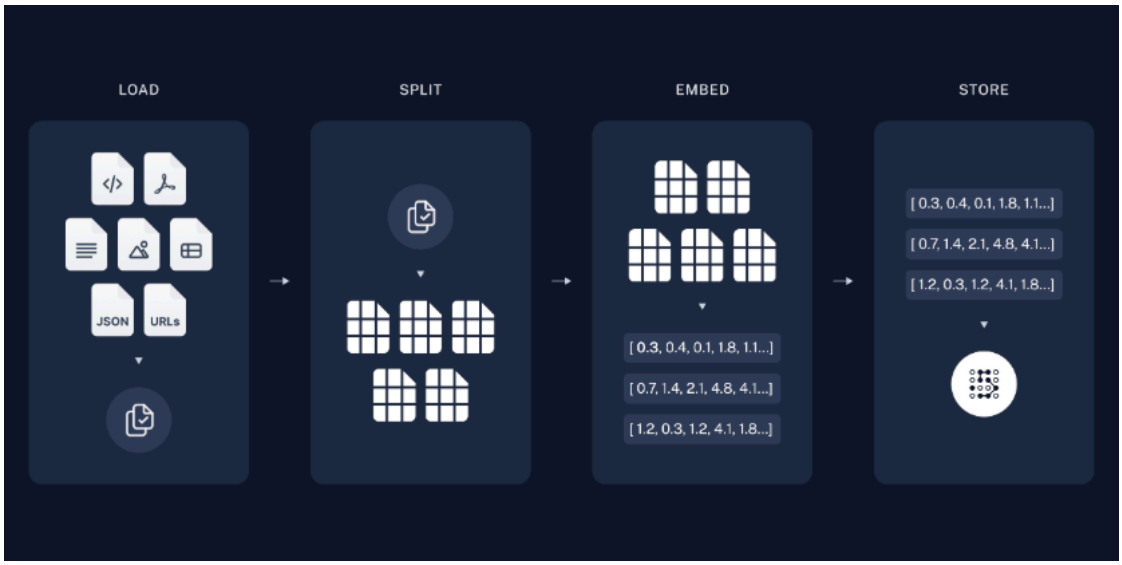

- 출처: LangChain 공식 문서 또는 해당 교재명
- 원본 URL: https://smith.langchain.com/hub/teddynote/summary-stuff-documents

---

1. RAG의 8단계 프로세스

1) 사전 준비단계

- RAG의 사전 준비단계
  - 

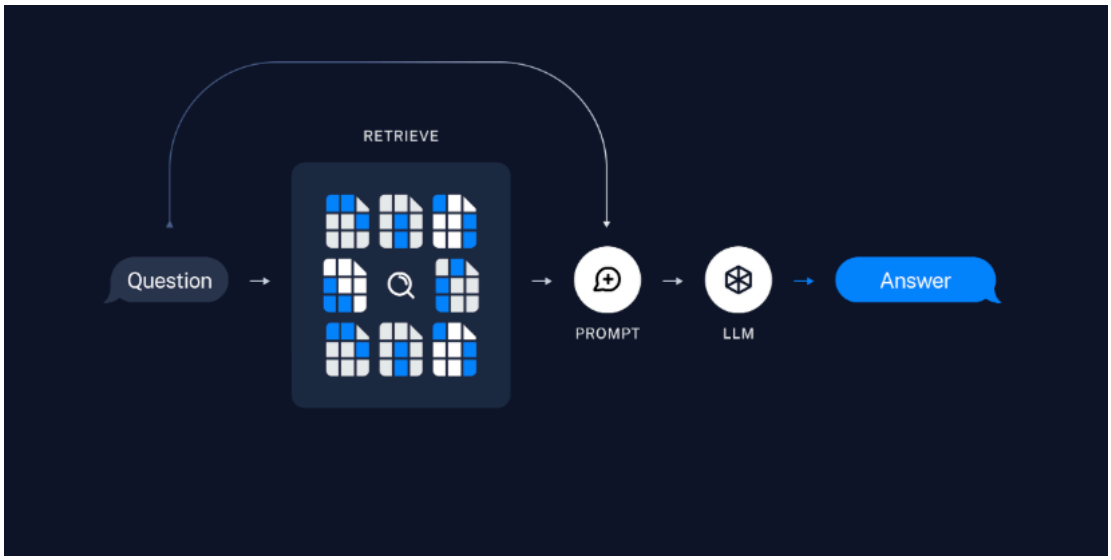
*

- 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 단위로 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`

✦



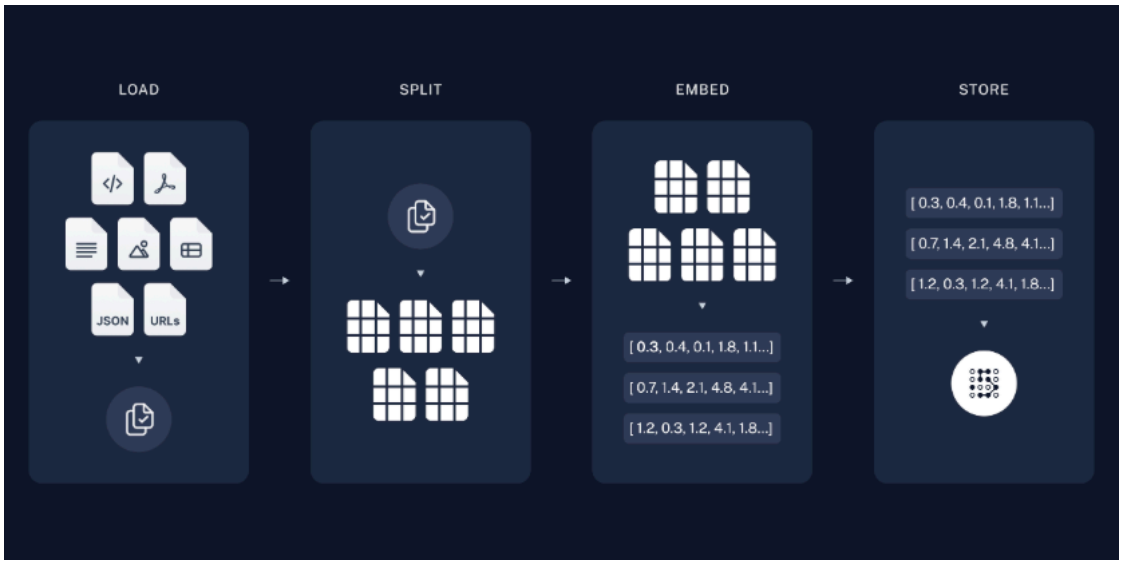
*

- 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

---

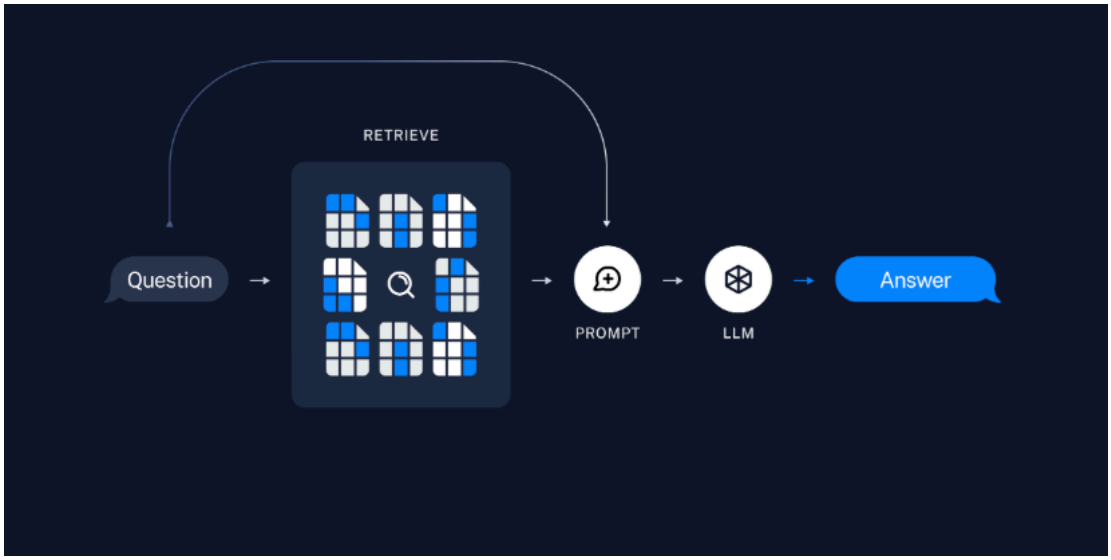
3) `PDF 문서 기반 QA` *(Question-Answer)*

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



- 사전 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
  

- 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

---

2. `네이버 뉴스 기반 QA` *(Question-Answering)* `챗봇`

- `뉴스기사 QA 앱 구축해보기`
  - 네이버 뉴스기사의 내용에 대해 질문할 수 있는 뉴스기사 QA 앱을 구축
  - `Chroma 벡터 스토어` 를 사용할 것

- 간단한 `인덱싱 파이프라인` 과 `RAG 체인` 을 약 20줄의 코드로 구현 가능

- 라이브러리 설명
  - `bs4` = 웹 페이지를 파싱하기 위한 라이브러리
  - `langchain`
    - `AI` 와 관련된 다양한 기능을 제공하는 라이브러리
    - `텍스트 분할` (RecursiveCharacterTextSplitter), `문서 로딩` (WebBaseLoader), `벡터 저장` (Chroma, FAISS), `출력 파싱` (StrOutputParser), `실행 가능한 패스스루` (RunnablePassthrough) 등을 사용할 예정

- `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.
```

```
import bs4
from langchain import hub
from langchain_text_splitters import RecursiveCharacterTextSplitter
from langchain_community.document_loaders import WebBaseLoader
from langchain_community.vectorstores import FAISS
from langchain_core.output_parsers import StrOutputParser
from langchain_core.runnables import RunnablePassthrough
from langchain_google_genai import ChatGoogleGenerativeAI
from langchain_huggingface import HuggingFaceEmbeddings
```

- USER_AGENT environment variable not set, consider setting it to identify your requests. (`3.7s`)

- `웹 페이지`의 내용을 `로드` → `텍스트`를 `청크`로 나눔 → `인덱싱` → `관련`된 텍스트 `스니펫 검색` → `새로운 내용`을 `생성`하는 `과정 구현`하기
- **`WebBaseLoader`** = 지정된 웹 페이지에서 필요한 부분만 파싱하기 위해 **`bs4.SoupStrainer`** 사용
  - *참고: `bs4.SoupStaraine`r = 편리하게 웹에서 원하는 요소를 가져올 수 있도록 해줌*

```
  bs4.SoupStrainer(
      "div",
      attrs={"class": ["newsct_article _article_body", "media_end_head_title"]},
  )
```

```
# 뉴스기사 내용 로드 → 청크로 나눔 → 인덱싱하기

loader = WebBaseLoader(
    web_paths=("https://n.news.naver.com/article/437/0000378416",),
    bs_kwargs=dict(
        parse_only=bs4.SoupStrainer(
            "div",
            attrs={"class": ["newsct_article _article_body", "media_end_head_title"]},
        )
    ),
)

docs = loader.load()
print(f"문서의 수: {len(docs)}")
docs
```

- 문서의 수: 1 (`0.4s`)

```
[Document(metadata={'source': 'https://n.news.naver.com/article/437/0000378416'}, page_content="\n출산 직원에게 '1억원
```

- **`RecursiveCharacterTextSplitter`** = 문서를 지정된 크기의 청크로 나누기

```
text_splitter = RecursiveCharacterTextSplitter(chunk_size=1000, chunk_overlap=100)

splits = text_splitter.split_documents(docs)
len(splits)                                                    # 3
print(f"분할된 문서의 수: {len(splits)}")
```

- 분할된 문서의 수: 3

- **`vectorstore`** (`FAISS` 혹은 `Chroma`): 이러한 청크를 바탕으로 문서의 벡터 표현을 생성함

```
# 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! (`6.2s`)

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

# 뉴스에 포함되어 있는 정보를 검색하고 생성하기
retriever = vectorstore.as_retriever()                                # 0.2s
```

- **`vectorstore.as_retriever()`** 통해 생성된 검색기 → **`hub.pull`로 가져온 `프롬프트`** + **`LLM`** 으로 새로운 내용 생성하기

```
from langchain_core.prompts import PromptTemplate

prompt = PromptTemplate.from_template(
    """당신은 질문-답변(Question-Answering)을 수행하는 친절한 AI 어시스턴트입니다. 당신의 임무는 주어진 문맥(context) 에서 주어진 질문(question) 에 답하
검색된 다음 문맥(context) 을 사용하여 질문(question) 에 답하세요. 만약, 주어진 문맥(context) 에서 답을 찾을 수 없다면, 답을 모른다면 `주어진 정보에서 질문에
한글로 답변해 주세요. 단, 기술적인 용어나 이름은 번역하지 않고 그대로 사용해 주세요.

#Question:
{question}

#Context:
{context}

#Answer:"""
)
```

```
print(prompt)
```

- `print(prompt)` 확인하기

```
input_variables=['context', 'question'] input_types={} partial_variables={} template='당신은 질문-답변(Question-Answe
```

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

- `print(prompt.template)` → *프롬프트의 내용만 확인하기*

```
당신은 질문-답변(Question-Answering)을 수행하는 친절한 AI 어시스턴트입니다. 당신의 임무는 주어진 문맥(context) 에서 주어진 질문(question
검색된 다음 문맥(context) 을 사용하여 질문(question) 에 답하세요. 만약, 주어진 문맥(context) 에서 답을 찾을 수 없다면, 답을 모른다면 `주어
한글로 답변해 주세요. 단, 기술적인 용어나 이름은 번역하지 않고 그대로 사용해 주세요.

#Question:
{question}

#Context:
{context}

#Answer:
```

```
# 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,
    )
```

- **`Chain` 생성하기**

```
# 체인 생성하기
rag_chain = (
    {"context": retriever, "question": RunnablePassthrough()}
    | prompt
    | gemini_lc
    | StrOutputParser()
)
```

- **`스트리밍 출력 함수`** (프롬프트 템플릿 적용, LLM 사용) → 응답 생성해보기
  - `LangChain`의 응답을 스트리밍으로 출력하는 기능
  - `LLM`이 답변을 생성하는 동안에도 실시간으로 응답을 출력해 사용자가 답변을 기다리는 시간을 줄여주는 기능 → 사용자 경험을 향상시키기 위해 사용됨

```
# 스트리밍 출력 함수
def stream_response(question, rag_chain):
    # 체인을 실행하고 스트리밍 방식으로 응답 생성
    for chunk in rag_chain.stream(question):
        print(chunk, end="", flush=True)
```

```
# 사용 예시_1
question_1 = "부영그룹의 출산 장려 정책에 대해 설명해주세요."
stream_response(question_1, rag_chain)
```

- 부영그룹은 2021년 이후 태어난 직원 자녀에게 1억원씩 총 70억원을 지원하는 파격적인 출산 장려 정책을 시행하고 있습니다. 또한, 셋째 자녀를 출산하는 직원에게는 국민주택을 제공하겠다는 계획도 밝혔습니다. (`1.3s`)

```
# 사용 예시_2
question_2 = "부영그룹은 출산 직원에게 얼마의 지원을 제공하나요?"
stream_response(question_2, rag_chain)
```

- 부영그룹은 2021년 이후 태어난 직원 자녀에게 1억원을 지원합니다. 연년생이나 쌍둥이 자녀의 경우 총 2억원을 받을 수 있으며, 셋째를 낳으면 국민주택을 제공하겠다는 계획도 밝혔습니다. (`0.8s`)

```
# 사용 예시_3
question_3 = "정부의 저출생 대책을 bullet points 형식으로 작성해 주세요."
stream_response(question_3, rag_chain)
```

- 3번째 질문 (`1.2s`)

```
정부의 저출생 대책은 다음과 같습니다.

*   매달 부모 급여 지급 (0세 아이는 100만원으로 인상)
*   첫만남이용권 지급
*   아동수당 지급 (아이 돌까지 1년 동안 1520만원 지원)
```

```
# 사용 예시_4
question_4 = "부영그룹의 임직원 숫자는 몇명인가요?"
stream_response(question_4, rag_chain)
```

```
stream_response(question_4, rag_chain)
```

- 주어진 정보에서 질문에 대한 정보를 찾을 수 없습니다 (`0.9s`)

- next: ***`03. RAG 의 기능별 다양한 모듈 활용기`***