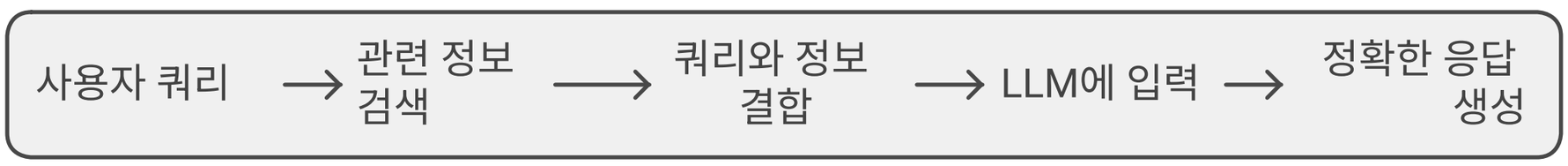
섹션 1: RAG와 LangChain 기초

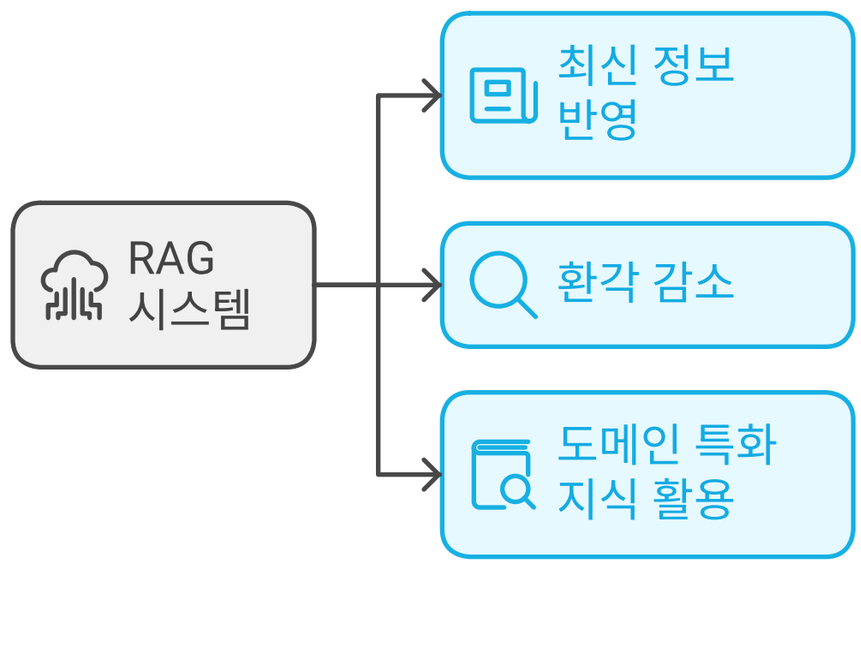
1. RAG의 기본 개념과 LangChain 소개

1.1 RAG(Retrieval-Augmented Generation)

- RAG는 대규모 언어 모델(LLM)의 성능을 향상시키기 위해 외부 지식을 활용하는 기술
- 기본 원리:
    - 질문에 관련된 정보를 검색
    - 검색된 정보를 LLM의 입력에 추가하여 더 정확하고 관련성 높은 답변을 생성



- RAG의 장점



- 활용 사례:
    1. 기업 내부 지식베이스 기반 챗봇
    2. 개인화된 고객 지원 시스템
    3. 법률 문서 분석 및 요약

1.2 LangChain 프레임워크 소개

- LangChain은 LLM 애플리케이션 개발을 위한 파이썬 프레임워크
- 주요 기능:
    1. 다양한 LLM과의 통합
    2. 문서 로딩 및 처리
    3. 벡터 저장소 관리
    4. 프롬프트 관리 및 체이닝

- LangChain을 사용한 RAG 구현의 이점
    1. 모듈화된 컴포넌트로 빠른 프로토타이핑 가능
    2. 다양한 백엔드와의 쉬운 통합
    3. 커뮤니티 지원 및 지속적인 업데이트

2. LangChain 설치 및 환경 설정

2.1 Python 환경 설정

a) Conda를 사용한 가상환경 생성

- Conda 설치
    - 공식 문서: https://docs.anaconda.com/miniconda/
    - 설치 영상 (Windows): https://youtu.be/x1ioyg1PS34

- Conda 설치 확인:

```
conda --version
```

- 새로운 환경 생성:

```
conda create --name langchain_env python=3.11
```

- 환경 활성화:

```
conda activate langchain_env
```

- 환경 비활성화 (작업 완료 후)

```
conda deactivate
```

b) Poetry를 사용한 가상환경 생성

- Poetry 설치 (아직 설치하지 않은 경우):
    - 공식 문서: https://python-poetry.org/docs/#installation
    - 설치 영상 (Windows):

프로젝트로 배우는 Python 챗봇 만들기 - LangChain, Gradio 활용

- 섹션 1 들어가며 ——> 2번째 강의: 개발환경 - Poetry 패키지 관리자 설치
- https://www.inflearn.com/course/lecture?courseSlug=%ED%94%84%EB%A1%9C%EC%A0%9D%ED%8A%B8%EB%A1%9C-%EB%B0%B0%EC%9A%B0%EB%8A%94-%ED%8C%8C%EC%9D%B4%EC%8D%AC-%EC%B1%97%EB%B4%87%EB%A7%8C%EB%93%A4%EA%B8%B0&unitId=213541
- 새 프로젝트 생성:

```
poetry new langchain_project
cd langchain_project
```

- Python 버전 지정 (pyproject.toml 파일 편집):

```
[tool.poetry.dependencies]
python = "^3.11"
```

- 가상환경 생성 및 활성화:

```
poetry env use python3.11
poetry shell
```

- 환경 비활성화 (작업 완료 후):

```
exit
```

- **[**강의 영상] Mac OS + Poetry 조합으로 강의를 진행합니다. 파이썬 기반의 가상환경에서 진행하기 때문에 Mac OS + Anaconda, Windows + Poetry, Windows + Anaconda 조합으로 강의를 수강하시는데 문제가 없습니다.

2.2 LangChain 및 필요한 라이브러리 설치

a) Conda를 사용한 설치:

- 가상환경 활성화

```
conda activate langchain_env
```

- 패키지 설치

```
conda install -c conda-forge langchain_openai langchain_community
langchain_chroma gradio
```

- 실습 패키지 한꺼번에 설치 (선택) - 첨부 수업자료의 requirements.txt 사용

```
pip install -r requirements.txt
```

b) Poetry를 사용한 설치:

- 프로젝트 폴더로 이동해서 패키지 설치

```
poetry add langchain langchain_openai langchain_community langchain_chroma
gradio
```

- 실습 패키지 한꺼번에 설치 (선택)
    1. 첨부 수업자료에서 pyproject.toml 내용을 그대로 복사
    2. 프로젝트 폴더에서 아래 명령을 실행

```
poetry install
```

2.3 환경 변수 설정 (OpenAI API 키):

- python-dotenv 설치

```
conda env: pip install python-dotenv
poetry env: poetry add python-dotenv
```

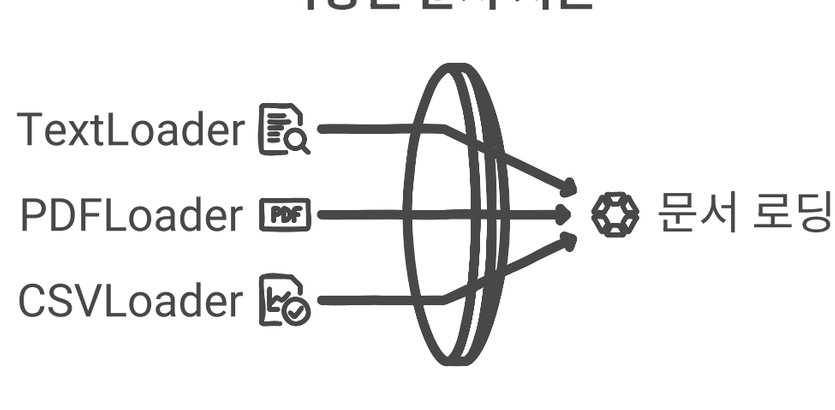
- 프로젝트 폴더에 .env 파일을 만들고 편집

```
OPENAI_API_KEY=your-api-key-here
```

3. LangChain의 주요 RAG 컴포넌트

3.1 문서 로더 (Document Loaders)

- 다양한 형식의 문서를 로드하는 기능 제공
- TextLoader 외에도 PDFLoader, CSVLoader 등 다양한 로더 존재



주요 단계

텍스트 파일 목록 가져오기 → TextLoader로 파일 로드 → 로드된 데이터 구조 확인

코드 예시

```
from glob import glob
from langchain_community.document_loaders import TextLoader

txt_files = glob(os.path.join('data', '*_KR.txt'))
data = []
for text_file in txt_files:
    loader = TextLoader(text_file, encoding='utf-8')
    data += loader.load()
```

Document 객체

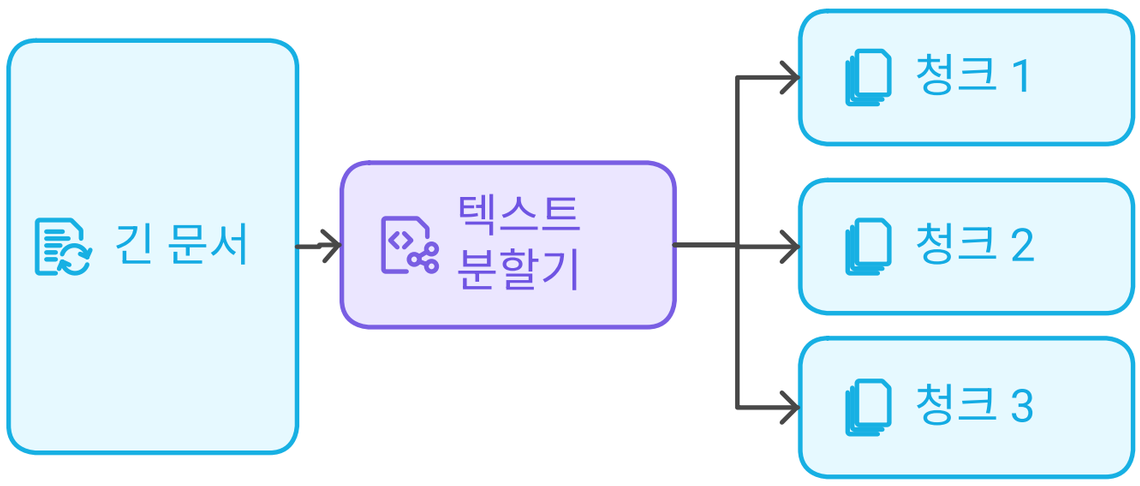
page_content
텍스트 파일의 내용

metadata

- source: 파일 경로
- file_path: 파일 경로
- file_name: 파일 이름
- file_type: 파일 타입

3.2 텍스트 분할기 (Text Splitters)

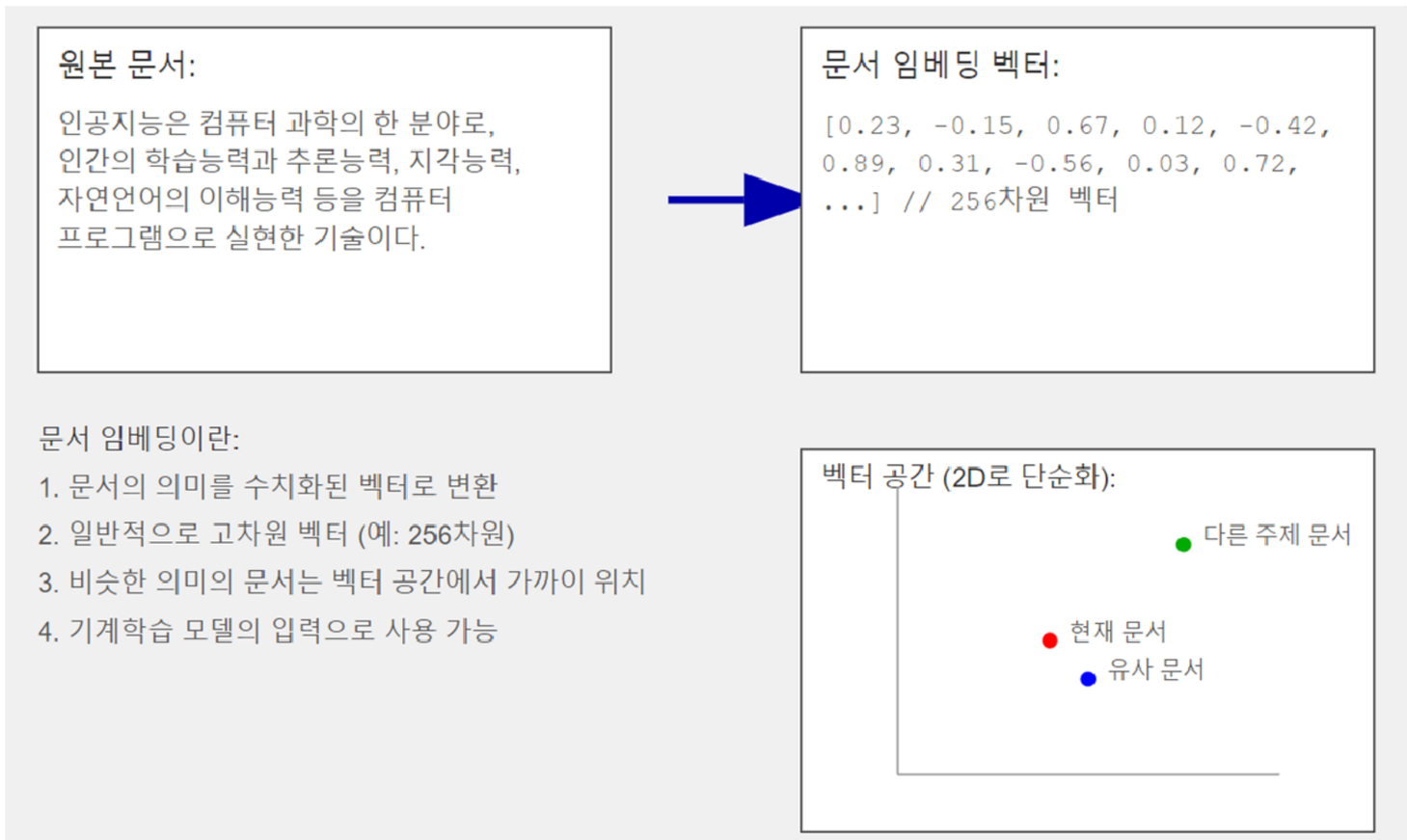
- 긴 문서를 처리 가능한 크기의 청크로 분할
- 다양한 분할 전략 제공 (문자 기반, 토큰 기반 등)




```
text_splitter = CharacterTextSplitter(chunk_size=1000, chunk_overlap=0)
texts = text_splitter.split_documents(documents)
print(f"생성된 텍스트 청크 수: {len(texts)}")
print(f"첫 번째 청크 내용 미리보기: {texts[0].page_content[:100]}...")
```

3.3 임베딩 모델 (Embeddings)

- 텍스트를 벡터로 변환하는 모델



- 임베딩 모델의 활용

- LangChain Embeddings 클래스

embed_documents 메소드

- 기능: 여러 텍스트를 한 번에 임베딩
- 입력: 문자열 리스트
- 출력: 임베딩 벡터 리스트

```python
from langchain.embeddings import OpenAIEmbeddings

embeddings = OpenAIEmbeddings()
texts = ["Hello world", "LangChain is awesome"]
doc_vectors = embeddings.embed_documents(texts)
```

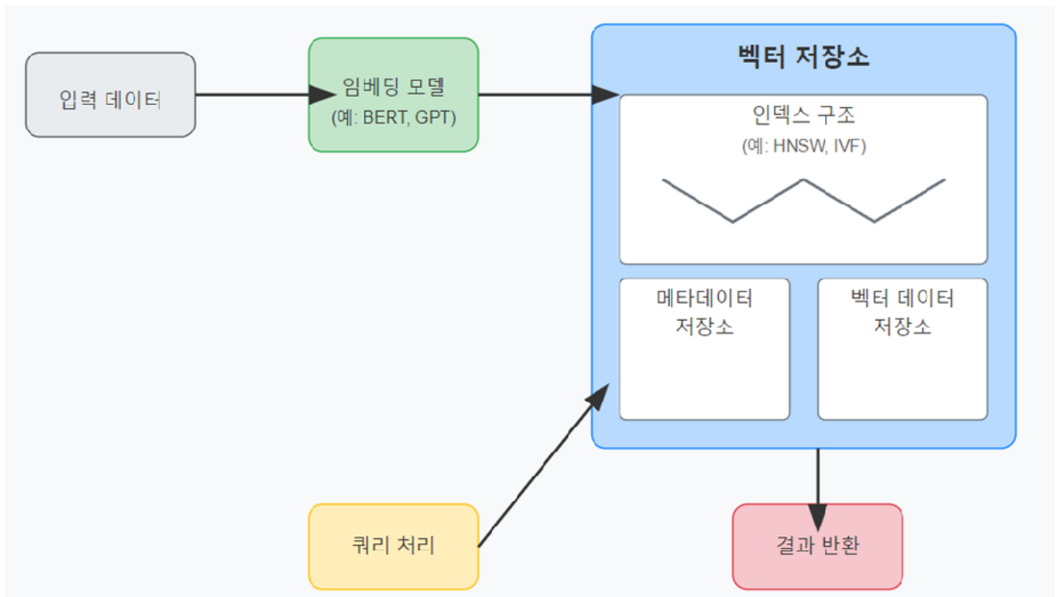
embed_query 메소드

- 기능: 단일 쿼리 텍스트를 임베딩
- 입력: 문자열
- 출력: 단일 임베딩 벡터

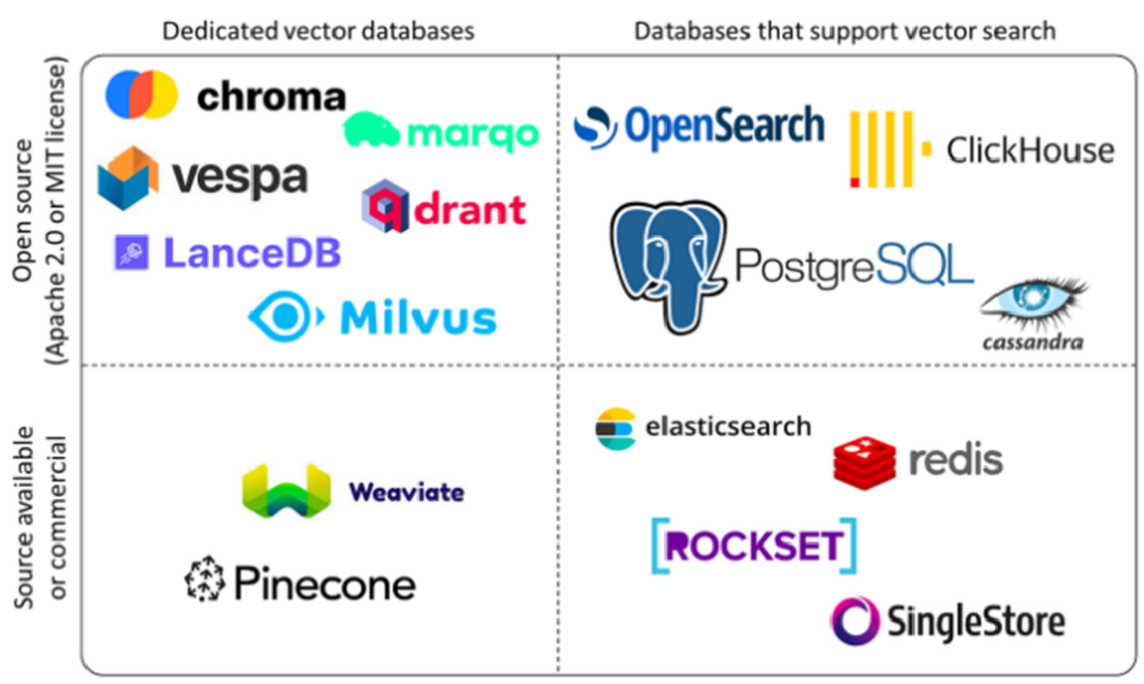
```python
query = "What is LangChain?"
query_vector = embeddings.embed_query(query)
```

3.4 벡터 저장소 (Vector Stores)

- 임베딩된 벡터를 저장하고 검색하는 데이터베이스
- 기본 개념:
    - 비정형 데이터(텍스트 등)를 벡터로 임베딩하여 저장
    - 입력된 쿼리를 벡터로 임베딩하여, 가장 '유사한' 임베딩 벡터를 검색



- Chroma, FAISS, Pinecone 등 다양한 옵션 제공



이미지 출처: https://blog.gopenai.com/high-level-comparison-of-information-retrieval-tools-chroma-faiss-pinecone-and-28694631237a

1. Chroma

- 사용자 편의성이 우수한 오픈소스 벡터 저장소
- `langchain-chroma` 패키지로 설치 가능

2. FAISS(Facebook AI Similarity Search)

- 효율적인 벡터 유사도 검색 및 클러스터링을 위한 오픈소스 벡터 저장소
- `faiss-cpu` 패키지로 설치 (GPU 버전은 `faiss-gpu`)
- 주요 특징:
    - 대규모 벡터 세트에서 효율적인 검색
    - 대용량 데이터셋 처리 (RAM 효율적 활용)
    - GPU 가속 지원
    - 다양한 인덱싱 알고리즘 제공 (속도와 정확도 조절 가능)

3.5 검색기 (Retrievers)

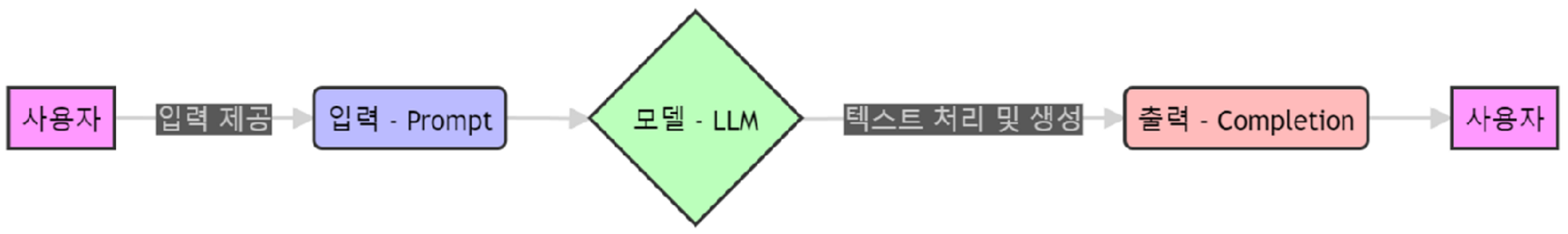
- 질의에 관련된 문서를 검색하는 컴포넌트

```
vectorstore = Chroma.from_documents(texts, embeddings)
print(f"벡터 저장소에 저장된 문서 수: {vectorstore._collection.count()}")
```

```
retriever = vectorstore.as_retriever(search_kwargs={"k": 2})
query = "What is the main topic of the document?"
relevant_docs = retriever.get_relevant_documents(query)
print(f"검색된 관련 문서 수: {len(relevant_docs)}")
print(f"첫 번째 관련 문서 내용 미리보기: {relevant_docs[0].page_content[:100]}...")
```

3.6 언어 모델 (LLMs)

- 텍스트 생성을 담당하는 대규모 언어 모델
- OpenAI의 GPT 모델 외에도 다양한 모델 지원




```
llm = OpenAI(temperature=0)
response = llm("Tell me a brief joke about programming.")
print(f"LLM 응답: {response}")
```

4. 간단한 RAG 파이프라인 구현 실습

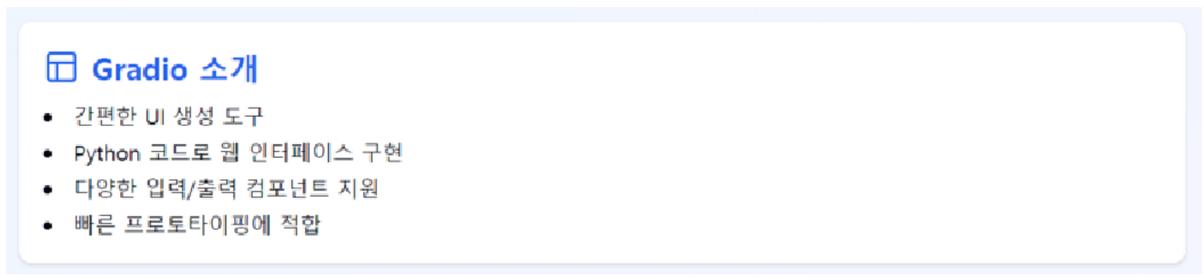
4.1 create_stuff_documents_chain

- 검색된 문서들을 하나의 컨텍스트로 결합하고, 이를 바탕으로 답변하는 체인을 생성

4.2 create_retrieval_chain

- 질문에 관련된 문서를 검색하고, 최공 답변을 생성하는 전체 RAG 파이프라인을 구축

5. Gradio 챗봇



- 예제 코드:

```python
import gradio as gr

def answer_invoke(message, history):
    response = rag_chain.invoke({"input": message})
    return response["answer"]


# Graiio 인터페이스 생성
demo = gr.ChatInterface(fn=answer_invoke, title="QA Bot")

# Graiio 실행
demo.launch()
```

- 실행 화면: