섹션 3: 데이터 처리 및 임베딩 기법

- 패키지 설치
    - conda

```
pip install pypdf bs4 jq tiktoken transformers langchain_experimental
langchain_huggingface langchain_ollama deepl langdetect
```

    - poetry

```
poetry add pypdf bs4 jq tiktoken transformers langchain_experimental
langchain_huggingface langchain_ollama deepl langdetect
```

1. 다양한 문서 형식 처리하기

1.1 PDF 문서

- **langchain_community** 패키지에서 제공하는 **PyPDFLoader**를 사용
- PDF 파일에서 텍스트를 추출 (페이지별로 구분하여 문서 객체로 변환)
- pypdf 패키지 설치
- 이외에도 PyMuPDF, Unstructured 등 다양한 랭체인 도구가 있음

1.2 웹 문서

- **langchain_community** 패키지에서 제공하는 **WebBaseLoader**를 사용
- 특정 웹 페이지의 내용을 로드
- bs4 패키지 설치
- 이외에도 RecursiveUrlLoader , SitemapLoader 등 다양한 랭체인 도구가 있음

1.3 JSON 파일

- **langchain_community** 패키지에서 제공하는 **JSONLoader**를 사용
- JSON 파일, JSONL 파일의 내용을 내용을 로드
- jq 패키지 설치

```
{
  "chatroom_name": "프로젝트 팀",
  "members": ["김철수", "이영희", "박민수"],
  "messages": [
    {
      "sender": "김철수",
      "timestamp": "2023-09-15 09:30:22",
      "content": "안녕하세요 여러분, 오늘 회의 시간 확인차 연락드립니다."
    },
    {
      "sender": "이영희",
      "timestamp": "2023-09-15 09:31:05",
      "content": "네, 안녕하세요. 오후 2시에 하기로 했어요."
    },
  ]
}
```

```
{"sender": "김철수", "timestamp": "2023-09-15 09:30:22", "content": "안녕하세요 여러분, 오늘 회의 시간 확인차 연락드립니다."}
{"sender": "이영희", "timestamp": "2023-09-15 09:31:05", "content": "네, 안녕하세요. 오후 2시에 하기로 했어요."}
```

1.4 CSV 문서

- **langchain_community** 패키지에서 제공하는 **CSVLoader**를 사용
- CSV 파일의 각 행을 추출하여 개별 문서 객체로 변환
- 패키지 설치: 해당사항 없음

2. 효과적인 텍스트 분할 전략

분할의 필요성

- 토큰 제한 극복
- 관련성 높은 컨텍스트 제공

분할 방법

1. 문자 레벨 분할 (CharacterTextSplitter)
2. 재귀적 분할 (RecursiveCharacterTextSplitter)
3. 토큰 기반 분할 (tiktoken, HuggingFace tokenizer)

코드 예시 (재귀적 분할)

```
from langchain_text_splitters import RecursiveCharacterTextSplitter

text_splitter = RecursiveCharacterTextSplitter(
    chunk_size=100,
    chunk_overlap=0,
)
docs = text_splitter.split_documents(data)
```

분할 결과 시각화

원본 문서: "이것은 긴 문서의 예시입니다. 문서는 여러 청크로 나뉩니다. 각 청크는 일정 크기를 가지며, 오버랩 영역이 있을 수 있습니다."

↓

1 이것은 긴 문서의 예시입니다. 문서는 여러 청크로 나뉩니다. 각 청크는 일정 크기를

2 각 청크는 일정 크기를 가지며, 오버랩 영역이 있을 수 있습니다.

3 있습니다. (이후 문서 내용 계속...)

오버랩 영역

참고: chunk_size와 chunk_overlap을 조절하여 분할 결과를 최적화할 수 있습니다.
- 큰 chunk_size: 더 많은 컨텍스트, 작은 chunk_size: 더 정확한 검색
- 오버랩: 청크 간 연속성 유지, 정보 손실 방지

2.1 RecursiveCharacterTextSplitter 사용

- LangChain에서 제공하는 고급 텍스트 분할 도구
- 텍스트를 재귀적으로 분할하여 더 자연스러운 청크를 생성
- 특징:

1. 재귀적으로 텍스트를 분할합니다.
2. 구분자를 순차적으로 적용하여 큰 청크에서 시작하여 점진적으로 더 작은 단위로 분할
3. 일반적으로 CharacterTextSplitter보다 더 엄격하게 크기를 준수하려는 경향

2.2 정규표현식 사용

- 정규표현식을 사용하면 특정 패턴을 기반으로 텍스트를 더 정확하게 분할 가능
- 구조화된 텍스트나 특정 형식의 문서에 유용
- 예시:
    1. 1장, 2장, 3장, …
    2. 문장 단위로 구분 (마침표, 느낌표, 물음표로 끝나는 문장)

2.3 토큰 수를 기반으로 분할

- 토큰 기반 분할은 LLM의 토큰 제한을 고려할 때 유용
- 각 청크가 특정 토큰 수를 초과하지 않도록 조절 가능
- tiktoken, transformers 패키지 설치
- 실습:
    - tiktoken: OpenAI에서 만든 BPE Tokenizer
    - Hugging Face tokenizer : 허깅페이스 오픈소스 모델

2.4 맥락을 기반으로 분할 (Semantic Chunking)

- 의미적 유사성에 기반하여 분할
- 문장들 사이의 임베딩 차이를 기반으로 작동
- langchain_experimental 패키지 설치
- 실습: OpenAI 임베딩 사용

3. 임베딩 모델의 활용

3.1 임베딩 모델이란

- 텍스트를 벡터 표현으로 변환하는 모델 (임베딩 벡터: 텍스트의 의미를 숫자 배열로 나타낸 것)
- 텍스트를 벡터로 표현함으로써 의미적으로 가장 유사한 다른 텍스트를 찾는 등의 수학적 연산을 수행 가능
- 의미 기반 검색, 문서 분류, 텍스트 유사도 분석 등 다양한 자연어 처리 작업을 수행
- OpenAI, Hugging Face, Ollama 등 다양한 임베딩 모델
- LangChain **Embeddings** 클래스:
    1. **embed_documents**: 여러 텍스트를 입력받아 임베딩
    2. **embed_query**: 단일 텍스트를 입력받아 임베딩

3.2 Openal 모델 사용

- OpenAI의 임베딩 모델을 사용하려면 OpenAIEmbeddings 클래스를 활용
- OpenAI API 키가 필요하며, 환경 변수로 설정하거나 직접 전달 가능
- 장점:
    - 높은 품질의 임베딩을 제공
    - 다양한 언어와 도메인에 대해 잘 작동
    - 지속적으로 업데이트되어 최신 기술을 반영
- 단점:

    - API 사용에 비용이 발생
    - 데이터가 외부 서버로 전송되므로 개인정보 보호 문제가 있을 수 있음
    - 인터넷 연결이 필요
- 링크: https://platform.openai.com/docs/guides/embeddings/embedding-models

3.3 HuggingFace 모델 사용

- HuggingFace의 임베딩 모델을 사용하려면 HuggingFaceEmbeddings 클래스를 활용
- 다양한 사전 훈련된 모델을 선택 가능
- langchain_huggingface 패키지 설치
- 장점:
    - 다양한 사전 훈련 모델 중에서 선택 가능
    - 오픈소스이므로 무료로 사용 가능
    - 로컬에서 실행 가능하여 개인정보 보호에 유리
- 단점:
    - 모델에 따라 성능 차이가 있음
    - 로컬 실행 시 하드웨어 자원이 필요
    - 일부 모델은 큰 용량을 차지할 수 있음 (실행 속도가 느림)
- 링크: https://huggingface.co/models?other=embeddings

3.4 Ollama 모델 사용

- Ollama의 임베딩 모델을 사용하려면 OllamaEmbeddings 클래스를 활
- Ollama는 로컬에서 실행되는 오픈소스 모델을 제공
- langchain_ollama 패키지 설치
- 장점:
    - 로컬에서 실행되어 개인정보 보호에 유리
    - 오픈소스로 무료 사용이 가능
    - 다양한 모델을 쉽게 설치하고 사용할 수 있음
- 단점:
    - 일부 대규모 모델의 경우 강력한 하드웨어가 필요
    - OpenAI나 일부 HuggingFace 모델에 비해 성능이 떨어질 수 있음 (특히, 한국어)
    - 모델 선택과 설정에 따라 결과가 크게 달라질 수 있음
- 링크: https://ollama.com/search?c=embedding

3.5 모델 선택 시 고려사항

1. 성능: 작업의 정확도가 중요한 경우 OpenAI나 잘 훈련된 HuggingFace 모델이 적합
2. 비용: 예산이 제한적인 경우 HuggingFace나 Ollama 모델이 좋은 선택
3. 개인정보 보호: 민감한 데이터를 다루는 경우 로컬에서 실행 가능한 HuggingFace나 Ollama 모델이 적합
4. 리소스: 가용 하드웨어 자원에 따라 적절한 모델을 선택

4. 다국어 RAG 시스템 구축 실습

4.1 다국어 지원 임베딩 모델을 활용한 언어 교차(cross-lingual) 검색

- 하나의 다국어 지원 임베딩 모델로 하나의 벡터저장소에 다국어 문서를 한번에 저장
- 임베딩 모델의 다국어 이해 능력이 중요
- OpenAI, Huggingface, Ollama 임베딩 성능 비교

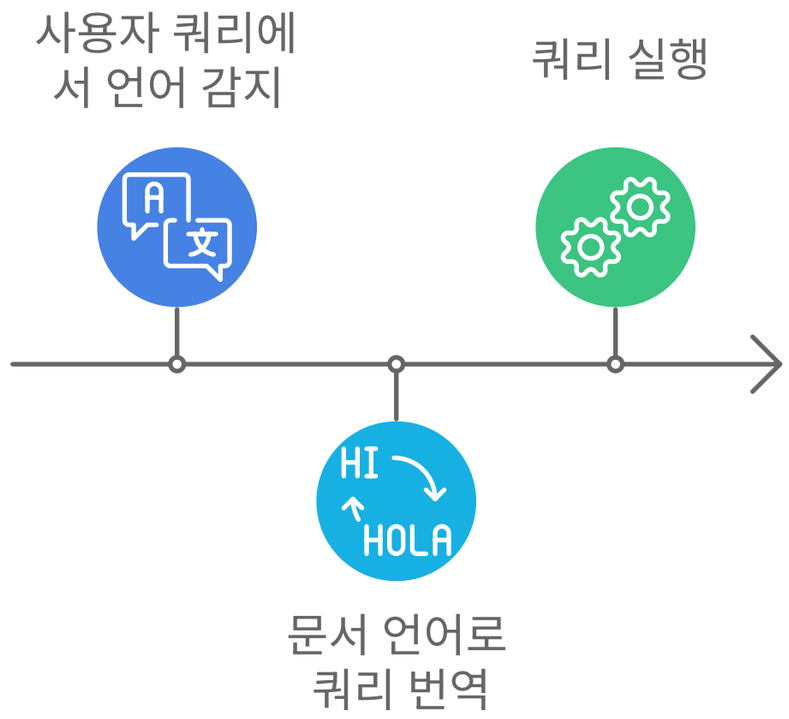
```
# OpenAI 임베딩 모델 생성
embeddings_openai = OpenAIEmbeddings(model="text-embedding-3-small")

# Hugoing Face 임베딩 모델 생성
embeddings_huggingface = HuggingFaceEmbeddings(model_name="BAAI/bge-m3")

# Ollama 임베딩 모델 생성
embeddings_ollama = OllamaEmbeddings(model="nomic-embed-text")
```

4.2 언어 감지 및 자동번역 통합

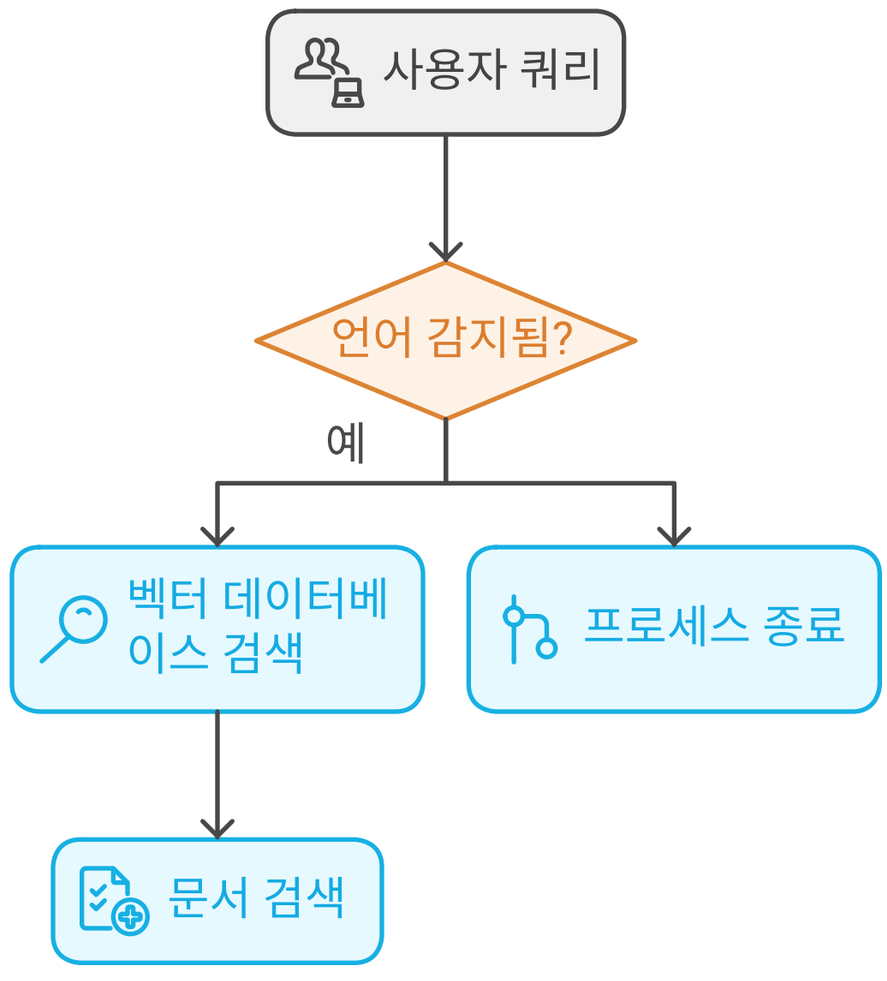
- 사용자의 쿼리에 사용된 언어를 감지하여 해당 언어로 벡터저장소의 문서와 같은 언어로 자동번역하여 쿼리 수행



- 자동화로 인해 때때로 부정확한 언어 감지나 번역 오류가 발생할 수 있음
- deepl, langdetect

4.3 언어 감지 및 검색 라우팅

- 사용자의 쿼리에 사용된 언어를 감지하여 해당 언어 문서가 저장되어 있는 벡터 저장소에서 검색
- 쿼리와 문서 간에 동일한 언어를 기반으로 하는 장점



4. Gradio 챗봇

- 채팅 히스토리 추가


QA Bot
Chatbot
특정 기업이나 개념을 의미하는 것이라면, 추가 정보를 주시면 더 구체적으로 설명해 드리겠습니다!
내가 이전에 이야기한 타슬라 회장의 이름은?
이전에 언급하신 "타슬라 회장"은 "홍길동"이라는 이름이었습니다. 하지만 "타슬라"라는 이름이 특정 기업을 지칭하는 것인지, 아니면 다른 의미가 있는지에 대한 정보가 부족하여 정확한 답변을
Retry
Undo
Clear
Type a message...
Submit
Use via API
· gradio로 제작되었습니다

—-