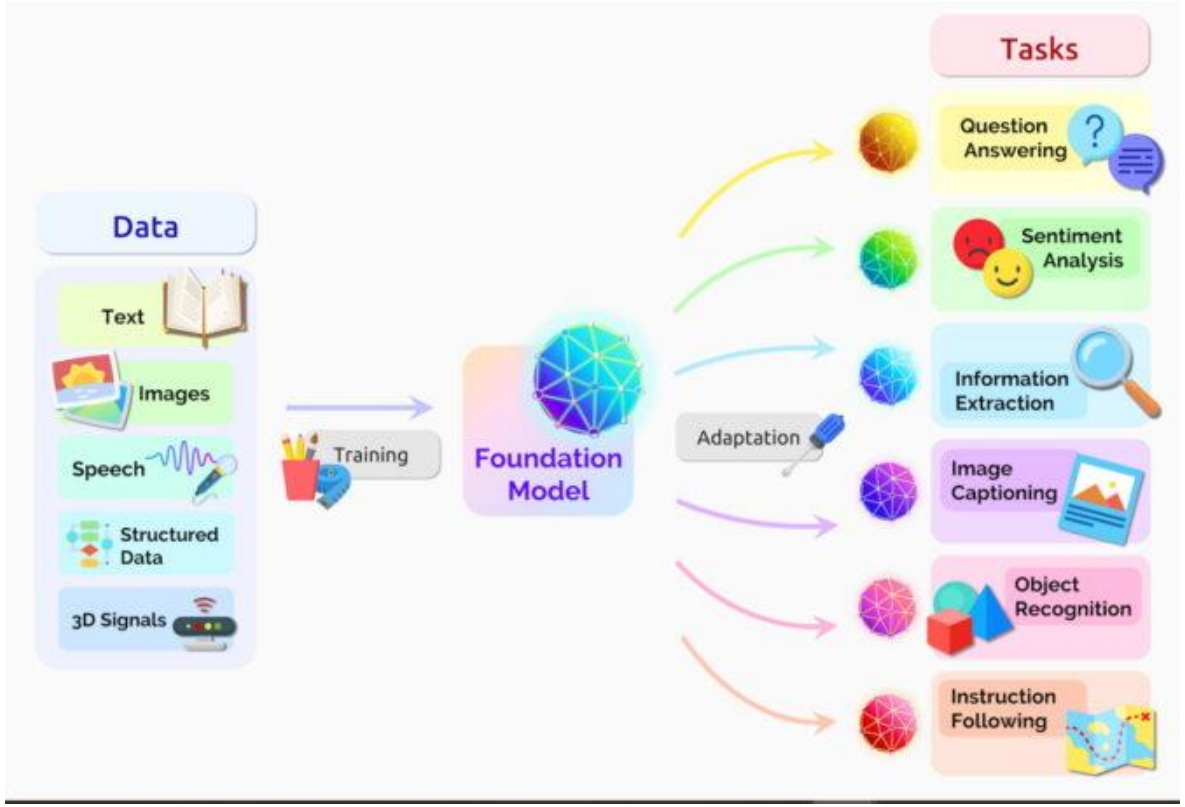
sLM을 활용한 RAG 구현 검토

2023/02/06

오퍼레이션DX기술팀/송태영C

LLM 배경

- 파운데이션 모델
  - 2021년 여름 스텐퍼드에서 AI 모델들을 범주화한 연구 논문에서 소개
  - 방대한 양의 데이터를 self-supervised 학습하여 특정 task를 지정하지 않아도, 요약, 분류, 번역 등 다양한 task를 수행 할 수 있는 모델 등장

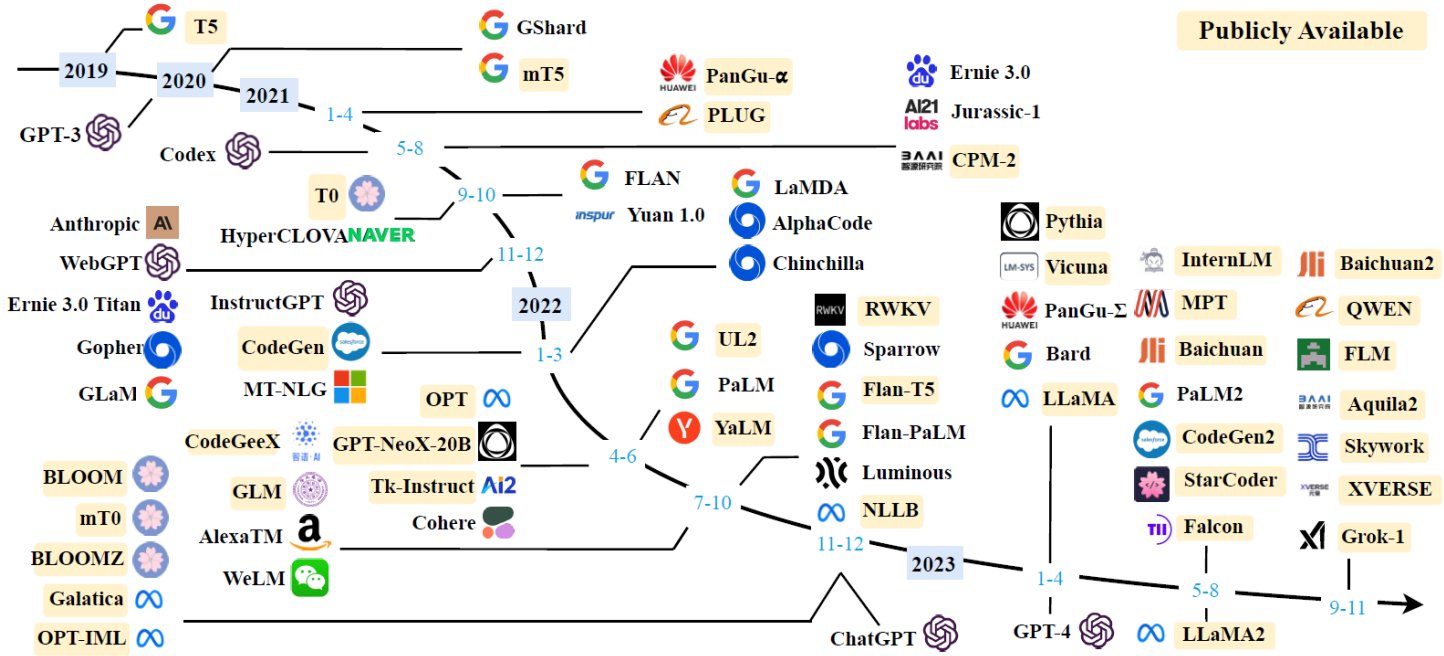


https://blogs.nvidia.co.kr/2023/04/04/what-are-foundation-models/

Rishi Bommasani, Drew A. Hudson, et al. "On the Opportunities and Risks of Foundation Models." arXiv preprint arXiv:2108.07258 (2022).

LLM 배경

- LLM 모델의 계보
  - GPT-3의 등장 이후 모델 크기는 점점 커지고, 범용 Task의 성능이 향상 되었으며 이후 ChatGPT, Bard 등 LLM기반 서비스들이 출시 됨



`Zhao, Wayne Xin, et al. "A survey of large language models." arXiv preprint arXiv:2023.18223 (2023).

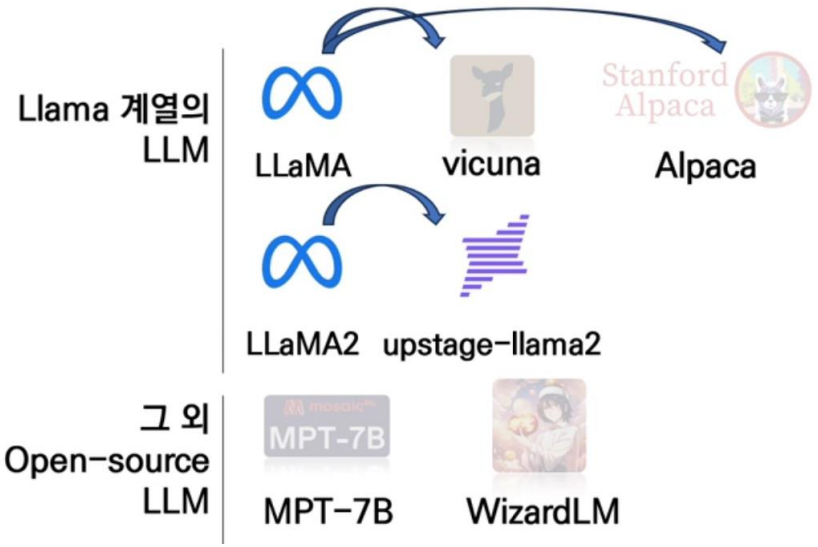
LLM 모델의 분류 – Open/Closed

- Closed Source

개발사	OpenAI	Google
개발 모델	GPT-3	PALM
	GPT-3.5	LaMDA
	GPT-4	Bard

장점 : 뛰어난 성능, API방식의 편리한 사용성
단점 : 보장할 수 없는 보안, API 호출 비용 발생

- Open Source



장점 : Closed source에 못지 않는 성능, 높은 보안성, 낮은 비용
단점 : 높은 개발 난이도, GPU 서버 필요

https://youtu.be/XwILeVhWCCc?si=egIOSmdAbQjaXSRx&t=381

LLM 모델의 분류 – Large/Small

- 용어
  - sLLM : LLM대비 상대적으로 작은 언어모델을 가리키는 용어
  - sLLM(Small LMM)은 sLM(Small Language Model)로 용어가 정리 되어 가고 있음.
  - 매개변수 300, 340억개(30B, 34B) 이하 모델을 지칭
- LLM 과 sLM 비교
  - LLM
    - 구글 PaLM : 5400억개, 학습 기간 슈퍼 컴퓨터 2대로 50일
    - OpenAI GPT-3 : 1750억개, 학습 비용 132억원
    - GPT-4에는 약 1조7000억개 (추정)
  - sLM
    - 스텐보드 Alaphaca-7b : 70억개 매개변수, 52000개, cloud pc 8대, 3시간, 600달러, GPT-3와 유사 성능
    - 데이터 브릭스 돌리: 서버 1대 3시간, 비용 100달러, 60억개 매개변수
- 유명 sLM
  - 메타의 LLaMA2 7B, 알리바바의 큐원 (Qwen 7B), 미스트랄(프랑스 스타업) Mistral 7B, 업스테이지(한국 스타트업) Solar 7B, 마이크로 소프트 Phi-2 3B
  - https://huggingface.co/spaces/HuggingFaceH4/open_llm_leaderboard
- 특징
  - 훈련에 필요한 데이터, 시간, 비용이 상대적으로 적고, 다른 애플리케이션, edge device 등과 통합 용이
  - 온디바이스 AI : 생성 AI 엔진을 스마트폰에서 돌릴 수 있게 된다.
    - Apple Jurnal, 삼성 가우스, Google Nano

Closed LLM의 한계

Fine Tuning 한계	LLM 모델의 fine tuning은 시간 및 비용 한계로 현실적으로 불가능
Hallucination	LLM 모델 내 domain knowledge 부족
Data update 반영의 어려움	새롭게 생성 되거나 갱신된 정보 대응 어려움

Domain knowledge 를 반영하기 위한
sLM 활용 방법의 검토 필요

유명한 한국어 sLM

KoAlpaca	
특징	Polyglot-ko를 한국어로 fine-tuned 한 모델, 지식인 데이터 셋 사용
장점	번역이 아닌 처음부터 한국어로 작성된 training set 포함
단점	다른 오픈소스 sLM 대비 평가 항목의 근거 부족
기타 특징	참고 자료가 많음. 한국어 sLM의 base

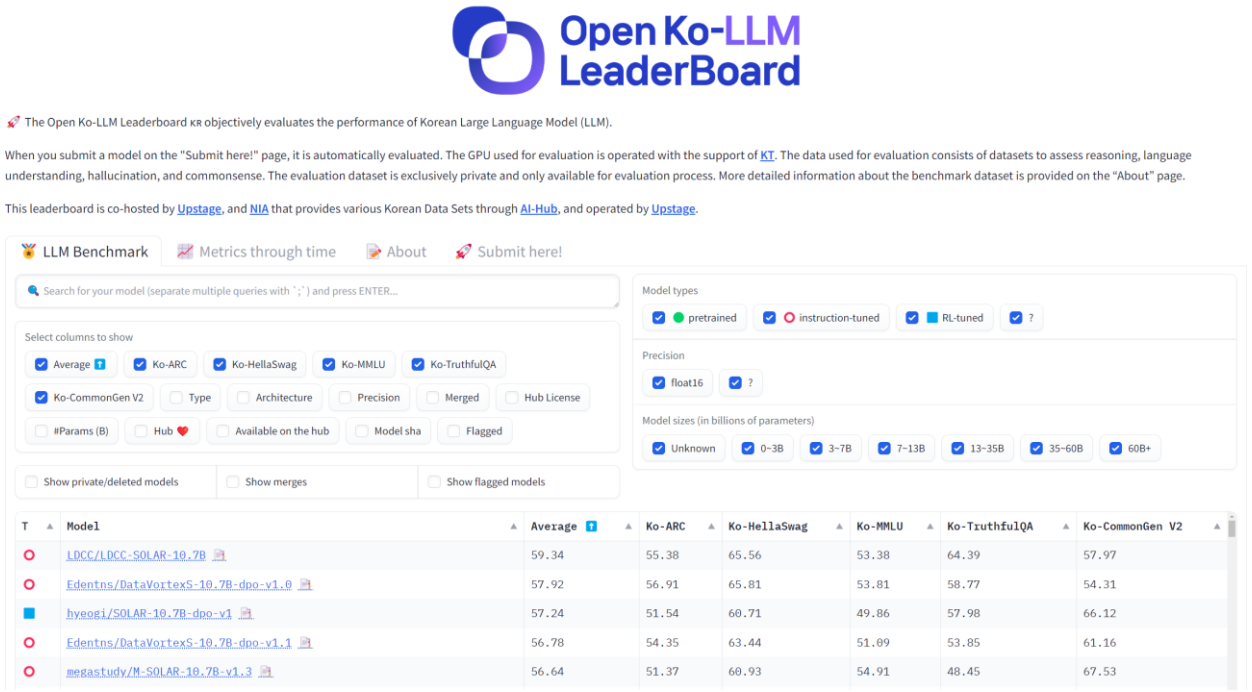
ko_vicuna	
특징	Vicuna를 한국어로 fine-tuned 한 모델
장점	ChatGPT의 QA 데이터셋으로 학습
단점	학습 데이터 set 자체가 ChatGPT 번역 성능에 의존
기타 특징	한국어 sLM 중 가장 높은 성능으로 평가 됨

kullm	
특징	Polyglot-ko를 한국어로 fine-tuned 한 모델 (LoRA, GPT4ALL 사용)
장점	GPT-4 활용한 데이터로 높은 평가
단점	타 sLM 대비 공개된 자료가 많지 않음
기타 특징	PoC 등에서 높은 평가 보고 됨

2023년, 한국어 데이터 셋으로 파인튜닝 한 sLM로 PoC 사례 다수

https://youtu.be/WWaPGDS7ZQs?si=BFyN8MTOmhILhWY6

한국어 sLM 리더 보드



업스테이지와 NIA가 주관하여 한국어 오픈소스 LLM 리더보드 운영 중

https://huggingface.co/spaces/upstage/open-ko-llm-leaderboard

[Demo] LLM모델 및 tokenizer 불러 오기

- Transformers의 AutoModel을 사용
  - 모든 클래스에 걸쳐 Auto 클래스가 존재
  - Model ID (ex:bert-base-cased) 만으로도 손쉽게 모델 다운로드 가능

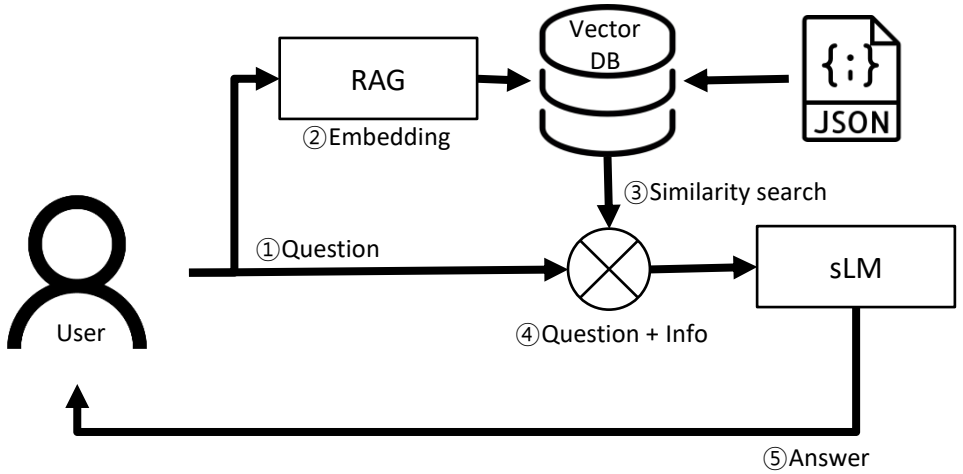
```python
import torch
from transformers import AutoTokenizer, AutoModelForCausalLM

model_id = "kyujinpy/Ko-PlatYi-6B"
model = AutoModelForCausalLM.from_pretrained(
        model_id,
        return_dict=True,
        torch_dtype=torch.float16,
        device_map='auto'
)
tokenizer = AutoTokenizer.from_pretrained(model_id)
```

```
Loading checkpoint shards: 100% 2/2 [00:04<00:00, 2.14s/it]
```

https://velog.io/@taekkim/Hugginface-transformer-%EC%84%A4%EA%B3%84%EA%B5%AC%EC%A1%B0-%EC%82%B4%ED%8E%B4%EB%B3%B4%EA%B8%B0

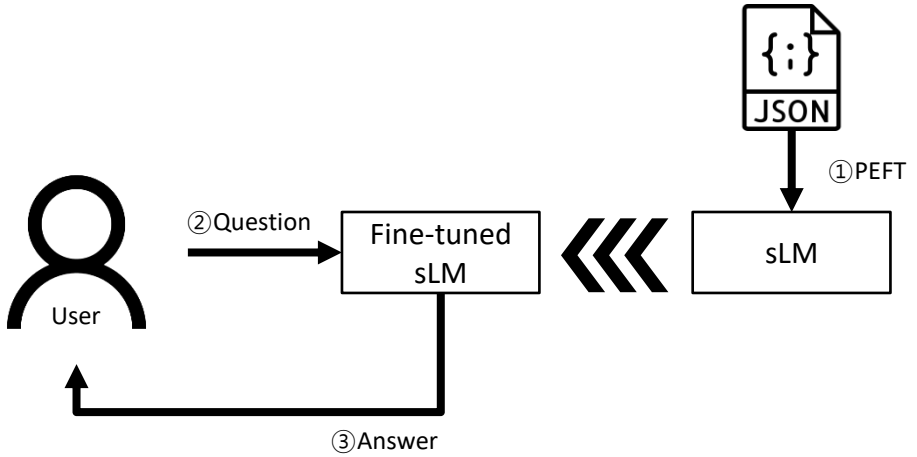
sLM에 데이터 반영 방법

- RAG(Retrieval Augmented Generation)



- 유사성(similarity) 기반으로 question과 유사한 문서를 Vector DB로 부터 검색
- 검색된 문서를 context로 하여 question과 함께 sLM에 문의

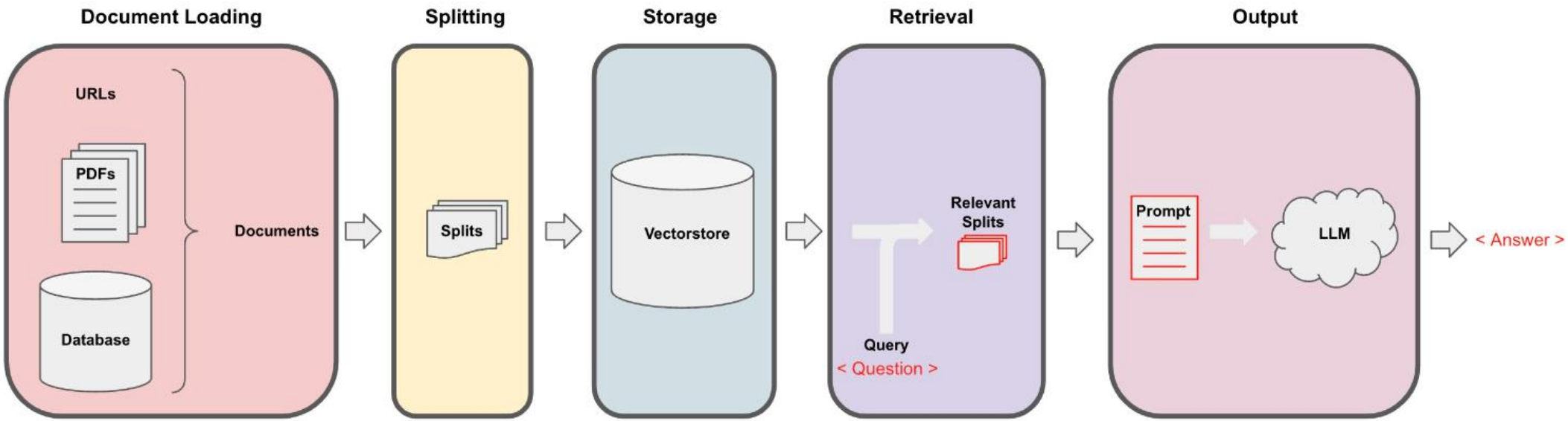
- PEFT(Parameter Efficient Fine Tuning)



- 모델 전체 parameter를 fine-tuning 하기엔 너무 비용이 큼
- 모델의 일부 parameter만 갱신하여 fine-tuning
- LoRA(Low-Rank Adaptation)

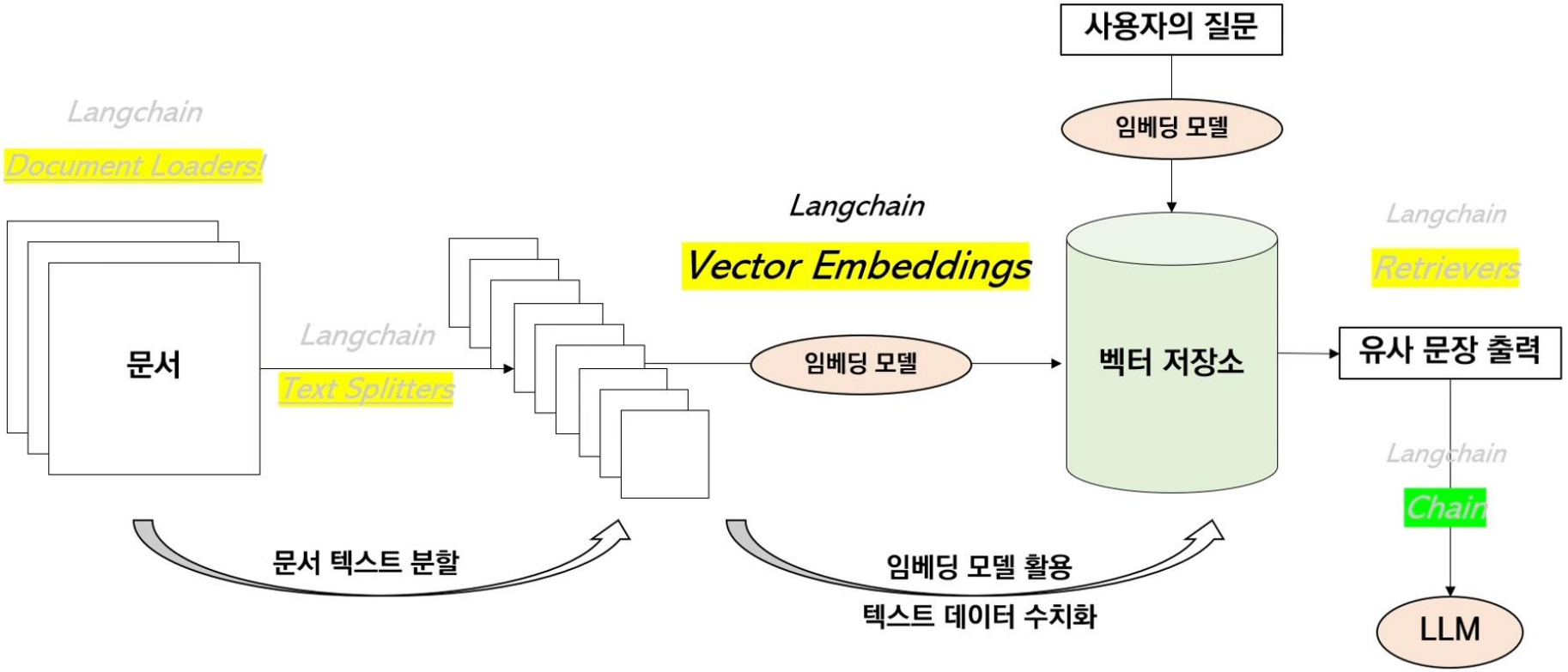
https://youtu.be/WWaPGDS7ZQs?si=BFyN8MTOmhILhWY6

The overall workflow for retrieval augmented generation (RAG)



https://learn.deeplearning.ai/langchain-chat-with-your-data/lesson/5/retrieval

The overall workflow for retrieval augmented generation (RAG) with langchain



https://youtu.be/tQUtBR3K1TI?si=plfQuxzVYP2GarFa

RAG의 구현 예

- Data 가 입력되지 않은 결과
  - KoAlpaca를 활용한 구현 결과

[질문] : 백종원의 증조 할아버지 이름이 뭐야?

[답변] 백종원의 증조할아버지의 이름은 John Smith입니다.

VectorStore에서 필요한 데이터를 잘 retrieval 하는 것이 포인트

- Data가 입력 된 경우 결과

[질문] : 백종원의 증조 할아버지 이름이 뭐야?

[데이터] : 백종원은 1966년 충남 예산군에서 집안의 종손으로 태어났다. 중학교 시절 상경해 강남 8학군 서울고등학교[23]를 졸업하고 연세대학교에 입학했다. 고등학교 졸업 직후엔 잠시 서울특별시 장한평에 위치한 중고차 시장에서 자동차 중개업자로 활동하기도 했다. 2019년, KBS의 토크쇼 프로그램인 대화의 희열에 출연하여 어렸을 적 이야기를 많이 했는데, 만석꾼이었던 증조할아버지 백영기(白榮基)의 피를 이어받은 영향인지 어렸을 때부터 장사꾼 기질이 있다고 스스로 자각하고 있었다고 한다. 9살 때에는 산에 놀러갔다가 본 버섯 농장에서 별다른 투자도 안 한 거 같은데 돈이 된다는 이야기를 듣고 꿈을 버섯 농사로 정한 적도 있었고, 초등학교 4학년때는 캔이 아닌 병에 음료가 나올 시절에 음료수 병을 보고 '저게 돈이 될 것 같다' 고 생각해 학교 리어카를 빌려 오락시간과 보물찾기 같은 시간을 다 건너뛰고 리어카 6개 분량의 공병을 모아서 고물상에 갖다 팔아 큰 돈을 벌었다고 한다. 그리고 5학년 1학기까지 이렇게 돈을 벌었고, 방위성금으로 다 냈다고 한다.

[답변] 백종원의 증조 할아버지 이름은 백영기(白榮基)입니다.

https://velog.io/@acdongpgm/LLM-KULLM%EA%B5%AC%EB%A6%84-Instruction-%EC%82%AC%EC%9A%A9%ED%95%B4%EB%B3%B4%EA%B8%B0

[Demo] Document Loading

- langchain.document_loaders 는 다양한 형식의 소스로 부터 Document 객체 생성 지원
- 지원하는 소스
  - csv, 파일 디렉토리, HTML, JSON, Markdown, PDF, Web
- 데이터 :
  - '아이유', '백종원', '김연경', '박찬호', '손흥민', '정우성', '김연아' 의 위키
- 사용 예

```
from langchain_community.document_loaders import PyPDFLoader

loader = PyPDFLoader("example_data/layout-parser-paper.pdf")
pages = loader.load_and_split()
```

PDF

```
from langchain_community.document_loaders import WebBaseLoader
```

```
loader = WebBaseLoader("https://www.espn.com/")
```

```
data = loader.load()
```

Web

https://python.langchain.com/docs/modules/data_connection/document_loaders/

[Demo] Text Splitting

- LLM은 입력 토큰에 제한(gpt 3.5는 4096토큰)이 있음 → 문서 분할(chunk) 필요 (본 demo에서는 500토큰)
- VectorStore는 chunk 하나당 하나의 vector 할당
- langchain.text_splitter의 Text splitter
  - CharacterTextSplitter : 구분자 1개를(예를 들어 개행) 기준으로 분할 (max_token을 지키지 못하는 경우 발생)
  - RecursiveCharacterTextSplitter : 줄바꿈, 마침표, 쉼표 순으로 재귀적으로 분할, max_token 지켜 분할

```python
from langchain.text_splitter import RecursiveCharacterTextSplitter

text_splitter = RecursiveCharacterTextSplitter(
        separators=['\n', '\n\n', '. ', '', ' '],
        chunk_size=500,
        chunk_overlap=50
) # token의 길이는 기본 함수 len 사용
docs = text_splitter.split_documents(pages)
```

RecursiveCharacterTextSplitter 예시

https://api.python.langchain.com/en/latest/text_splitter/langchain.text_splitter.TextSplitter.html

[Demo] Embedding

- Split 된 document의 chunk를 VectorStore에 넣기 전에 embedding을 해준다.
- VectorStore에서는 입력 받은 query 와 chunk vector 간에 거리를 유사도 기반으로 제공 한다
- HuggingFace로 부터 받는 embedding model 들은, 언어 별로 학습이 되어 있음 (단점 임), 따라서 여러 언어가 섞여 있을 경우 성능 저하 가능 성이 있음.

```python
from langchain.embeddings import HuggingFaceEmbeddings

model_name = "jhgan/ko-sbert-nli"
model_kwargs = {'device':device}
encode_kwargs = {'normalize_embeddings': True}
hf = HuggingFaceEmbeddings(
    model_name=model_name,
    model_kwargs=model_kwargs,
    encode_kwargs=encode_kwargs
)
```

HuggingFace로 부터 embedding 모델 불러오기

```python
import numpy as np
from sentence_transformers import SentenceTransformer
model = SentenceTransformer('multi-qa-MiniLM-L6-cos-v1')
```

```python
sentence1 = "i like dogs"
sentence2 = "i like Bulldog"
sentence3 = "the weather is ugly outside"
embedding1 = model.encode(sentence1)
embedding2 = model.encode(sentence2)
embedding3 = model.encode(sentence3)
```

```python
np.dot(embedding1, embedding2)
```

```
0.6957926
```

```python
np.dot(embedding1, embedding3)
```

```
0.08369687
```

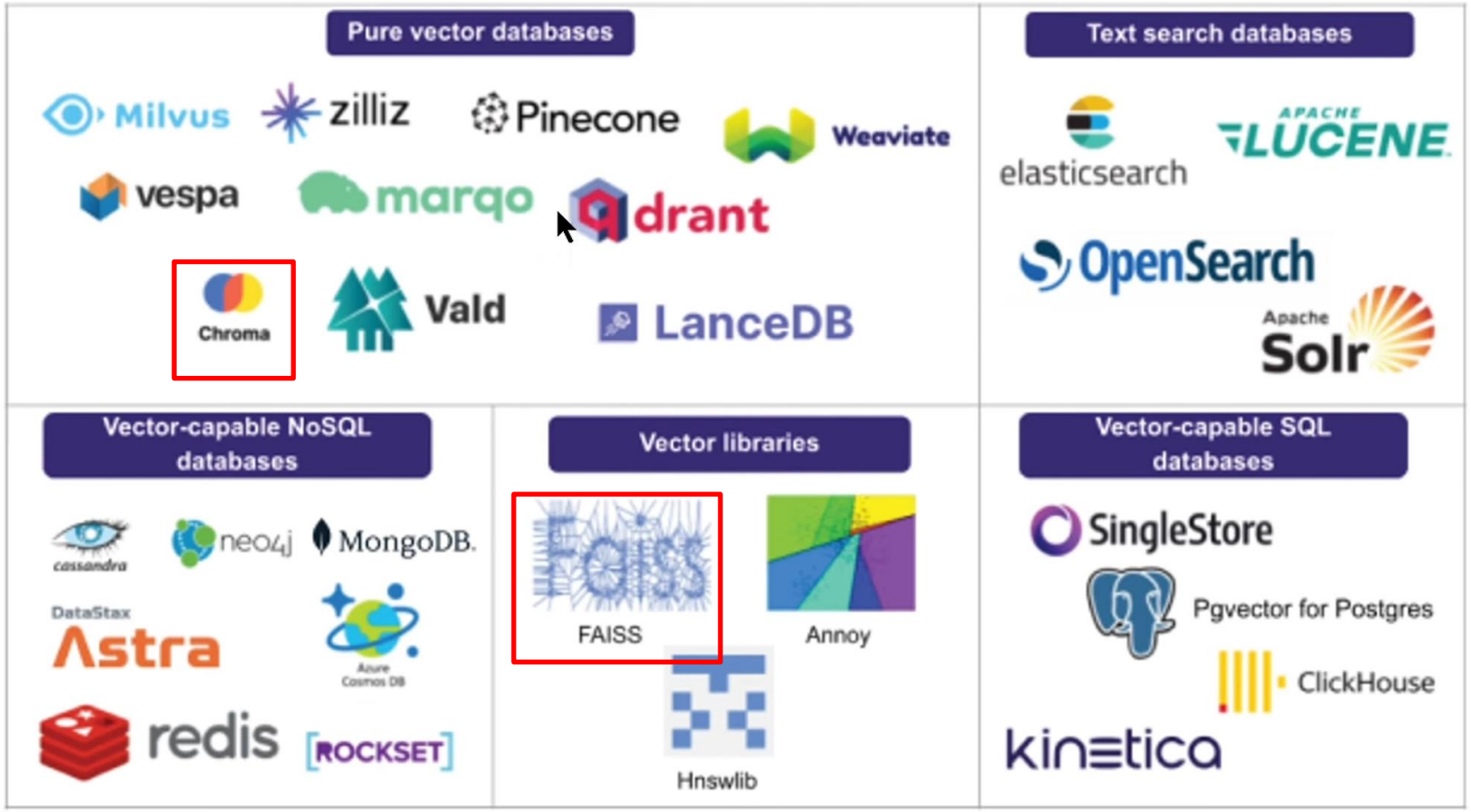
```python
np.dot(embedding2, embedding3)
```

```
0.030988622
```

Embedding 및 유사도의 예

https://docs.llamaindex.ai/en/stable/examples/embeddings/huggingface.html

VectorStore



https://www.linkedin.com/pulse/choosing-vector-database-your-gen-ai-stack-abhinav-srivastava/

[Demo] Chroma/Faiss

- Chroma
  - 대표적인 오픈소스 VectorStore
  - 텍스트, 오디오, 비디오 등의 비구조화된 데이터를 머신 러닝 모델이 사용할 수 있는 형태로 인코딩
  - 추천 시스템, 이미지 인식, 자연어 처리에 사용
  - API 제공, 테이블(컬랙션)생성, 문서 추가, 컬렉션 쿼리
- Faiss(Facebook AI Similarity Search)
  - Facebook AI 유사성 검색은 고밀도 벡터의 효율적인 유사성 검색 및 클러스터링을 위한 라이브러리
  - 모든 크기의 벡터 집합에서 검색하는 알고리즘이 포함되어 있음
  - 평가 및 매개변수 조정을 위한 코드도 제공

```python
db_type = "Chroma"
# db_type = "Faiss"

if db_type == "Chroma":
    db = Chroma.from_documents(docs, hf, persist_directory='./chroma_db')
elif db_type == "Faiss":
    db = FAISS.from_documents(docs, hf)
    db.save_local('./faiss_db')
else:
    raise('You have wrong db type')
```

VectorStore 생성 및 저장

```python
db_type = "Chroma"
# db_type = "Faiss"

if db_type == "Chroma":
    db = Chroma(persist_directory='./chroma_db', embedding_function=hf)
elif db_type == "Faiss":
    db = FAISS.load_local("./faiss_db", hf)
else:
    raise('You have wrong db type')
```

VectorStore 불러오기

```python
db = Chroma.from_documents(docs, hf, persist_directory='./chroma_db')
```

Split 된 document

Embedding model

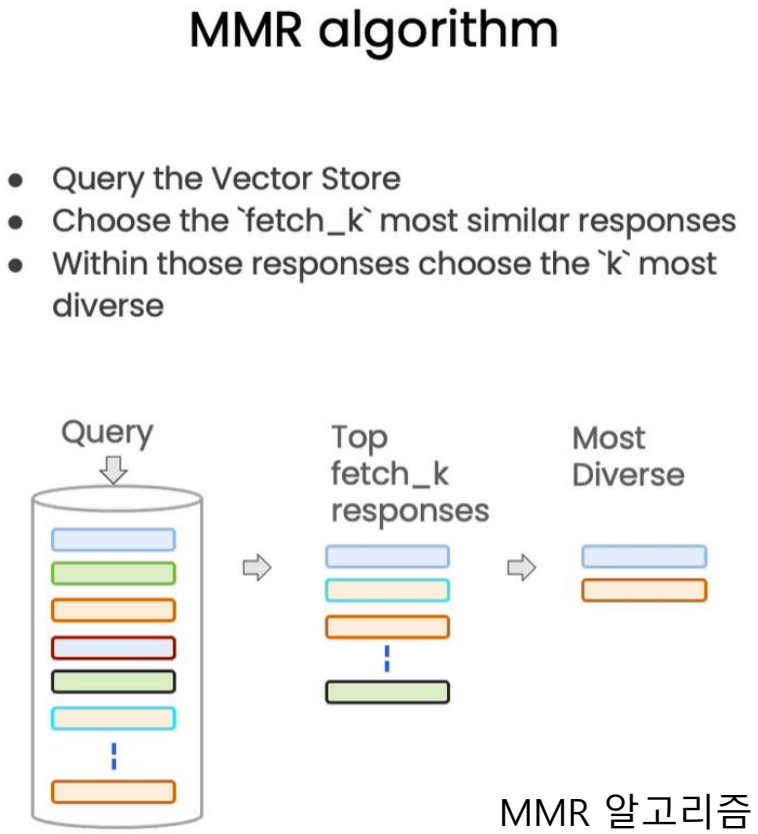
저장할 위치

Retriever

- VectorStore로 만든 검색기
- search_type : VectorStore에서 사용자 질문과 연관된 chunk를 뽑아내는 방법
  - similarity
  - mmr : max_marginal_relevance_search, 유사한 것들 중에 가능하면 다양한 결과를 뽑아 내도록 한다
  - similarity_score_threshold

```
tmp_query = "정우성의 가장 친한 사람?"
tmp_text = db.max_marginal_relevance_search(
    tmp_query, k=5, fetch_k=10, lambda_mult=1)
```

MMR 구현 예



MMR 알고리즘

https://learn.deeplearning.ai/langchain-chat-with-your-data/lesson/5/retrieval

[Demo] Chain

```python
from langchain.llms import HuggingFacePipeline
from transformers import pipeline

text_generation_pipeline = pipeline(
    model=model,
    tokenizer=tokenizer,
    task="text-generation",
    temperature=0.2,
    return_full_text=True,
    max_new_tokens=300,
)

koplatyi_llm = HuggingFacePipeline(pipeline=text_generation_pipeline)
```

Huggingface의 pipeline 객체 및 llm 객체 생성

```python
from langchain.prompts import PromptTemplate
from langchain.chains import LLMChain

prompt_template = """
### [INST]
Instruction: Answer the question based on your knowledge.

Here is context to help:

{context}

### QUESTION:
{question}

[/INST]
 """

# Create prompt from prompt template
prompt = PromptTemplate(
    input_variables=["context", "question"],
    template=prompt_template,
)

# Create llm chain
llm_chain = LLMChain(llm=koplatyi_llm, prompt=prompt)
```

PromptTemplate 및 LLMChain 객체 생성

[Demo] 질문해 보기

- 손홍민의 아버지 이름은? : 손웅정
- 김연아의 첫 CF : KB국민은행
- 김연경의 키는? : 192cm
- 박찬호 고등학교? : 공주고등학교
- 박찬호가 출연한 영화 : 서울서칭
- 김연아가 졸업한 고등학교 : 수리고등학교
- .
- .
- .
- Retrieval 되는 문서에 따라 답의 품질이 결정 된다.

```
from langchain.schema.runnable import RunnablePassthrough

retriever = db.as_retriever(
                                search_type="similarity",
                                search_kwargs={'k': 6}
                            )

rag_chain = (
 {"context": retriever, "question": RunnablePassthrough()}
 | llm_chain
)

question = "손흥민 아버지 이름은"
result = rag_chain.invoke(question)
result
```

참고 문서 수

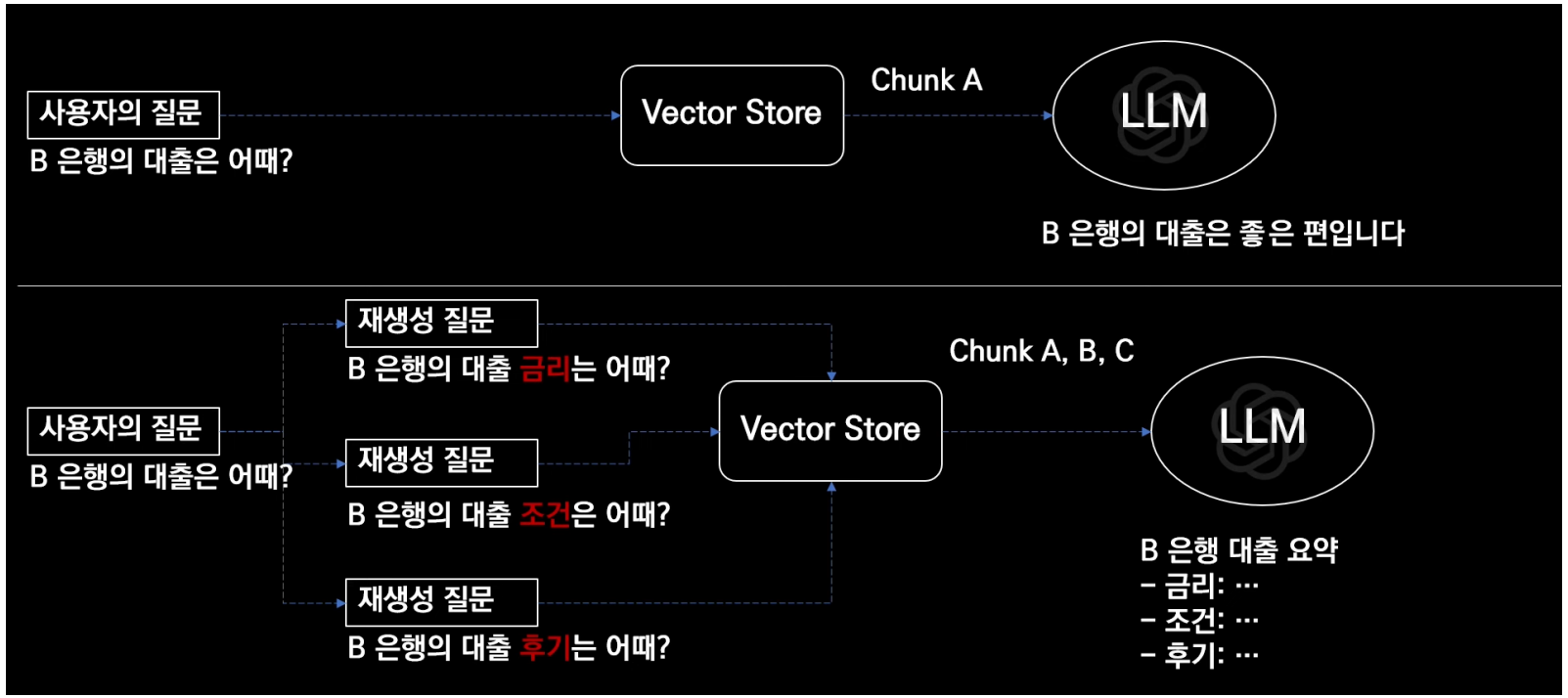
Retrieval 품질을 올리는 기법들

- Retriever는 "콩떡같이 말해도 찰떡같이 알아들어야 한다."
  - **Multi-query Retriever** [Demo] : 사용자의 질문을 보완하는 복수개의 질문을 LLM을 통해 만든다
- 질문과 관련된 앞뒤 문맥을 잘 전달 해야 한다
  - Parent-document Retriever : 전달하는 문서의 상위 문서(전체 배경이 포함된)를 함께 전달 한다.
- 오래된 자료는 덜 참조하게 하고 싶다
  - Time-Weighted Vector Retriever
- 문맥 뿐만 아니라 토큰이 확실하게 포함된 검색을 하고 싶다
  - **Ensemble Retriever** [Demo] : Sparse Retriever와 Dense Retriever의 장점을 모두 사용
- RAG의 Source Document가 많을 수록, 중간 순서는 무시
  - Long Context Order Retriever

https://youtu.be/J2AsmUODBak?si=rUpyj7-OamNM47n8
https://youtu.be/ehP4vphl_Us?si=_mORI1sE0Hf2TVUq

[Demo] Muti-Query Retriever

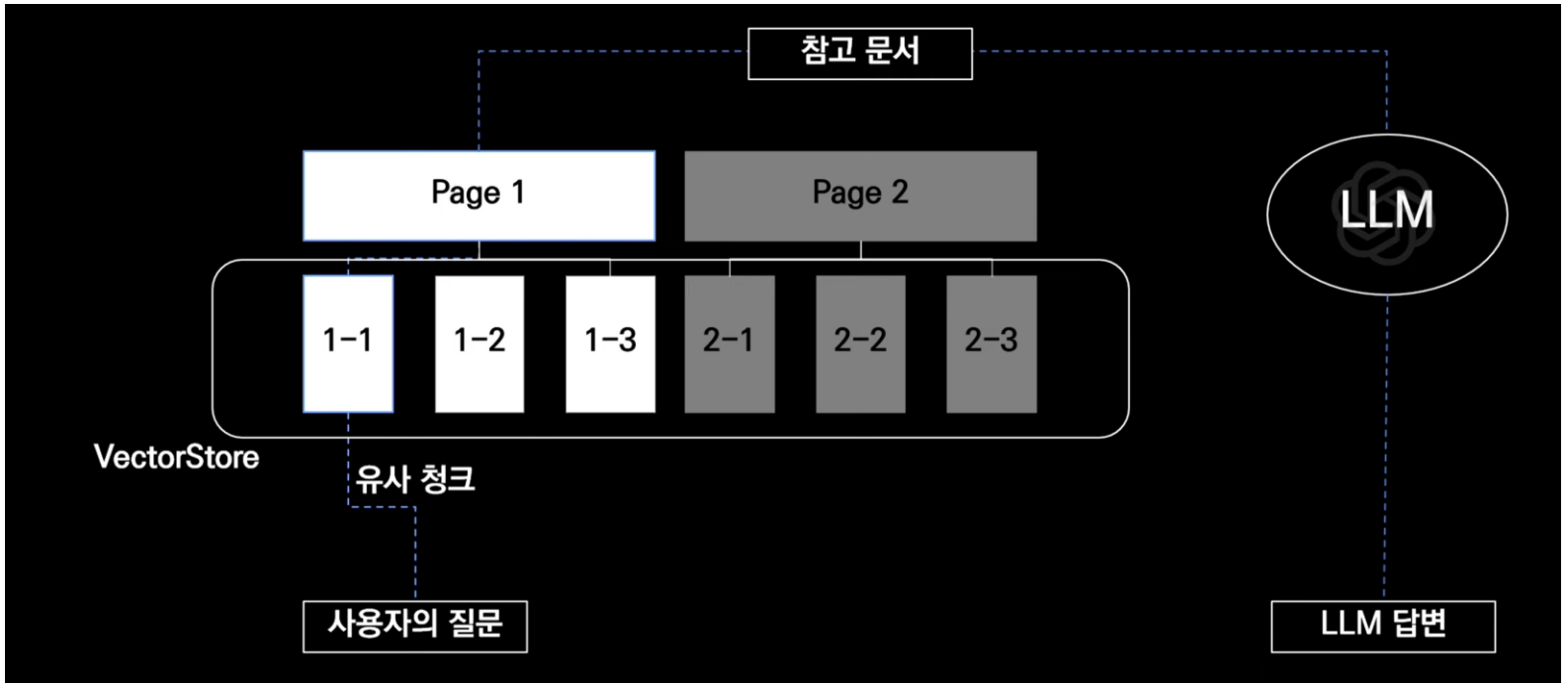
- 콩떡같이 말해도 찰떡같이 알아들게 하기 위해, 사용자 질문을 여러 개의 유사 질문으로 재생성



https://youtu.be/J2AsmUODBak?si=rUpyj7-OamNM47n8
https://youtu.be/ehP4vphl_Us?si=_mORI1sE0Hf2TVUq

Parent-Document Retriever

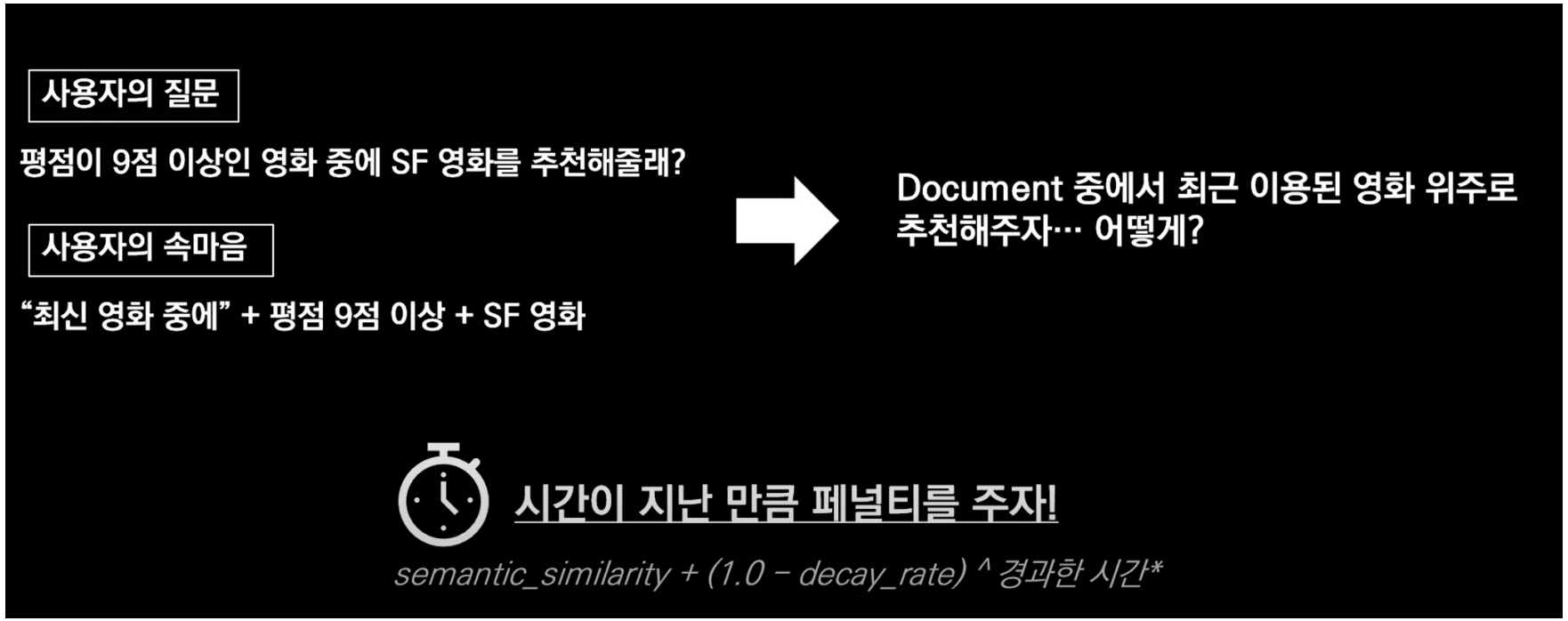
- 유사 문서의 부모 문서를 참고하므로, 조금 더 맥락을 담아 LLM에게 제공



https://youtu.be/J2AsmUODBak?si=rUpyj7-OamNM47n8
https://youtu.be/ehP4vphl_Us?si=_mORI1sE0Hf2TVUq

Time-Weighted Vector Retriever

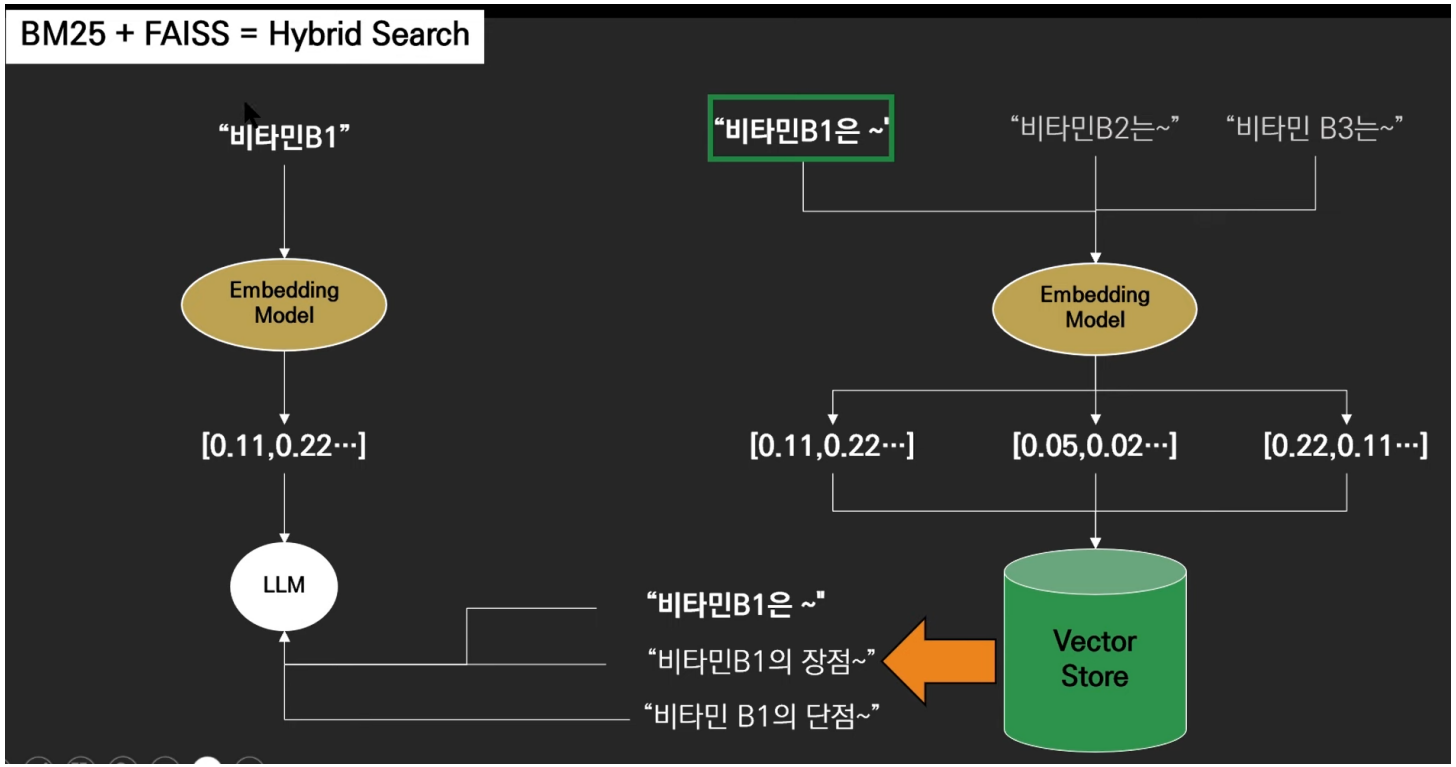
- 가장 최근에 이용된 문서를 기준으로 먼저 참고 하도록 가중치



https://youtu.be/J2AsmUODBak?si=rUpyj7-OamNM47n8
https://youtu.be/ehP4vphl_Us?si=_mORI1sE0Hf2TVUq

[Demo] Ensemble Retriever

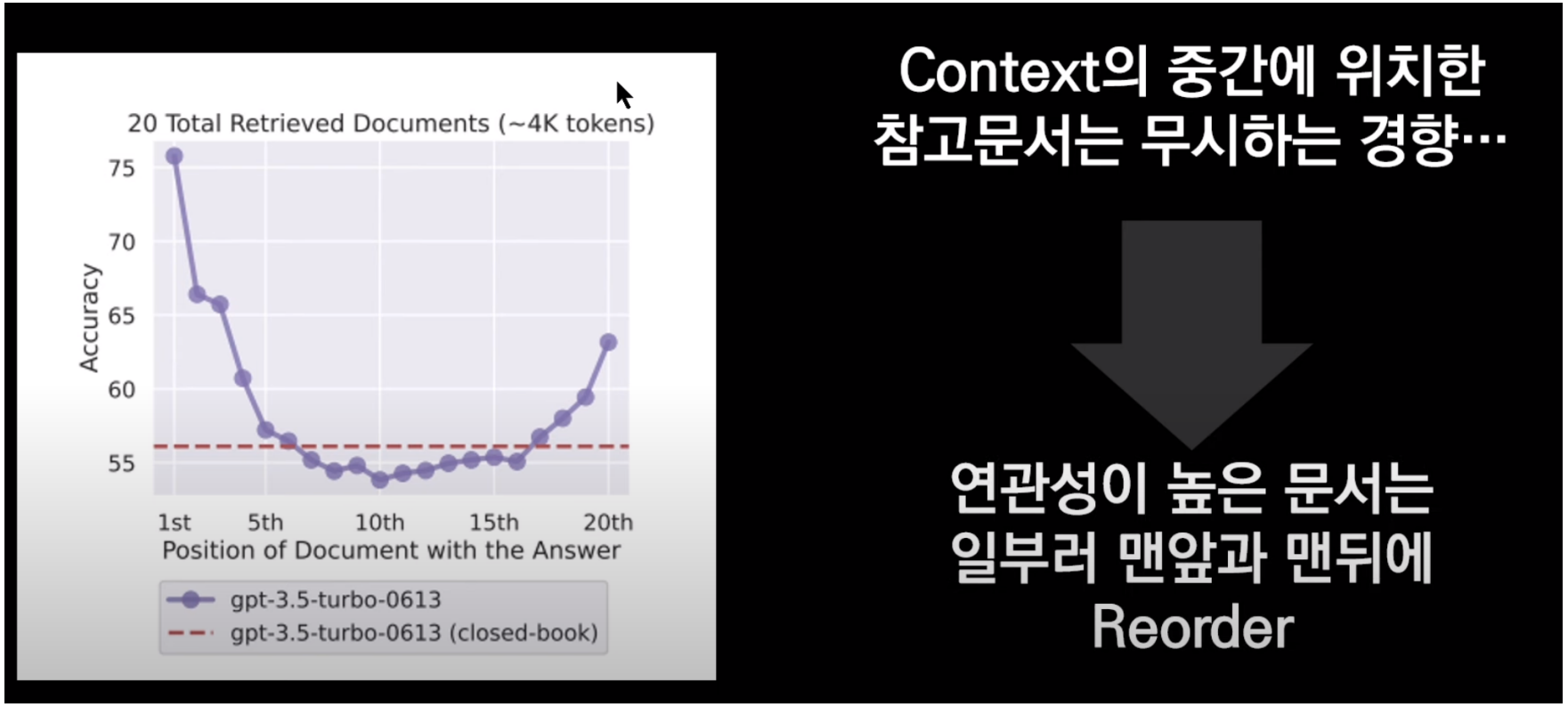
- Sparse Retriever인 BM25와 결합



https://youtu.be/J2AsmUODBak?si=rUpyj7-OamNM47n8
https://youtu.be/ehP4vphl_Us?si=_mORI1sE0Hf2TVUq

Long Context Reorder

- RAG의 Source Document가 많을 수록, 중간 순서는 무시



https://youtu.be/J2AsmUODBak?si=rUpyj7-OamNM47n8
https://youtu.be/ehP4vphl_Us?si=_mORI1sE0Hf2TVUq

[유첨] Open AI 비용

OpenAI API 이용 요금

OpenAI API는 유료이지만, OpenAI 계정 생성 시 5달러 상당의 무료 크레딧(사용 기한 3개월)이 제공됩니다.

> 이용 요금 – OpenAI API
> https://openai.com/pricing

OpenAI API의 텍스트 생성 및 채팅은 사용하는 모델과 LLM 입출력의 토큰 수에 따라 이용료가 달라집니다.

토큰은 LLM이 처리하기 위해 작게 쪼갠 텍스트의 단위입니다. 영어의 경우 1단어가 1~2토큰 정도이지만, 한국어의 경우 1문자가 1~2토큰 정도로 비싸게 책정돼 있습니다. OpenAI 플레이그라운드에서 실제 소요되는 토큰 수를 확인할 수 있습니다(추후 설명).

각 모델의 이용 요금은 다음과 같습니다.

GPT-4

GPT-4는 LLM 입출력의 최대 토큰 수가 다른 두 가지 모델(8K와 32K)이 제공되고 있습니다. 또한, 프롬프트와 컴플리션에 따라 요금이 다릅니다. K는 1,000을 의미하는 단위로, 1K 토큰은 1,000 토큰입니다.

표 2-2-1 GPT-4 요금

모델	종류	요금(달러)	대한민국 원화 기준
8K	프롬프트 컴플리션	0.03달러/1K 토큰 0.06달러/1K 토큰	1달러 1,300원으로 환산 시 약 39원 1달러 1,300원으로 환산 시 약 78원
32K	프롬프트 컴플리션	0.06달러/1K 토큰 0.12달러/1K 토큰	1달러 1,300원으로 환산 시 약 78원 1달러 1,300원으로 환산 시 약 156원

OpenAI API 이용 요금

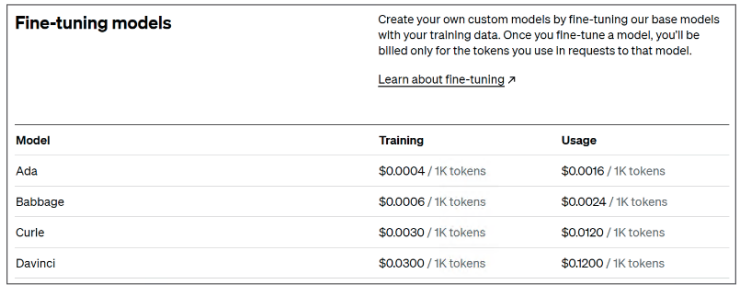
파인튜닝도 OpenAI API 이용료가 부과됩니다. 학습 시에는 사용하는 모델과 학습 데이터의 토큰 수, 사용 시에는 사용하는 모델과 LLM 입출력의 토큰 수에 따라 이용료가 달라집니다.

표 4-4-1 파인튜닝 모델의 요금(학습)

모델	요금(달러)	대한민국 원화 기준
Davinci	0.0300달러/1K 토큰	1달러 1,300원으로 환산 시 약 39원
Curie	0.0030달러/1K 토큰	1달러 1,300원으로 환산 시 약 3.9원
Babbage	0.0006달러/1K 토큰	1달러 1,300원으로 환산 시 약 0.78원
Ada	0.0004달러/1K 토큰	1달러 1,300원으로 환산 시 약 0.52원

표 4-4-2 파인튜닝 모델의 요금(사용)

모델	요금(달러)	대한민국 원화 기준
Davinci	0.1200달러/1K 토큰	1달러 1,300원으로 환산 시 약 156원
Curie	0.0120달러/1K 토큰	1달러 1,300원으로 환산 시 약 15.6원
Babbage	0.0024달러/1K 토큰	1달러 1,300원으로 환산 시 약 3.12원
Ada	0.0016달러/1K 토큰	1달러 1,300원으로 환산 시 약 2.08원



Fine-tuning models

Create your own custom models by fine-tuning our base models with your training data. Once you fine-tune a model, you'll be billed only for the tokens you use in requests to that model.

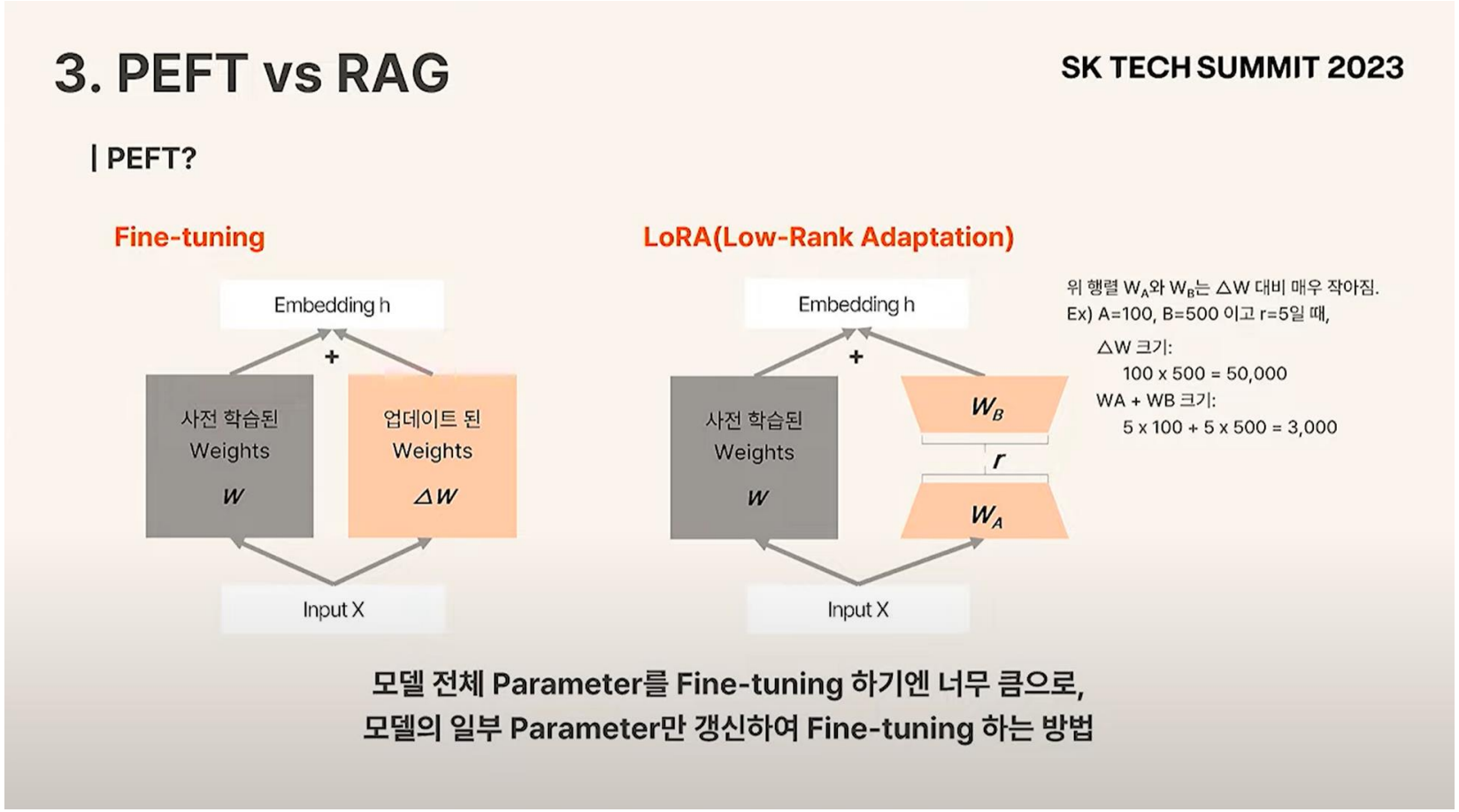
Learn about fine-tuning ↗

Model	Training	Usage
Ada	$0.0004 / 1K tokens	$0.0016 / 1K tokens
Babbage	$0.0006 / 1K tokens	$0.0024 / 1K tokens
Curie	$0.0030 / 1K tokens	$0.0120 / 1K tokens
Davinci	$0.0300 / 1K tokens	$0.1200 / 1K tokens

그림 4-4-1 파인튜닝 모델의 요금

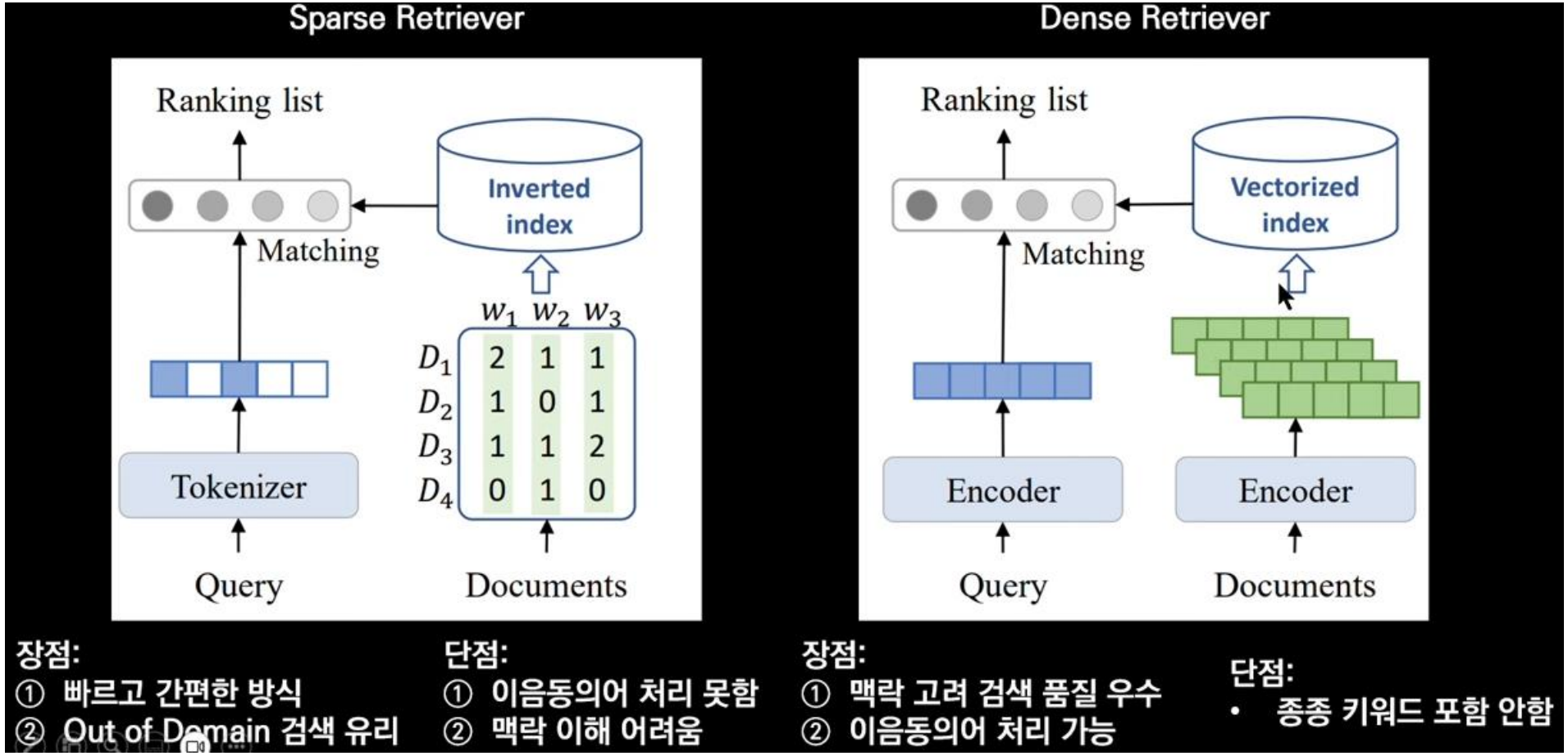
https://ridibooks.com/books/1160000139?_rdt_sid=category_bestsellers&_rdt_idx=1&_rdt_arg=2220

[유첨] LoRA



https://youtu.be/WWaPGDS7ZQs?si=BFyN8MTOmhILhWY6

[유첨] Sparse/Dense Retriever



정리

- sLM은 parameter의 수가 LLM대비 상대적으로 적은 LM
- 학습과 사용에 비용이 적게 들지만, retrieval 품질에 영향을 많이 받는다.
- 다양한 retrieval 기술들이 계속 개발되고 있다

Q&A