Google의 NotebookLM처럼 \*\*다수의 문서를 학습하고 질의응답(QA)을 가능하게 하는 에이전트\*\*를 무료로 구현하는 방법을 단계별로 설명드리겠습니다. 이 시스템은 \*\*RAG(Retrieval-Augmented Generation)\*\* 아키텍처를 기반으로 합니다.

---

### 1. 핵심 아키텍처: RAG (검색 증강 생성)

```

사용자 질문 → 문서 검색 → 관련 내용 추출 → LLM이 답변 생성

```

- \*\*장점\*\*:

- 최신 정보 활용 가능

- 환각(Hallucination) 감소

- 문서 출처 추적 가능

---

### 2. 무료 기술 스택

| 구분 | 도구/라이브러리 | 설명 |

|------------------|--------------------------|-------------------------------|

| 문서 처리 | `LangChain`, `Unstructured` | PDF, Word, 텍스트 등 다양한 형식 지원 |

| 임베딩 | `SentenceTransformers` | 무료 오픈소스 임베딩 모델 (예: `all-MiniLM-L6-v2`) |

| 벡터 저장소 | `ChromaDB`, `FAISS` | 임베딩 벡터 저장 및 유사도 검색 |

| LLM 연동 | `Hugging Face Transformers` | 로컬 LLM (예: `Flan-T5`, `Llama 2`) |

| RAG 파이프라인 | `LangChain` | 전체 워크플로우 통합 |

---

### 3. 단계별 구현 방법

#### Step 1: 환경 설정

```bash

pip install langchain sentence-transformers chromadb unstructured[all] transformers

```

#### Step 2: 문서 로드 및 전처리

```python

from langchain.document\_loaders import DirectoryLoader

from langchain.text\_splitter import RecursiveCharacterTextSplitter

# 디렉토리 내 모든 문서 로드 (PDF, TXT 등)

loader = DirectoryLoader('./documents', glob="\*\*/\*.pdf", show\_progress=True)

documents = loader.load()

# 텍스트 분할 (청크 크기: 1000, 중첩: 200)

text\_splitter = RecursiveCharacterTextSplitter(chunk\_size=1000, chunk\_overlap=200)

texts = text\_splitter.split\_documents(documents)

```

#### Step 3: 임베딩 및 벡터 저장소 생성

```python

from langchain.embeddings import HuggingFaceEmbeddings

from langchain.vectorstores import Chroma

# 무료 임베딩 모델 로드

model\_name = "sentence-transformers/all-MiniLM-L6-v2"

embeddings = HuggingFaceEmbeddings(model\_name=model\_name)

# ChromaDB에 임베딩 저장

vectorstore = Chroma.from\_documents(texts, embeddings, persist\_directory="./db")

vectorstore.persist() # DB 저장

```

#### Step 4: 검색 기반 QA 체인 구성

```python

from langchain.chains import RetrievalQA

from langchain.llms import HuggingFaceEndpoint

# Hugging Face의 무료 LLM (예: Flan-T5)

# Note: Hugging Face Hub 토큰 필요 (무료 계정 발급 가능)

llm = HuggingFaceEndpoint(

repo\_id="google/flan-t5-large",

task="text2text-generation",

max\_length=512

)

# 검색기 설정 (유사도 상위 3개 문서 검색)

retriever = vectorstore.as\_retriever(search\_kwargs={"k": 3})

# RAG 체인 생성

qa\_chain = RetrievalQA.from\_chain\_type(

llm=llm,

chain\_type="stuff", # 검색된 문서를 모두 컨텍스트로 사용

retriever=retriever,

return\_source\_documents=True

)

```

#### Step 5: 질의응답 실행

```python

query = "NotebookLM의 주요 기능은 무엇인가요?"

result = qa\_chain(query)

print("답변:", result['result'])

print("출처 문서:")

for doc in result['source\_documents']:

print(f"- {doc.metadata['source']} (페이지 {doc.metadata.get('page', 'N/A')})")

```

---

### 4. 고급 기능 추가

#### (1) 로컬 LLM 사용 (무료)

- \*\*Llama.cpp\*\* 활용:

```bash

# Llama 2 모델 다운로드 (Hugging Face에서 gguf 변환)

pip install llama-cpp-python

```

```python

from langchain.llms import LlamaCpp

llm = LlamaCpp(model\_path="./models/llama-2-7b-chat.Q4\_K\_M.gguf")

```

#### (2) 대화 기억 추가 (Memory)

```python

from langchain.memory import ConversationBufferMemory

from langchain.chains import ConversationalRetrievalChain

memory = ConversationBufferMemory(memory\_key="chat\_history", return\_messages=True)

qa\_chain = ConversationalRetrievalChain.from\_llm(

llm=llm,

retriever=retriever,

memory=memory

)

```

#### (3) 다중 문서 유형 지원

- `Unstructured` 라이브러리로 PDF, Word, Excel, HTML 등 처리:

```python

from langchain.document\_loaders import UnstructuredFileLoader

loader = UnstructuredFileLoader("document.docx", mode="elements")

```

#### (4) 평가 및 피드백

- \*\*RAGAS\*\* (평가 프레임워크):

```bash

pip install ragas

```

```python

from datasets import Dataset

from ragas import evaluate

from ragas.metrics import faithfulness, answer\_relevancy

# 평가 데이터 생성

data = {

"question": ["..."],

"answer": ["..."],

"contexts": [["..."]]

}

dataset = Dataset.from\_dict(data)

results = evaluate(dataset, metrics=[faithfulness, answer\_relevancy])

```

---

### 5. 무료 배포 옵션

1. \*\*Streamlit/Gradio 웹 인터페이스\*\*:

```python

import streamlit as st

st.title("문서 QA 시스템")

query = st.text\_input("질문을 입력하세요")

if query:

result = qa\_chain(query)

st.write(result['result'])

```

- 실행: `streamlit run app.py`

2. \*\*Hugging Face Spaces\*\*:

- 무료 호스팅 (Git 연동)

- 예시: [Hugging Face Spaces](https://huggingface.co/spaces)

3. \*\*로컬 실행\*\*:

- Jupyter Notebook 또는 Python 스크립트로 테스트

---

### 6. 성능 최적화 팁

- \*\*임베딩 모델 선택\*\*:

- `all-MiniLM-L6-v2` (빠름, 384차원)

- `all-mpnet-base-v2` (정확도 높음, 768차원)

- \*\*청크 크기 조정\*\*:

- 작은 청크: 정밀도 ↑, 컨텍스트 이해 ↓

- 큰 청크: 컨텍스트 이해 ↑, 노이즈 ↑

- \*\*검색 전략\*\*:

- `MMR` (Maximal Marginal Relevance): 다양성 있는 검색

```python

retriever = vectorstore.as\_retriever(search\_type="mmr", search\_kwargs={"k": 5, "fetch\_k": 20})

```

---

### 7. 주의사항

- \*\*모델 크기\*\*: 무료 LLM은 상용 모델(GPT-4)보다 성능이 낮을 수 있음

- \*\*메모리 관리\*\*: 대용량 문서 시 청크 단위 처리 필수

- \*\*개인정보\*\*: 민감한 문서는 로컬에서만 처리

- \*\*라이선스\*\*: 사용하는 모델의 라이선스 확인 (예: Llama 2는 상업적 사용 제한)

---

### 8. 학습 자료

- \*\*LangChain 문서\*\*: [python.langchain.com](https://python.langchain.com/)

- \*\*Hugging Face 임베딩 모델\*\*: [sbert.net](https://www.sbert.net/)

- \*\*RAG 튜토리얼\*\*: [LangChain RAG Tutorial](https://python.langchain.com/docs/use\_cases/question\_answering/)

이 구조를 기반으로 문서를 학습하고 질의응답하는 에이전트를 무료로 구축할 수 있습니다. 추가로 궁금한 점(예: 특정 문서 유형 처리, 고급 검색 기법 등)이 있다면 알려주세요!