Ch_12 Retrieval Augmented Generation(RAG)

RAG

> 자연어 처리(NLP) 분야에서의 혁신적인 기술,
> 기존의 언어 모델의 한계를 넘어서 정보 검색과 생성을 통합하는 방법론

- 대규모 문서 데이터베이스에서 관련 정보를 검색
    - 모델이 더 정확하고 상세한 답변을 생성할 수 있음

RAG의 8단계 프로세스

[사전 준비단계]

1. **Document Loader**

: 외부 데이터 소스에서 필요한 문서를 로드하고 초기 처리함 / 작업을 하기 전 사전에 필요한 준비물 준비

2. **Text Splitter**

: 로드된 문서를 처리 가능한 작은 단위로 분할함 / 큰 책을 챕터별로 분할

3. **Embedding**

: 각 문서 또는 문서의 일부를 벡터 형태로 변환 → 문서의 의미를 수치화 / 문서의 내용을 요약하여 핵심 키워드로 표현

4. **Vector Store 저장**

: 임베딩된 벡터들을 데이터베이스에 저장함. / 요약된 키워드를 색인하여 나중에 빠르게 찾기 위함

[RunTime 단계]

5. **Retriever(검색기)**

: 질문이 주어지면 → 이와 관련된 벡터를 벡터 데이터베이스에서 검색함. / 질문에 가장 잘 맞는 챕터 찾기

6. **Prompt**

: 검색된 정보를 바탕으로 언어 모델을 위한 질문을 구성함. / 정보를 바탕으로 어떻게 질문할지 결정함

7. **LLM(Large Language Model)**

: 구성된 프롬프트를 사용하여 언어 모델이 답변을 생성함. / 수집된 정보를 바탕이나 보고서를 작성~

8. **Chain 생성**

: 이전의 모든 과정을 하나의 파이프라인으로 묶어주는 체인을 생성함.

I. PDF 문서 기반 QA(Question-Answer) vs 네이버 뉴스기사 QA(Question-Answer)

공통점

: RAG의 1~8단계 파이프라인

단계	내용
1~4	문서 로드 → 분할 → 임베딩 → 벡터DB 저장
5	Retriever로 관련 문서 검색
6	검색 결과를 Context로 프롬프트 구성
7	LLM 호출
8	Chain으로 연결

차이점

i) 문서 로드 방식

`<PDF>`

```
# 단계 1: 문서 로드(Load Documents)
loader = PyMuPDFLoader("data/SPRI_AI_Brief_2023년12월호_F.pdf")
docs = loader.load()
print(f"문서의 페이지수: {len(docs)}")
```

<네이버 뉴스>

```
# 뉴스기사 내용을 로드하고, 청크로 나누고, 인덱싱합니다.
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
```

구분	PDF 문서 QA	네이버 뉴스 QA
Loader	`PyMuPDFLoader`	`WebBaseLoader`
데이터	로컬 파일(PDF)	웹 페이지(HTML)
특징	구조가 안정적	HTML 구조에 의존

PDF

- 학술자료, 보고서, 사내 문서에 적합
- 페이지 단위로 깔끔하게 로딩됨

뉴스

- 실시간/외부 데이터
- `bs4.SoupStrainer`로 **필요한 HTML 부분만 추출**해야 함

ii) 데이터 성격

구분	PDF	뉴스
성격	정적(static)	동적(dynamic)

구분	PDF	뉴스
업데이트	거의 없음	자주 바뀜
신뢰성	높음	기사마다 다름

PDF

- 정책 보고서
- 논문
- 매뉴얼

뉴스

- 시사 이슈
- 사건 요약
- 트렌드

iii) 분할 전략 차이

항목	PDF	뉴스
Chunk 기준	문단/페이지 중심	기사 흐름 중심
Overlap	작아도 됨	조금 크게 주는 게 안정적

⇒ 뉴스는 문맥이 짧아서 `chunk_overlap=100`처럼 겹침을 더 주는 게 좋음

iv) 프롬프트 설계 차이

PDF QA 프롬프트

- 사실 기반 질문
- “OO는 무엇인가?”
- “OO의 정의는?”

뉴스 QA 프롬프트

- 요약, 설명, bullet points
- “정책을 설명해줘”

- “핵심 내용 정리해줘”

그래서 뉴스 쪽 프롬프트는 보통

- 요약 지시
- bullet points
- “정보 없으면 없다고 말해라” 강조가 들어감

II. RAG 의 기능별 다양한 모듈 활용기

i) RAG 전체 구조 요약

- **Indexing (사전 작업)**

: 데이터 로드 → 분할 → 임베딩 → 벡터스토어 저장

- **Retrieval & Generation (실행 시점)**

: 검색(Retriever) → 프롬프트 구성 → LLM 답변 생성

ii) 문서 로드

- Web: `WebBaseLoader` + `bs4.SoupStrainer`
- PDF: `PyPDFLoader`
- CSV: `CSVLoader`
- TXT: `TextLoader`
- 폴더 전체: `DirectoryLoader`
- Python 코드: `PythonLoader`

iii) 문서 분할 (Text Split)

- **CharacterTextSplitter**
  - 문자 기준 분할
  - 간단하지만 문맥 깨질 수 있음

- **RecursiveCharacterTextSplitter (권장)**
  - 문단 → 문장 → 단어 순으로 분할
  - 의미 보존에 유리
- **SemanticChunker**
  - 임베딩 유사도 기반 분할
  - 가장 자연스럽지만 비용 ↑

iv) 임베딩 (Embedding)

- **OpenAI (유료)**
  - `text-embedding-3-small / large`
- **오픈소스 (무료)**
  - `HuggingFaceBgeEmbeddings`
  - `FastEmbedEmbeddings`

⇒ 비용 vs 성능 트레이드오프 고려

v) 벡터스토어 (VectorStore)

- **FAISS**
  - 빠르고 로컬 실습에 적합
- **Chroma**
  - 영속 저장, 프로덕션 친화적

vi) Retriever (검색 전략)

- **Similarity:** 의미 유사도 검색 (기본)
- **Similarity + Threshold:** 신뢰도 높은 결과만
- **MMR:** 중복 줄이고 다양성 확보
- **MultiQueryRetriever:** 질문을 여러 개로 확장
- **BM25:** 키워드 기반 (Sparse)

- **EnsembleRetriever:**

  BM25 + FAISS 결합 (가장 안정적)

검색이 안 맞으면 → **Retriever 수정**

답변 형태가 이상하면 → **Prompt 수정**

vii) 프롬프트 (Prompt)

- Hub 활용 가능: `hub.pull("rlm/rag-prompt")`
- Context 기반 답변 강제
- 정보 없으면 "모른다" 명시하도록 설계

viii) LLM (모델)

- OpenAI: `gpt-3.5-turbo`, `gpt-4`
- HuggingFace: `flan-t5-xxl` 등
- 토큰/비용 확인 가능 (`get_openai_callback`)

ix) 체인 (Chain)

```
Question
 → Retriever
 → Context
 → Prompt
 → LLM
 → OutputParser
```

LangChain의 핵심은 **모듈 조합 + 교체 가능성**

III. RAPTOR: 긴 문맥 요약(Long Context Summary)

구분	RAG	RAPTOR
목적	질문에 답변	**긴 문서를 이해·요약·검색**
처리 단위	문서 청크	**문서 → 요약 → 요약의 요약 (트리)**
구조	평면 (flat)	**트리 구조 (hierarchical)**
검색 대상	원문 청크	**원문 + 중간 요약 + 최상위 요약**
강점	빠르고 단순	**긴 문맥, 전체 구조 파악에 강함**
단점	긴 문서에서 맥락 약함	**구현 복잡, 비용↑**

RAG 다시 쉽게 말하면...

```
문서 → 쪼갬 → 임베딩 → DB 저장
질문 → 관련 청크 검색 → LLM이 답변
```

특징

- **"필요한 부분만"** 가져옴
- 전체 맥락 요약 X
- 긴 문서에선 놓치는 정보 많음

RAPTOR

1. 시작: leaf (원문)

- 문서 / 청크 / 페이지 하나하나

2. 묶기

- 임베딩 → 비슷한 것끼리 클러스터링

3. 요약

- 클러스터마다 요약 생성

4. 재귀 반복

- 요약들을 다시 묶어서

  👉 요약의 요약

👉 요약의 요약의 요약

결과: 트리 구조

```
최상위 요약 (문서 전체 의미)
 ├ 중간 요약A
 │   ├ 원문1
 │   └ 원문2
 └ 중간 요약B
     ├ 원문3
     └ 원문4
```

“Collapsed Tree Retrieval”

> 트리를 다시 평탄화(flat) 함

```
원문 + 중간 요약 + 최상위 요약
→ 전부 벡터DB에 넣고
→ 한 번에 kNN 검색
```

왜 좋음?

- 질문이 추상적이면 → 상위 요약이 걸림
- 질문이 구체적이면 → 원문이 걸림

  → 레벨 자동 선택 효과

그래서 언제 뭐 쓰냐면

RAG 쓰면 좋은 경우

- FAQ
- 짧은 문서
- 고객 응대 챗봇
- 빠른 구현 필요

RAPTOR 쓰면 좋은 경우

- 논문 / 매뉴얼 / 법률 문서
- 위키, 기술 문서 묶음

- 전체 핵심 요약 질문
- 이 문서가 뭘 말하는지 질문

> **RAG = 검색 중심**
>
> **RAPTOR = 이해 + 요약 중심**

IV) 대화내용을 기억하는 RAG 체인

> `RunnableWithMessageHistory`를 사용하면 RAG 체인이 이전 대화를 기억하고 이어서 답변할 수 있음.

i) 사용 목적

일반 RAG는 매 질문을 독립적으로 처리함

→ “방금 말한 거 번역해줘”, “그거 다시 설명해줘” X

→ 대화형 QA를 만들려면 이전 질문/답변을 프롬프트에 함께 전달해야 함

ii) RunnableWithMessageHistory

- 세션별 대화 기록 관리
- ChatMessageHistory에 자동 저장
- `session_id` 기준으로 대화 분리

패턴 1) 일반 Chain + 대화 기억

구조

```
Prompt (chat_history 포함)
 → LLM
 → Output
```

핵심 포인트

- `MessagesPlaceholder("chat_history")` 필수
- 세션 ID로 대화 유지

동작 예

```
Q1: 나의 이름은 테디입니다.
A1: 안녕하세요, 테디님.

Q2: 내 이름이 뭐라고?
A2: 당신의 이름은 테디입니다.
```

패턴 2) RAG + 대화 기억 (실전용)

구조

```
Question
 → Retriever
 → Context
 → Prompt (chat_history + context)
 → LLM
 → Answer
```

프롬프트 핵심

```
#Previous Chat History:
{chat_history}

#Question:
{question}

#Context:
{context}
```

---

iii) 구현 핵심 흐름

RAG 체인 생성

- 문서 로드
- 분할
- 임베딩

- 벡터DB
- Retriever

RunnableWithMessageHistory로 감싸기

```
rag_with_history = RunnableWithMessageHistory(
    chain,
    get_session_history,
    input_messages_key="question",
    history_messages_key="chat_history",
)
```

session_id로 대화 유지

```
config={"configurable": {"session_id":"rag123"}}
```

---

iv) 실제 동작 예시

```
Q1: 삼성전자가 만든 생성형 AI 이름은?
A1: 삼성 가우스입니다.

Q2: 이전 답변을 영어로 번역해줘
A2: Samsung Gauss
```

→ 이전 답변을 "기억"해서 처리됨

v) 주의사항

- `chat_history` key 이름 바꾸지 말 것
- 프롬프트에 `{chat_history}` 없으면 X
- 세션 ID 다르면 대화 끊김
- 대화가 길어지면 → 요약 메모리 필요