Langchain

강의순서

1. 문서 로더 및 텍스트 분할기
2. 임베딩과 벡터 저장소
3. 검색기 및 리랭커
4. RAG(Retrieval-Augmented Generation)

Chapter 0

RAG (Retrieval-Augmented-Generation)

langchain

RAG에 대해 알아봅시다

1. RAG의 필요성

- LLM은 훈련 시점 이후의 새로운 정보나 사내 문서에는 접근할 수 없음
- 하지만 실무에서는 최신 문서 기반의 정확한 답변이 중요함
- RAG는 LLM + 검색엔진을 결합하여 이 문제를 해결하는 기술
  → RAG = LLM의 지식 보강 시스템

langchain

RAG에 대해 알아봅시다

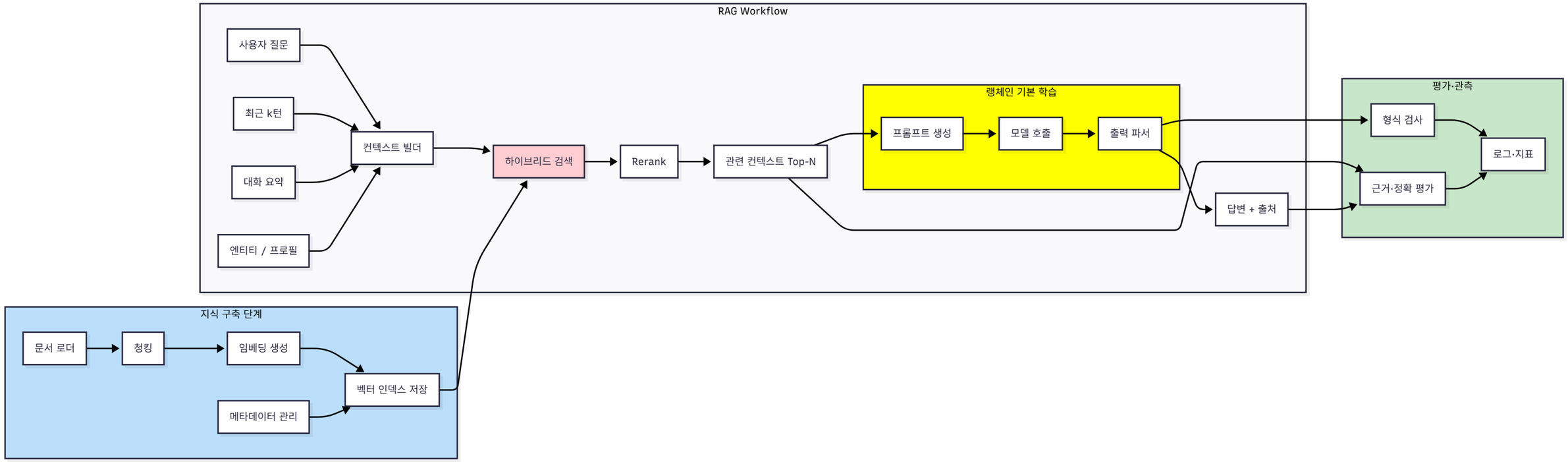
2. RAG의 기본 구조

- 사용자 질문 → 검색기(Retriever) → 관련 문서 추출 → 프롬프트 구성 → LLM 응답 → 답변 + 출처
- Retrieval (검색)
  - 질문과 의미상 가까운 문서 청크를 벡터DB에서 가져옴
- Augmented Generation (생성 보강)
  - 검색된 문서를 기반으로 LLM이 답변 생성
- 결과
  - 단순한 생성이 아닌, 근거가 있는 생성

langchain RAG에 대해 알아봅시다

2. RAG의 기본 구조

- 지식 구축단계 부분이 RAG 파트 라고 생각하면 됩니다



- 큰 범위에서, 이전 대화 내용도 RAG 라고 볼 수 있습니다

langchain RAG에 대해 알아봅시다

3. RAG 구성요소

구성요소	설명	LangChain 구성 요소
문서 로더 (Loader)	PDF, TXT, 웹 등에서 문서 불러오기	DocumentLoader
텍스트 스플리터 (Splitter)	문서를 문단 단위로 분리	RecursiveCharacterTextSplitter
임베딩 (Embedding)	각 문단을 벡터화	OpenAIEmbeddings, HuggingFaceEmbeddings
벡터 저장소 (Vector Store)	벡터 + 메타데이터 저장	Chroma, Qdrant, LanceDB
검색기 (Retriever)	유사 문서 검색	.as_retriever()
프롬프트 & 모델 (Prompt + LLM)	검색된 문서 기반 답변 생성	ChatPromptTemplate, ChatOpenAI
파서 (Output Parser)	모델 출력 정제	StrOutputParser, PydanticOutputParser

langchain

RAG에 대해 알아봅시다

4. RAG 기본 맛보기

- 간단한 기본 예제 RAG 를 구현해보고, 각 파트별로 자세하게 나가도록 하겠습니다
- 예제: 1_rag_basic.ipynb

Chapter 1

문서 로더 및 텍스트 분할기

langchain

문서 로더 및 텍스트 분할기

1. 문서 로더의 종류

1. 텍스트
2. PDF - 텍스트 기반
3. PDF - 이미지 기반
4. 웹문서

주로 쓰는 문서는 pdf 라고 생각하면 됩니다

langchain 문서 로더 및 텍스트 분할기

2. 개념 이해

- 문서 로더(Document Loader)
  - 외부 데이터를 LangChain 문서 객체(Document) 형태로 불러오는 단계
  - 다양한 형식(PDF, DOCX, HTML, CSV, 웹페이지 등)을 통합된 포맷으로 변환

- 텍스트 스플리터(Text Splitter)
  - 너무 긴 문서를 적당한 길이의 청크(chunk)로 나누는 단계
  - 모델이 처리 가능한 입력 단위로 쪼개서 RAG 품질을 높임

langchain 문서 로더 및 텍스트 분할기

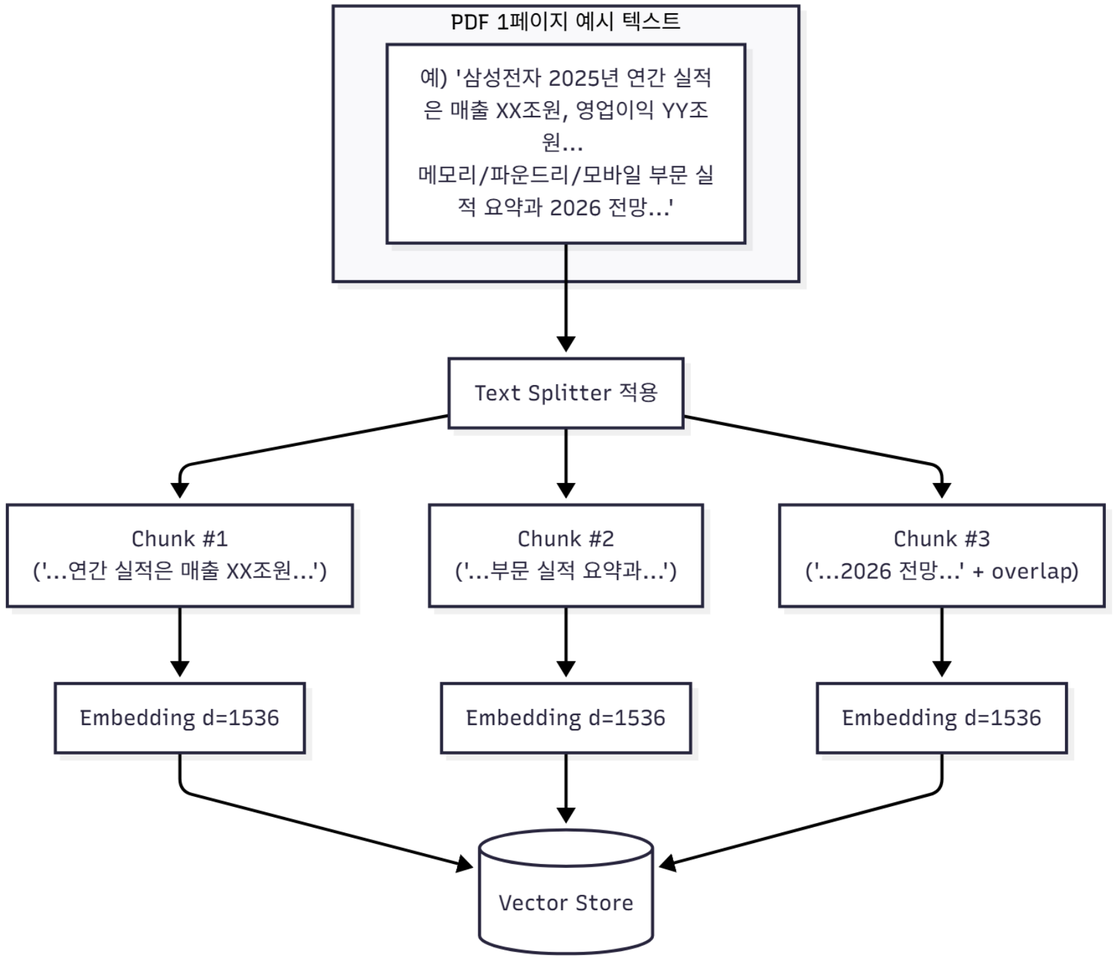
3. 왜 필요한가?

항목	이유
Document Loader	현실의 데이터는 다양한 형식이라, 통일된 구조 필요
Text Splitter	모델의 토큰 제한 때문에 문서를 직접 넣을 수 없음
Chunk 단위 검색	작은 단위로 검색 → 더 정밀한 컨텍스트 반환 가능

langchain 문서 로더 및 텍스트 분할기

4. 흐름도

- 이렇게 만들어진 청크들이 나중에 벡터 스토어에 저장되고
- 질문이 들어오면 가장 유사한 청크를 찾아오는 기능을 합니다



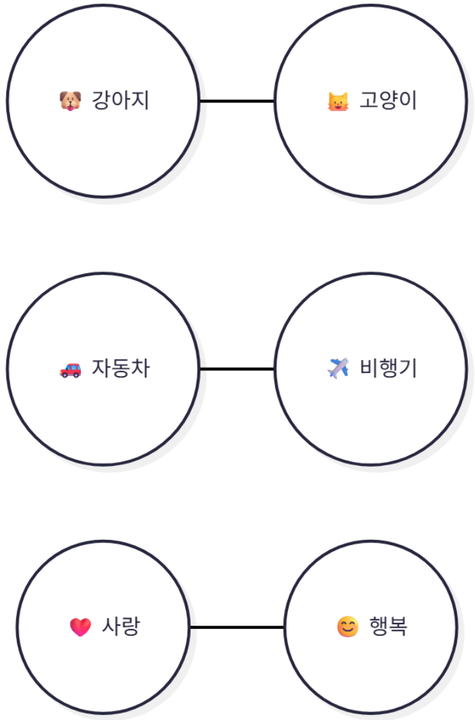
Chapter 2

임베딩과 벡터 저장소

langchain 임베딩과 벡터 저장소

임베딩이란?

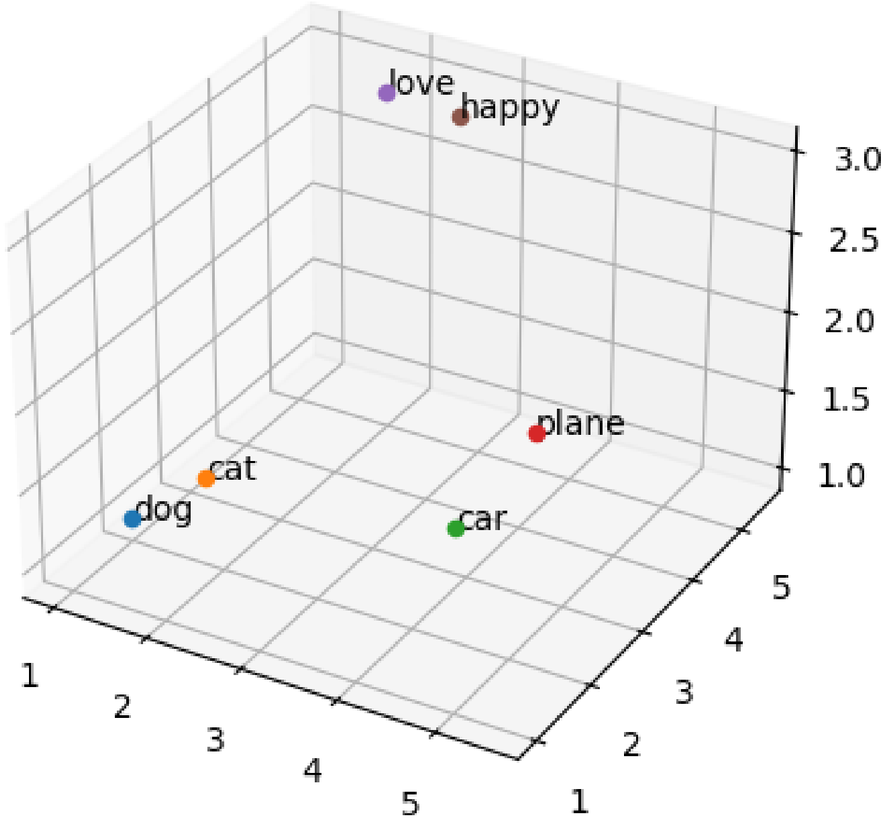
- 임베딩은 텍스트를 숫자 벡터로 바꾸는 과정
- 문장을 숫자로 바꾸되, 단순한 코드가 아니라 의미가 비슷한 것들은 가까운 위치에
- 다른 의미는 멀리 떨어지게 만드는 게 핵심



langchain

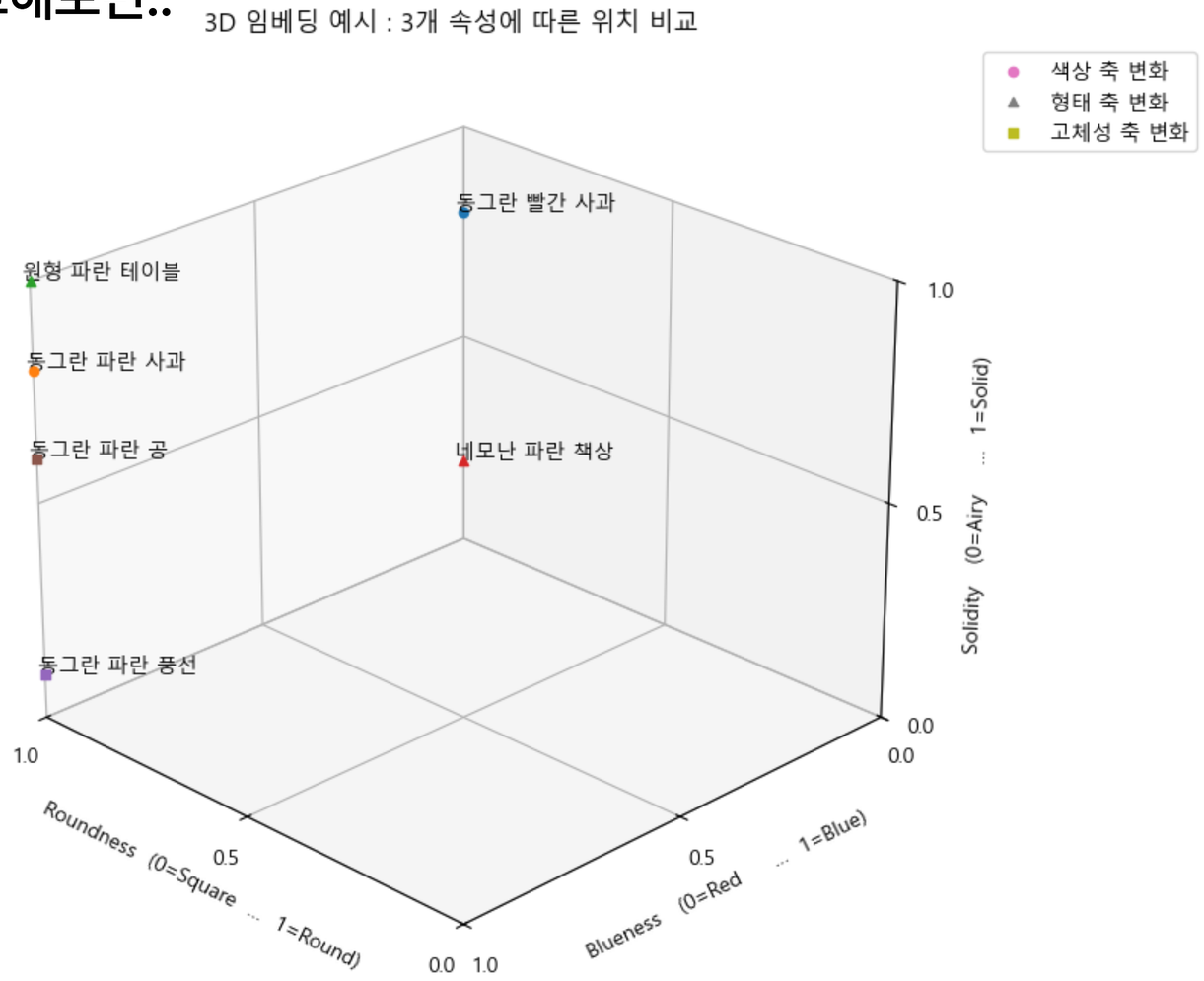
임베딩과 벡터 저장소

3차원으로 적용해보면



langchain 임베딩과 벡터 저장소

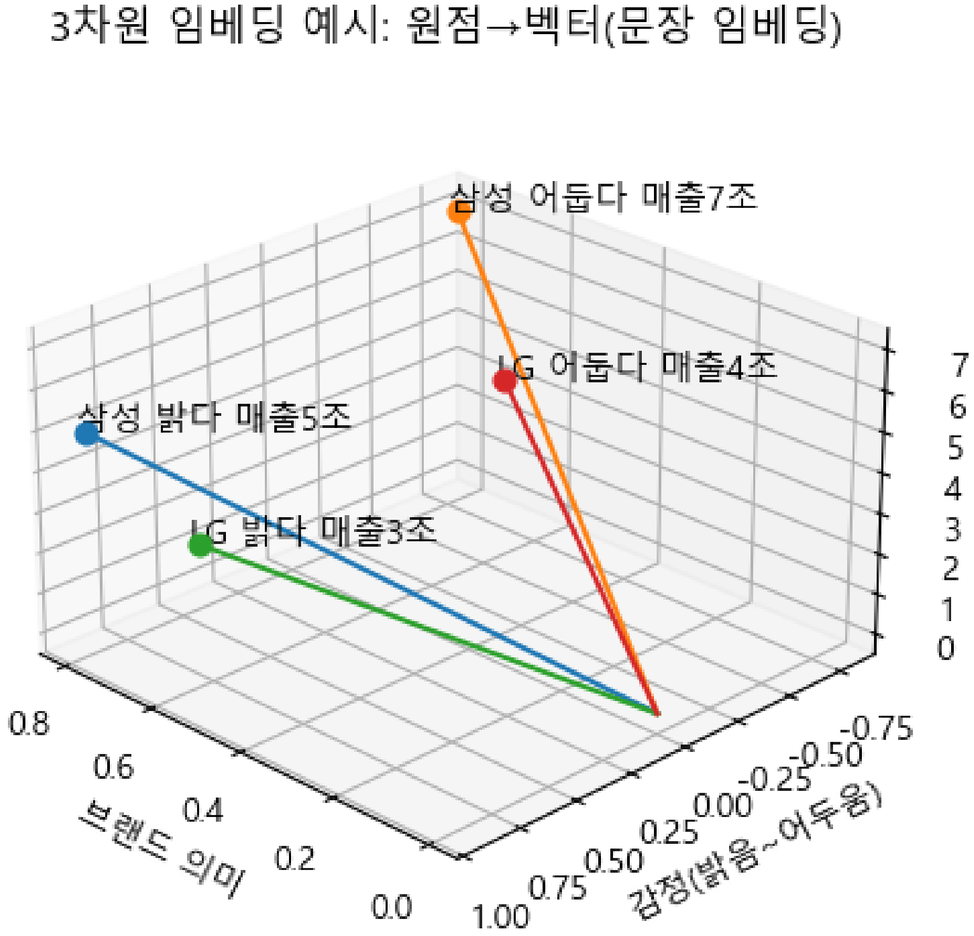
3속성으로 내용의 위치를 비교해보면..



langchain

임베딩과 벡터 저장소

임베딩을 문서 내용으로 적용해보면



langchain 임베딩과 벡터 저장소

임베딩(Embedding)의 개념

1. 벡터란?
   - 문장을 고차원 수치 표현으로 바꾼 것 (예: 1536차원)
2. 예시
   - 친환경 경영 ↔ 탄소 감축 활동 → 의미적으로 가깝다
   - 매출 증가 ↔ 탄소 감축 활동 → 멀다
3. 특징
   - 유사 의미 문장은 코사인 유사도가 높음
   - 같은 의미라도 언어적 표현이 달라도 검색됨

langchain 임베딩과 벡터 저장소

임베딩 차원 예시

모델	차원 수	특성	
text-embedding-3-small	1536	빠르고 저렴	기본 RAG용
text-embedding-3-large	3072	정밀도 ↑	정확도 우선
BAAI/bge-m3	1024	오픈소스 대안	비용 절감

langchain 임베딩과 벡터 저장소

왜 벡터 스토어가 필요한가

1. 문서 검색의 두 방식 비교
   - Keyword 기반 (BM25, TF-IDF) - 예전에 했던 ..
   - Semantic 기반 (Embedding + Similarity)
   - 예
     - 친환경 경영 → 문서에 동일 단어가 있는지 확인
     - 친환경 경영 검색 → 친환경 제품만 나옴
2. 자연어 의미 검색의 한계 → 의미 벡터로 극복
   - 예
     - 친환경 경영 → 지속가능한 생산, 탄소중립 전략 도 유사하게 인식
     - 친환경 경영검색 → 지속가능한 생산, 탄소중립 전략 도 나옴
3. RAG 파이프라인에서의 위치
   - (로더 → 스플리터 → 임베딩 → 벡터DB → 검색 → LLM)

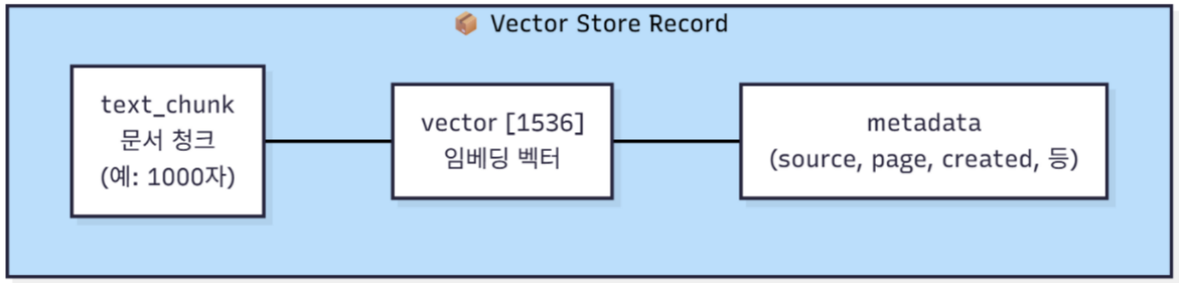
langchain

임베딩과 벡터 저장소

벡터스토어(Vector Store)의 역할

- 벡터(임베딩)를 저장하고, 가까운 벡터를 빠르게 찾는 데이터베이스

벡터스토어 형식



- 하나의 문서 조각(청크) 이 벡터 DB에 저장될 때의 내부 구성
- text_chunk:
  분리된 문서의 일부 텍스트(예: 약 1,000자).
- vector [1536]
  : text_chunk를 임베딩 모델(예: text-embedding-3-small)
  로 변환한 1536차원 의미 벡터.
- metadata (source, page, created, ...)
  : 해당 조각의 출처 정보.
  RAG의 출처 추적(grounding) 과 필터 검색에 활용됨.

Chapter 3

검색기 및 리랭커

langchain 검색기 및 리랭커

1. 벡터 기반 Retriever (Similarity / MMR)

- 개념
  - 질문·문서를 임베딩 벡터로 변환하고 벡터 유사도(주로 cosine)로 Top-K를 찾음
  - MMR(Maximal Marginal Relevance)은 관련도 유지 + 중복 최소화
- 언제 쓰나
  - 일반 텍스트 RAG 기본값
  - 중복 청크가 많으면 MMR로 다양성 확보
- 핵심 파라미터
  - k: 최종 반환 개수(LLM 컨텍스트에 들어갈 청크 수)
  - fetch_k(MMR): 예선 후보 수. fetch_k >= k 권장
  - lambda_mult(MMR): 관련도 가중치(0=다양성↑, 1=관련도↑). 보통 0.3~0.7

langchain 검색기 및 리랭커

2. 하이브리드 Retriever (Vector + Keyword/BM25)

- 개념
  - 의미 검색(벡터)과 정확 키워드 매칭(BM25)을 가중 앙상블로 결합
  - 키워드 의존/용어 민감한 도메인에서 회수율↑
- 언제 쓰나
  - 용어가 정확히 중요(규정/코드/번호)하거나
  - 한글 문서에서 형태소·띄어쓰기 이슈로 벡터만으론 놓치는 케이스
- 핵심 파라미터
  - 앙상블 가중치 weights=[w_vec, w_bm25] (예: [0.6, 0.4])
  - BM25의 k(반환 수), 필요 시 커스텀 토크나이저

langchain 검색기 및 리랭커

3. 압축 Retriever (LLM 압축 / 임베딩 기반 경량 압축)

- 개념
  - 1차 검색 결과에서 불필요 문장을 제거해 컨텍스트 길이를 줄이고 밀도를 올림
  - 두 계열:
    - LLMChainExtractor: 문서 내 관련 문장만 발췌(정밀, 비용↑)
    - EmbeddingsFilter: 쿼리와 유사도 낮은 문장 제거(빠름, 비용0)
- 언제 쓰나
  - 프롬프트 토큰 한도에 빡빡할 때
  - 문서가 장문/중복/군더더기가 많을 때
- 핵심 파라미터
  - LLM 압축: 모델은 경량 + temperature=0 권장
  - 임베딩 필터: similarity_threshold(0.15~0.25부터 탐색), 필요 시 k

langchain 검색기 및 리랭커

4-1. 리랭커 (Reranker)란?

- 개념
  - 1차 검색 상위 후보들을 문장-질문 매칭으로 정밀 재평가해 순서/Top-N 재선정
  - Cross-Encoder(로컬) 또는 Cohere/Jina 등 API형.
- 언제 쓰나
  - 상위권 후보는 맞는데 정확도 한 끗 차로 순서가 아쉬울 때
  - 근거와 질문 정합성을 매우 엄격히 보정하고 싶을 때.
- 핵심 파라미터
  - 모델 선택: cross-encoder/ms-marco-MiniLM-L-6-v2(가볍고 실용적) → 더 큰 모델은 정확도↑/속도↓
  - top_n: 상위 몇 개만 남길지.

langchain 검색기 및 리랭커

4-2. 리랭커 (Reranker)는 왜 필요할까

- 기존 벡터 유사도 검색은 비슷한 의미를 찾지만,
  리랭커는 질문에 가장 직접적으로 답이 되는 문장을 찾는다는 것이 핵심
- 리랭커는 Cross-Encoder 기반
- 즉, 문서와 질문을 한 번에 입력으로 넣고,
  그 둘의 문맥적 관련성을 LLM 스타일로 재평가하는 스타일

- 입력 예시:
  [CLS] 탄소 감축 목표는? [SEP] 삼성전자는 2050년 탄소중립을 선언했다. [SEP]
  ↓
  출력: 0.97 (매우 관련)
- 정밀한 문장-질문 관련도를 계산해서
  - 정말 답이 될 가능성이 높은 것을 위로 올림

langchain 검색기 및 리랭커

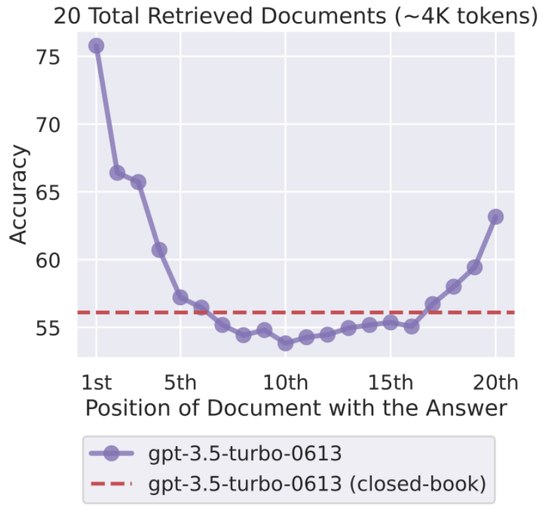
5. LongContextReorder

- 개념
  - 여러 문서(청크)를 LLM에 전달하기 전에,
  - 읽기 좋은 순서로 재배치하는 후처리기(Document Transformer)
- 문서의 내용은 그대로 두고,
  - 중요도·연관도·맥락 연결성을 고려해 순서만 조정함
- LLM이 프롬프트를 읽을 때 핵심 → 배경 → 부록 순서로 이해하도록 도와줌
- 긴 컨텍스트(10개 이상 문서)에서 중앙부 정보 손실(Lost in the Middle) 문제 완화
  → RAG 프롬프트에 다수의 문서(Top-K) 를 그대로 넣을 때
  → 모델이 앞부분 정보만 과도하게 참고하는 현상을 완화

langchain 검색기 및 리랭커

Reorder 가 왜 필요할까?

- 점수가 높은 문서들을 양쪽 끝(앞과 뒤)에 배치
- LLM이 앞/뒤 정보를 잘 기억하니까 정답 포함 문서를 양 끝에 두려는 것



Liu, Nelson F., et al. Lost in the Middle: How Language Models Use Long Contexts.
arXiv preprint arXiv:2307.03172, 2023. Web.

langchain

검색기 및 리랭커

ParentDocumentRetriever

- RAG 프롬프트에 문서 10개 이상 넣을 때 LLM이 어떤 문서부터 읽을지 순서가 중요함
- 문서마다 구조가 섞여 있을 때 서론/본문/부록 순서가 뒤섞이면 맥락이 깨짐
- 답변이 엉뚱한 부분에서 시작될 때 읽기 순서 문제일 가능성 큼 → ParentDocumentRetriever로 해결
- chunk 단위로 앞뒤 연결해서 맥락을 보존

langchain 검색기 및 리랭커

검색기 및 리랭커, 리오더의 조합

상황	추천 조합
문서가 길고, 청크가 많을 때	retriever → LongContextReorder
관련도 순 정확히 정렬하고 싶을 때	retriever → Reranker
최고 품질의 RAG	retriever → Reranker → LongContextReorder (정확도+중요도 유지)

langchain

최종 RAG 체인 생성

최종 RAG 체인 생성해보기

1. 실습으로.. 3_rag_chain.ipynb
2. 대화 내용 기록은 체인을 연결해서 사용