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

---

`CH12 RAG` *Retrieval Augmented Generation*

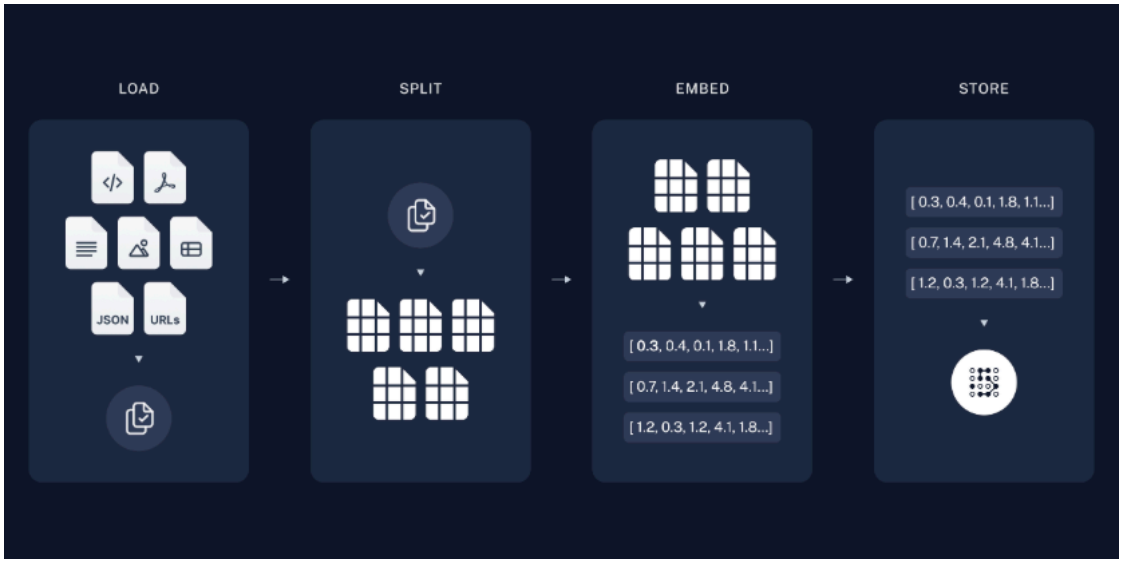
- **`RAG`**: `Retrieval-Augmented Generation`
  - 자연어 처리(`NLP`) 분야에서의 혁신적인 기술
  - 기존의 언어 모델의 한계를 넘어서 정보 검색과 생성을 통합하는 방법론

- 기본 역할:
  - 풍부한 정보를 담고 있는 대규모 문서 데이터베이스에서 관련 정보 검색
  - 언어 모델이 더 정확하고 상세한 답변을 생성할 수 있게 함
  - 예시: *`최신 뉴스 이벤트`*나 *`특정 분야의 전문 지식`*과 같은 주제에 대해 질문 → *`RAG`*는 관련 문서를 찾아 그 내용을 바탕으로 답변을 구성

---

1. `RAG의 8단계 프로세스`

1) `사전 준비단계`

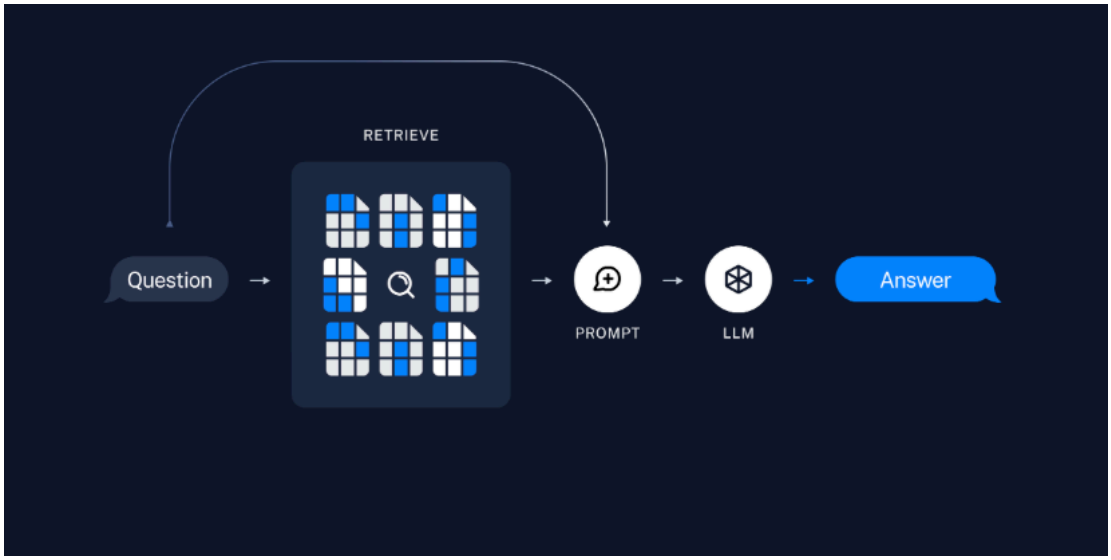
- `RAG`의 사전 준비단계
  - 

*

- a. **`도큐먼트 로드`** (**`Document Loader`**):
  - 외부 데이터 소스에서 **`필요한 문서를 로드`** → **`초기 처리`**
  - ≒ 마치 책을 여러 권 챙겨 도서관에서 공부하는 것과 비슷 / 학생이 공부하기 전에 필요한 책들을 책장에서 골라오는 과정
- b. **`텍스트 분할`** (`Text Splitter`):
  - **로드된 문서를 처리 가능한 `작은 단위`로 `분할`**
  - ≒ 큰 책을 챕터별로 나누는 것과 유사
- c. **`임베딩`** (`Embedding`):
  - 각 **`문서`** 또는 **`문서의 일부`**를 **`벡터`** **형태로** **`변환`** → **문서의 의미 수치화**
  - ≒ 책의 내용을 요약하여 핵심 키워드로 표현하는 것과 비슷
- d. **`벡터스토어`** (`Vector Store`) 저장:
  - **`임베딩된 벡터`**들을 **`데이터베이스`**에 **`저장`**

- ≒ 요약된 키워드를 색인화하여 나중에 빠르게 찾을 수 있도록 하는 과정

2) `런타임 (RunTime 단계)`

- `Runtime`
  - 

*

- e. **`검색기`** (`Retriever`):
  - **`질문`** → 이와 관련된 벡터를 **`벡터 데이터베이스`**에서 검색
  - ≒ 질문에 가장 잘 맞는 책의 챕터를 찾는 것과 유사
- f. **`프롬프트`** (`Prompt`):
  - **`검색된 정보`**를 **`바탕`**으로 언어 모델을 위한 **`질문`**을 **`구성`**
  - **`=`** 정보를 바탕으로 **`어떻게 질문 할지 결정`**하는 과정
- g. **`LLM`** (Large Language Model):
  - 구성된 `프롬프트` 사용 → **`언어 모델`**이 **`답변`**을 **`생성`**
  - ≒ 수집된 정보를 바탕으로 과제 or 보고서를 작성하는 학생과 같음
- h. **`체인`** (**`Chain`**) 생성:
  - 이전의 **`모든 과정`**의 **`하나의 파이프라인`**으로 묶어주는 **`체인`**(**`Chain`**)을 생성

---

3) `PDF 문서 기반 QA` *(Question-Answer)*

- **`RAG 기본 구조 이해하기 - 1`** = **`사전작업`** *(1~4단계)*
  -  사전 작업단계
  -  RAG 작업단계1

- 사전 작업 단계: 데이터 소스를 `Vector DB`에 문서를 **`로드`**-**`분할`**-**`임베딩`**-**`저장`** 하는 **`4단계`**를 진행
  - 1단계: **`문서로드`**(**`Document Load`**) = 문서 내용 불러오기
  - 2단계: **`분할`**(**`Text Split`**): 문서를 특정 기준(`Chunk`)으로 분할하기
  - 3단계: **`임베딩`**(**`Embedding`**): 분할된(`Chunk`) → **`임베딩`** → 저장하기
  - 4단계: **`벡터DB 저장`**: 임베딩된 `Chunk` → `DB`에 저장

- **`RAG - 2 = 수행`** *(RunTime)* - 5~8단계
  -  RAG2
  -  runtime단계

- 수행 단계
  - 5단계: **`검색기`**(**`Retriever`**)

        - 쿼리 (`Query`) → `DB` 에서 `검색` 하여 결과를 가져오기 위하여 `리트리버` 를 `정의` 하기
        - 리트리버 = 검색 알고리즘 / 아래 2개의 종류
            - **`Dense`** : **`유사도 기반 검색`**
            - **`Sparse`** : **`키워드 기반 검색`**
    - 6단계: **`프롬프트`**
        - **`RAG`** **를 수행하기 위한** **`프롬프트 생성`**
        - 프롬프트의 `context` = **`문서에서 검색된 내용`**
        - 프롬프트 엔지니어링 → `답변의 형식` 을 `지정 가능`
    - 7단계: **`LLM`**
        - **`모델`** **정의하기**
        - GPT-3.5, GPT-4, Claude, gemini-2.5-flash, ...
    - 8단계: **`Chain`** = **`프롬프트`** **-** **`LLM`** **-** **`출력`** **에 이르는 체인을 생성**

- **`실습 활용 문서`**
    - `소프트웨어정책연구소(SPRi) - 2023년 12월호`
        - 저자: 유재흥(AI정책연구실 책임연구원), 이지수(AI정책연구실 위촉연구원)
        - 링크: https://spri.kr/posts/view/23669
        - 파일명: `SPRI_AI_Brief_2023년12월호_F.pdf`

- **`환경설정`**

```
# 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 않음" # API 키 값은 직접 출력하지 않음

if langchain_tracing_v2 == "true" and os.getenv('LANGCHAIN_API_KEY') and langchain_project:
    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: '{langchain_tracing_v2}').")
    if not os.getenv('LANGCHAIN_API_KEY'):
        print("  - LANGCHAIN_API_KEY가 설정되어 있지 않습니다.")
    if not langchain_project:
        print("  - LANGCHAIN_PROJECT가 설정되어 있지 않습니다.")
```

- 셀 출력

```
--- LangSmith 환경 변수 확인 ---
✅ LangSmith 추적 활성화됨 (LANGCHAIN_TRACING_V2='true')
✅ LangSmith 프로젝트: 'LangChain-prantice'
✅ LangSmith API Key: 설정됨
-> 이제 LangSmith 대시보드에서 이 프로젝트를 확인해 보세요.
```

---

3) `RAG 기본 파이프라인` *(1~8단계)*

```
from langchain_text_splitters import RecursiveCharacterTextSplitter
from langchain_community.document_loaders import PyMuPDFLoader
from langchain_community.vectorstores import FAISS
from langchain_core.output_parsers import StrOutputParser
from langchain_core.runnables import RunnablePassthrough
from langchain_core.prompts import PromptTemplate
from langchain_google_genai import ChatGoogleGenerativeAI
from langchain_huggingface import HuggingFaceEmbeddings
```

```
# 단계 1: 문서 로드(Load Documents)

loader = PyMuPDFLoader("../12_RAG/data/SPRI_AI_Brief_2023년12월호_F.pdf")
docs = loader.load()
print(f"문서의 페이지수: {len(docs)}")
```

- 문서의 페이지수: 23 (`0.9s`)

```
# 단계 2: 문서 분할(Split Documents)

text_splitter = RecursiveCharacterTextSplitter(chunk_size=1000, chunk_overlap=50)
split_documents = text_splitter.split_documents(docs)
print(f"분할된 청크의수: {len(split_documents)}")
```

- 분할된 청크의수: 43 (`0.0s`)

```
# 단계 3: 임베딩(Embedding) 생성
from langchain_huggingface import HuggingFaceEmbeddings
import warnings

# 경고 무시
warnings.filterwarnings("ignore")

# HuggingFace Embeddings 사용
embeddings = HuggingFaceEmbeddings(
    model_name="sentence-transformers/all-MiniLM-L6-v2",
    model_kwargs={'device': 'cpu'},
    encode_kwargs={'normalize_embeddings': True}
)

print("✅ hugging-face 임베딩 모델 로딩 완료!")
```

- ✅ hugging-face 임베딩 모델 로딩 완료! (`7.2s`)

```
# 단계 4: DB 생성(Create DB) 및 저장

# 벡터스토어 생성하기
vectorstore = FAISS.from_documents(
    documents=split_documents,
    embedding=embeddings
)

print("🎉 Hugginface model을 사용하여 로컬에서 벡터스토어(FAISS)가 성공적으로 생성되었습니다!")
```

- 🎉 Hugginface model을 사용하여 로컬에서 벡터스토어(FAISS)가 성공적으로 생성되었습니다! (`1.0s`)

```
# 단계 5: 검색기(Retriever) 생성
# 문서에 포함되어 있는 정보를 검색하고 생성하기
retriever = vectorstore.as_retriever()
```

```
# 검색기에 쿼리를 날려 검색된 chunk 결과 확인하기
import os
os.environ["TOKENIZERS_PARALLELISM"] = "false"      # 병렬처리 비활성화

# 검색기에 쿼리를 날려 검색된 chunk 결과 확인하기
retriever.invoke("삼성전자가 자체 개발한 AI 의 이름은?")
```

- `chunk` 확인하기

```
[Document(id='e84c1482-442c-44e2-8c6d-7a51702ce0e0', metadata={'producer': 'Hancom PDF 1.3.0.542', 'creator': 'Hw
Document(id='e0ce8fed-10cb-4cb4-bc8c-478a90bfbdf3', metadata={'producer': 'Hancom PDF 1.3.0.542', 'creator': 'Hwp
```

```
Document(id='cf8f1453-b998-43ac-9cb0-2d18a758cbae', metadata={'producer': 'Hancom PDF 1.3.0.542', 'creator': 'Hwp
Document(id='b9f35d59-9547-4da9-911c-2eba1898dd5a', metadata={'producer': 'Hancom PDF 1.3.0.542', 'creator': 'Hwp
```

```
# 단계 6: 프롬프트 생성(Create Prompt)

# 프롬프트 생성하기

prompt = PromptTemplate.from_template(
    """You are an assistant for question-answering tasks.
Use the following pieces of retrieved context to answer the question.
If you don't know the answer, just say that you don't know.
Answer in Korean.

#Question:
{question}
#Context:
{context}

#Answer:"""
)
```

```
print(prompt)
```

- `print(prompt)` 확인

```
input_variables=['context', 'question'] input_types={} partial_variables={} template="You are an assistant for qu
```

```
print(prompt.template)
```

- `print(prompt.template)` 확인하기 = *프롬프트 내용만 확인하기*

```
You are an assistant for question-answering tasks.
Use the following pieces of retrieved context to answer the question.
If you don't know the answer, just say that you don't know.
Answer in Korean.

#Question:
{question}
#Context:
{context}

#Answer:
```

```
# 단계 7: 언어모델(LLM) 생성
# 모델(LLM) 생성하기
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 = 0으로 설정
        max_output_tokens=4096,
    )
```

- `LLM` 생성하기

```
E0000 00:00:1759658522.522070  683443 alts_credentials.cc:93] ALTS creds ignored. Not running on GCP and untruste
```

```
# 단계 8: 체인(Chain) 생성
```

```
chain = (
    {"context": retriever, "question": RunnablePassthrough()}
    | prompt
    | gemini_lc
    | StrOutputParser()
)
```

- 생성된 체인에 **`쿼리`** (`질문`)을 입력 → 실행하기

```
# 체인 실행(Run Chain)
# 문서에 대한 질의를 입력하고, 답변을 출력하기
question = "삼성전자가 자체 개발한 AI 의 이름은?"
response = chain.invoke(question)
print(response)
```

- 삼성전자가 자체 개발한 AI의 이름은 '삼성 가우스'입니다. (`1.2s`)

- next: ***`02. 네이버 뉴스기사 QA (Question-Answer)`***
