 버전으로 업그레이드하고 설치하는 방법을 설명합니다. pip 명령어를 사
# openai 라이브러리를 최신 버전으로 업그레이드하여 설치합니다.
!pip install --upgrade openai -q

그리고 다음을 실행하여 최신 상태인지 확인하세요.
openai 패키지의 버전 정보를 확인 합니다.
# openai 패키지의 버전 정보를 확인합니다.
!pip show openai | grep Version


Version: 1.12.0
```

- 2번 문서 출력하기

```
질문에 답변할 때 Assistant가 제공된 문서나 지식 기반으로 답변할 수 있게하는 기능입니다.


검색은 독점적인 제품 정보나 사용자가 제공한 문서 등 모델 외부의 지식으로 Assistant 의 답변을 보강합니

파일을 업로드하여 어시스턴트에 전달하면 OpenAI가 자동으로 문서를 청크 처리(분할)하고, 임베딩을 색인화 

이 기능 역시 대시보드에서 업데이트할 수 있거나 API에서도 활성화할 수 있으며, 방법은 아래에서 다룹니다.
파일 업로드
실습을 위해 활용한 파일을 미리 준비합니다.
참고


지원되는 파일 목록은 지원파일 목록 에서 확인할 수 있습니다(대부분의 문서 형식은 지원합니다).


단, 최대 파일 크기는 512MB, 토큰 수는 2,000,000개 이하입니다(파일 첨부 시 자동으로 계산됨).
```

- 3번 문서 출력하기

```
Assistant 를 생성하지 않은 경우
```

```
# 1-1. Assistant ID를 불러옵니다(Playground에서 생성한 Assistant ID)
ASSISTANT_ID = "asst_V8s4Ku4Eiid5QC9WABlwDsYs"

# 1-2. Assistant 를 생성합니다.
from openai import OpenAI

# OpenAI API를 사용하기 위한 클라이언트 객체를 생성합니다.
client = OpenAI(api_key=api_key)
```

---

3) `사용방법`

- **`MultiQueryRetriever`** 에 사용할 `LLM`을 지정 → `질의 생성`에 사용 → `retriever` = 나머지 작업 처리

```
from langchain.retrievers.multi_query import MultiQueryRetriever
from langchain_google_genai import ChatGoogleGenerativeAI
from dotenv import load_dotenv
import os

load_dotenv()

# API 키 확인
if not os.getenv("GOOGLE_API_KEY"):
    os.environ["GOOGLE_API_KEY"] = input("Enter your Google API key: ")

# LLM 초기화
gemini_lc = ChatGoogleGenerativeAI(
        model="gemini-2.5-flash-lite",
        temperature=0,                                          # temperature
        max_output_tokens=4096,
    )

# MultiQueryRetriever를 언어 모델을 사용하여 초기화하기
multiquery_retriever = MultiQueryRetriever.from_llm(
    retriever=db.as_retriever(),                                # 벡터 데이터베이
    llm=gemini_lc,
)
```

- 다중 쿼리를 생성하는 중간 과정을 디버깅하기 위해 실행하는 코드
  - **`langchain.retrievers.multi_query`** 로커 가져오기
  - **`logging.getLogger()`** 함수 사용
  - 로거의 로그 레벨 = **`INFO`** 설정 → *`INFO`* 레벨 이상의 로그 메시지만 출력되도록 할 수 있음

```
# 쿼리에 대한 로깅 설정
import logging

logging.basicConfig()
logging.getLogger("langchain.retrievers.multi_query").setLevel(logging.INFO)
```

- **`retriever_from_llm`** 객체의 **`invoke`** 메서드 사용 → 주어진 **`question`** 과 관련된 문서 검색
- 검색된 문서 = **`unique_docs`** 변수에 저장
  - 이 변수의 길이를 확인 = 검색된 관련 문서의 총 개수를 확인할 수 있음
- 이 과정을 통해 사용자의 질문에 대한 관련 정보를 효과적으로 찾아내고 그 양을 파악할 수 있음

```
# 질문 정의하기
question = "OpenAI Assistant API의 Functions 사용법에 대해 알려주세요."

# 문서 검색하기
relevant_docs = multiquery_retriever.invoke(question)

# 검색된 고유한 문서의 개수를 반환하기
print(
    f"===============\n검색된 문서 개수: {len(relevant_docs)}",
    end="\n===============\n",
)

# 검색된 문서의 내용 출력하기
print(relevant_docs[0].page_content)
```

- 0번째 문서 내용 출력하기 (1.2s)

```
INFO:langchain.retrievers.multi_query:Generated queries: ['OpenAI Assistant API(
===============
검색된 문서 개수: 5
===============
OpenAI의 새로운 Assistants API는 대화와 더불어 강력한 도구 접근성을 제공합니다. 본 튜토리얼은 Ope



주요내용
```

---

4) `LCEL Chain 활용하는 방법`

- 사용자 정의 프롬프트 정의 → 프롬프트와 함께 `Chain` 생성
- `Chain`

- 사용자의 질문 입력 받으면 *(아래의 예제의 경우)* 5개의 질문 생성 → `"\n"` 구분자로 구분 → 생성된 5개의 질문 반환하기

```
from langchain_core.runnables import RunnablePassthrough
from langchain_core.prompts import PromptTemplate
from langchain_core.output_parsers import StrOutputParser

# 프롬프트 템플릿을 정의하기
# 5개의 질문을 생성하도록 프롬프트를 작성함
prompt = PromptTemplate.from_template(
    """You are an AI language model assistant.
Your task is to generate five different versions of the given user question to re
By generating multiple perspectives on the user question, your goal is to help th
Your response should be a list of values separated by new lines, eg: `foo\nbar\nb

#ORIGINAL QUESTION:
{question}

#Answer in Korean:
"""
)

# 언어 모델 인스턴스를 생성하기
gemini_lc = ChatGoogleGenerativeAI(
        model="gemini-2.5-flash-lite",
        temperature=0,                                          # temperature
        max_output_tokens=4096,
    )

# LLMChain 생성하기
custom_multiquery_chain = (
    {"question": RunnablePassthrough()} | prompt | gemini_lc | StrOutputParser()
)

# 질문 정의하기
question = "OpenAI Assistant API의 Functions 사용법에 대해 알려주세요."

# 체인을 실행하여 생성된 다중 쿼리를 확인하기
multi_queries = custom_multiquery_chain.invoke(question)

# 결과 확인하기
# 5개 질문 생성
multi_queries
```

- 다중 쿼리 생성하기 (체인 생성) (0.9s)

```
'OpenAI Assistant API에서 함수 호출 기능을 어떻게 사용하는지 설명해주세요.\n
OpenAI Assistant API의 함수 기능에 대한 사용 가이드라인을 제공해주세요.\n
Assistant API를 사용하여 OpenAI에서 함수를 호출하는 방법에 대한 정보를 얻고 싶습니다.\n
OpenAI Assistant API의 함수 기능 구현 방법을 알려주세요.\n
OpenAI Assistant API에서 함수를 정의하고 사용하는 예시를 보여주세요.'
```

- 이전에 생성한 `Chain` → **`MultiQueryRetriever`** 에 전달 → `retriever` 가능

```
multiquery_retriever = MultiQueryRetriever.from_llm(
    llm=custom_multiquery_chain, retriever=db.as_retriever()
)
```

- **`MultiQueryRetriever`** 사용 → 문서 검색, 결과 확인 가능

```
# 결과
relevant_docs = multiquery_retriever.invoke(question)


# 검색된 고유한 문서의 개수 반환하기
print(
    f"===============\n검색된 문서 개수: {len(relevant_docs)}",
    end="\n===============\n",
)

# 검색된 문서의 내용을 출력하기
print(relevant_docs[0].page_content)
```

- 셀 출력

```
INFO:langchain.retrievers.multi_query:Generated queries: ['OpenAI Assistant API
===============
검색된 문서 개수: 5
===============
OpenAI의 새로운 Assistants API는 대화와 더불어 강력한 도구 접근성을 제공합니다. 본 튜토리얼은 Ope



주요내용
```

- *next:* ***`다중 벡터저장소 검색기 (MultiVectorRetriever)`***
