05. 랭체인으로 RAG 구현하기

5.3 PDF 요약 웹사이트 만들기~

5.4 독립형 질문 챗봇 만들기

발표 일시 : 2025.05.06(화)
발표자 : 김효진

목차

1. 배경

- 기존 목적 : 로컬에 LLM 구현하기
- 그러나, API를 사용하지 않고 로컬에 LLM 개발시 한계
  1. 하드웨어 제약
  - GPU 없이 실행 시 추론 속도가 느리고 실시간 응답이 사실상 불가능
  2. 기회비용
  - 모델 튜닝보다 인프라 문제 해결에 더 많은 시간 소모
- 전환 배경
  1. LangChain과의 호환성
  - LangChain에서 API 기반 LLM (ChatOpenAI, HuggingFaceHub)을 기본 지원

2. 개요

<RAG를 활용한 PDF 문서 전처리>

- 회사에서 거의 대부분의 데이터는 문서로 존재함
- 그러나 개별적으로 시행되어 통합 관리가 어려움 -> 자료 탐색이 매우 불편 + 최신화에도 어려움
- 업무 생산성을 개선하기 위해 다양한 문서를 처리하는 것이 필요

3. 학습내용

3-1. 기본 구조

<PDF 문서 기반 QA 따라하기>

- RAG 프로세스

1	문서 로드	5	검색(Retriever)
2	문서를 chunk 단위로 분할	6	검색 결과를 바탕으로 원하는 결과를 도출하기 위한 프롬프트 작성
3	문서를 벡터 표현으로 변환	7	모델 선택
4	벡터를 DB에 저장	8	결과

3. 학습내용

3-1. 기본 구조

```
# 단계 1: 문서 로드(Load Documents)
loader = PyMuPDFLoader("sample.pdf")
docs = loader.load()

# 단계 2: 문서 분할(Split Documents)
text_splitter = RecursiveCharacterTextSplitter(chunk_size=1000, chunk_overlap=50)
split_documents = text_splitter.split_documents(docs)

# 단계 3: 임베딩(Embedding) 생성
embeddings = OpenAIEmbeddings()

# 단계 4: DB 생성(Create DB) 및 저장
# 벡터스토어를 생성합니다.
vectorstore = FAISS.from_documents(documents=split_documents, embedding=embeddings)
```

3. 학습내용

3-1. 기본 구조

```
# 단계 5: 검색기(Retriever) 생성
retriever = vectorstore.as_retriever()

# 단계 6: 프롬프트 생성(Create Prompt)
# 프롬프트를 생성합니다.
prompt = PromptTemplate.from_template(
    """

다양한 개선 방식이 존재

"""
)
```

3. 학습내용

3-1. 기본 구조

```
# 단계 7: 언어모델(LLM) 생성
# 모델(LLM) 을 생성합니다.
llm = ChatOpenAI(model_name="gpt-4o", temperature=0)


# 단계 8: 체인(Chain) 생성
chain = (
    {"context": retriever, "question": RunnablePassthrough()}
    | prompt
    | llm
    | StrOutputParser()
)
```

3. 학습내용

3-1. 기본 구조

결과

question = "퍼플렉시티와 오픈AI 딥리서치 차이는?"

response = chain.invoke(question)

print(response)

==========

퍼플렉시티와 오픈AI의 딥 리서치의 주요 차이는 다음과 같습니다:

1. **가격 및 접근성**: 오픈AI의 딥 리서치는 유료 구독자에게만 제공되며, 초기에는 월 200달러의 구독료가 필요합니다. 반면, 퍼플렉시티의 딥 리서치는 무료로 제공되며, 무료 사용자도 하루 5개의 쿼리를 실행할 수 있습니다.
2. **정확도**: HLE 벤치마크에서 오픈AI의 딥 리서치는 26.6%의 정확도를 기록한 반면, 퍼플렉시티의 딥 리서치는 21.1%의 정확도를 기록했습니다.
3. **작업 완료 시간**: 퍼플렉시티의 딥 리서치는 대부분의 작업을 3분 이내에 완료하는 반면, 오픈AI의 딥 리서치는 작업 완료에 5~30분이 걸릴 수 있습니다.

이러한 차이점들이 퍼플렉시티와 오픈AI 딥 리서치의 주요 차이점입니다.

3. 학습내용

3-2. 추가학습

VectorStoreRetrieverMemory

- 백터 스토어에 메모리를 저장하고 호출될 때마다 가장 '눈에 띄는' 상위 K개의 문서를 쿼리
- 대화순서를 명시적으로 추적하지 않음

벡터 스토어 종류

Faiss	Chroma	Pinecone
Python 패키지 (faiss-cpu, faiss-gpu) -> 로컬 O	Python 패키지 (chromadb) -> 로컬 O	API 키 발급 후 RESTful or SDK 호출 -> 로컬 X
초기 비용이 낮고, 커스터마이징 자유도가 높음		대규모 데이터셋에 뛰어난 확장성 제공
오픈소스 - Apache 2.0	오픈소스 - MIT	유료

출처 : https://wikidocs.net/251190

3. 학습내용

3-2. 추가학습

벡터 저장소 생성 (from_documents)

db = Chroma.from_documents(documents=split_doc1 + split_doc2, embedding=OpenAIEmbeddings())

벡터 저장소를 검색기로 변환(as_retriever)

retriever = db.as_retriever(search_type="mmr", search_kwargs={"k": 6, "lambda_mult": 0.25, "fetch_k": 10})

- search_type (검색 유형 : "similarity", "mmr", "similarity_score_threshold")
  - similarity : 상위 `k`개 유사도 높은 결과만 반환
  - mmr : 다양성 + 유사도 균형 위해 후보군 필요
  - similarity_score_threshold : threshold 이상 중 상위 k개 추출
- search_kwargs (검색 함수에 전달할 추가 인자)
  - k: 반환할 문서 수 (기본값: 4)
  - score_threshold: 최소 유사도 임계값
  - fetch_k: 검색 후보군의 수 (기본값: 20) / similarity에서는 사용 X
  - lambda_mult: 유사한 문서를 뽑되, 비슷한 문장만 고르지 않도록 하기 위한 조절값 (0~1, 기본값: 0.5)
  - filter: 문서 메타데이터 필터링

3. 학습내용

3-2. 추가학습

Chroma 와 Faiss 차이 : 벡터 저장소 디렉토리

벡터 저장소	저장 디렉토리 지정 방법	저장 함수	경로 미지정시
Chroma	persist_directory="경로"	.persist()	기본 임시 폴더에 저장됨
FAISS	FAISS.save_local("경로")	.save_local()	오류 발생

3. 학습내용

3-3. 보완

1. 업무 활용시 고려할 점

- Drm 문서, 한글파일 읽어오기
- 문서 내 다양한 형식의 자료(표, 이미지 등) 읽어오기

2. 대용량 문서 처리 방법

- 수백만건 이상의 대용량 문서를 어떻게 벡터 스토어로 구축해야 할까?

3. 아직 전체 LLM 파이프라인에 대한 세부 이해 부족

- Runnable
- Chain

4. 후기

- 새로 알게된 것 : Langsmith
  - LLM 기반 애플리케이션의 실시간 실행 로그 추적, 오류 디버깅, 성능 평가를 도움
  - ex) rag에서 실제 원하는 문서를 활용했는지 확인 가능

- 문서 요약은 시중에서 가장 흔한 서비스라서 구현이 쉬울 것이라고 판단했음
- 그러나 직접 해보는 것은 다른 일

- 가장 흔하지만 보안과 신뢰 이슈로 전사 활용 목적으로 구현하기에 까다로운 서비스일 것 같다고 생각
- 따라서 개인 업무 활용 목적(생산성 향상, 인수인계 등)으로 서비스 개발을 지속하는 것은 의미있다고 판단