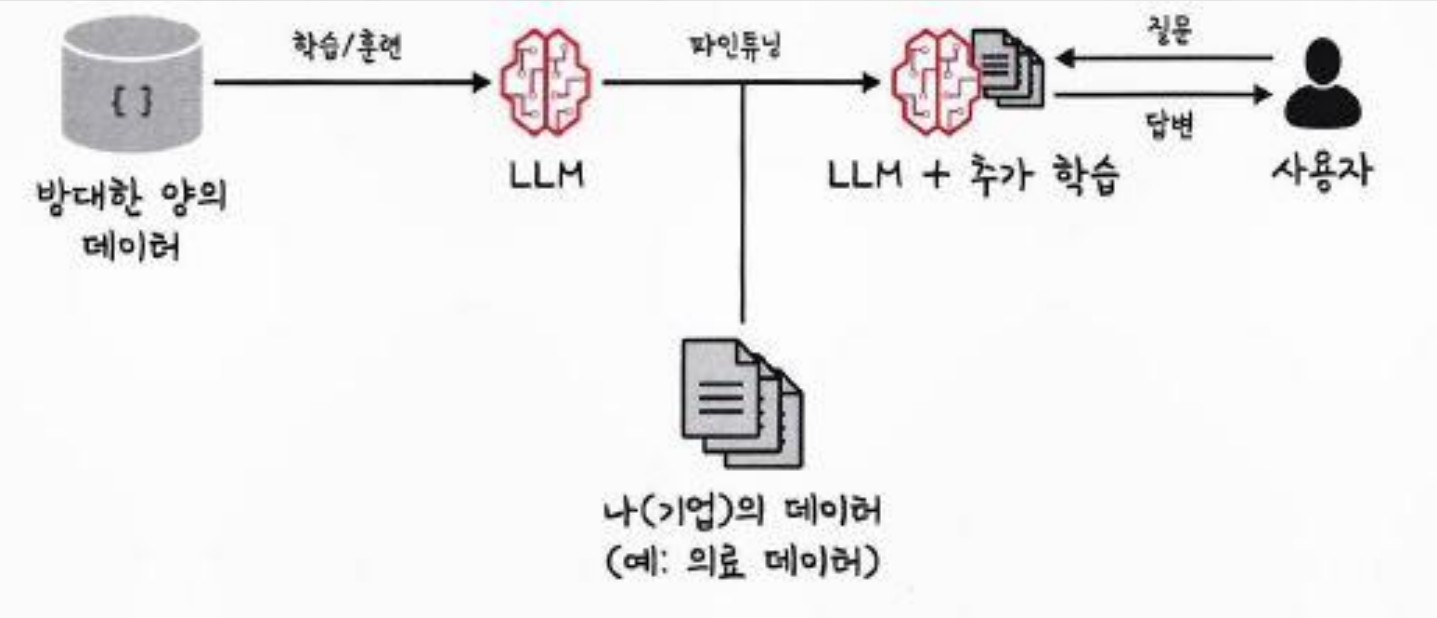
LLM 활용

파인튜닝

파인튜닝

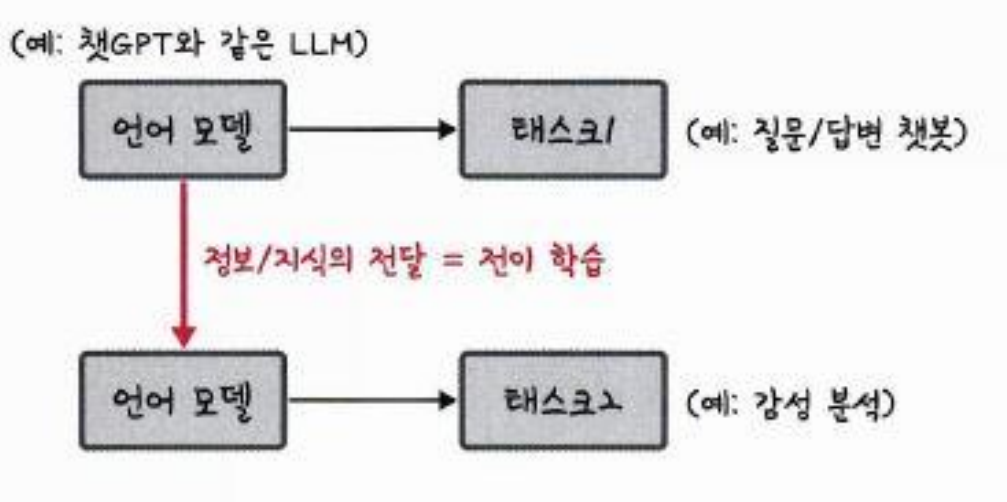
- 파인튜닝(Fine-Tuning)은 기존의 LLM을 특정한 작업이나 상황에 맞게 조금 더 훈련시키는 과정
- 이미 많은 것을 배운 LLM을 특별한 상황에 더 잘 맞게 '가르치는'것



파인튜닝

전이 학습

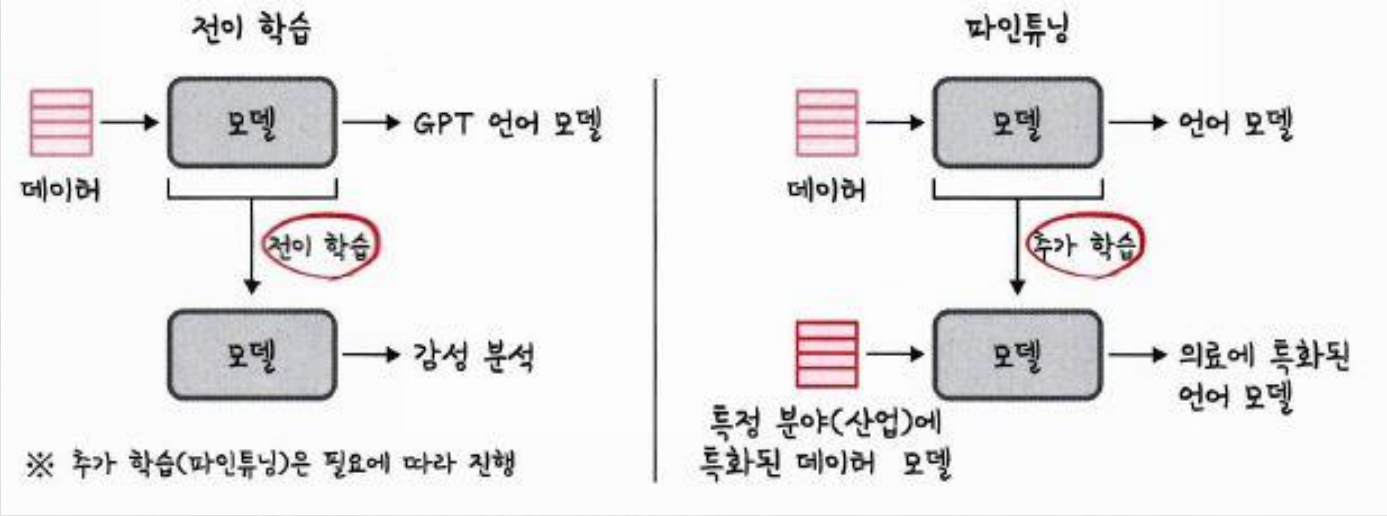
- 한 분야에서 배운 지식을 다른 분야의 문제 해결에 사용하는 방법을 전이 학습(Transfer Learning)
- 다음과 같이 이전 모델에서 학습한 내용이 그대로 다른 작업에도 적용될 수 있도록 정보가 전이되는 형태



파인튜닝

전이 학습

- 전이 학습은 이미 학습된 모델을 새로운 작업에 적용하는 것으로 파인튜닝 보다는 좀 더 포괄적인 의미
- 파인튜닝은 전이 학습의 한 형태로 모델을 특정 분야나 작업에 최적화 시키기 위해 추가적인 학습을 시키는 과정



https://platform.openai.com/docs/guides/fine-tuning/preparing-your-dataset

파인튜닝

파인튜닝 방법 고려 사항

- 이미 만들어진 LLM에 추가 학습만 진행하면 된다고 하지만 어느 정도의 데이터로 어느 정도의 훈련을 시켜야 기대하는 성능(정확도)을 얻을 수 있을지 누구도 확신
- 학습을 위해 과도한 비용이 발생(LLM은 말 그대로 거대 언어 모델이기 때문 에 여기에 추가 학습을 시키기 위해서는 LLM을 만드는 회사 뿐)
- 데이터 준비가 어려움(파인튜닝을 위해서는 데이터를 '질문-답변' 세트 형식으로 준비해야 하는데, 기존에 보유된 데이터가 이런 형식으로 정리 되어 있지 않은 경우가 대부분)

https://platform.openai.com/docs/guides/fine-tuning/preparing-your-dataset

RAG(Retrieval-Augmented Generation)

RAG(Retrieval Augmented Generation)

- 자연어 처리 분야에서 사용되는 기술로, 정보 검색과 생성을 결합한 인공 지능 모델
- RAG는 특히 복잡하고 정보가 필요한 질문에 답변하기 위해 설계
- RAG는 크게 두 단계로 구성
  - 정보 검색(retrieval) 단계와 텍스트 생성(generation) 단계
  - 이런 결합은 모델이 질문에 대해 보다 정확하고 관련성 높은 답변을 생성할 수 있도록

RAG(Retrieval-Augmented Generation)

RAG(Retrieval Augmented Generation)

- 정보 검색단계
  - 질문: 사용자로부터 질문이 입력
  - 쿼리(문서 검색): 모델은 대규모의 문서 데이터베이스나 콘텐츠 저장소에서 질문과 관련 된 문서나 정보를 검색
  - 정보 검색 결과: 검색 결과 중에서 가장 관련성 높은 문서와 사용자의 질문을 결합하여 LLM 에 전달

RAG(Retrieval-Augmented Generation)

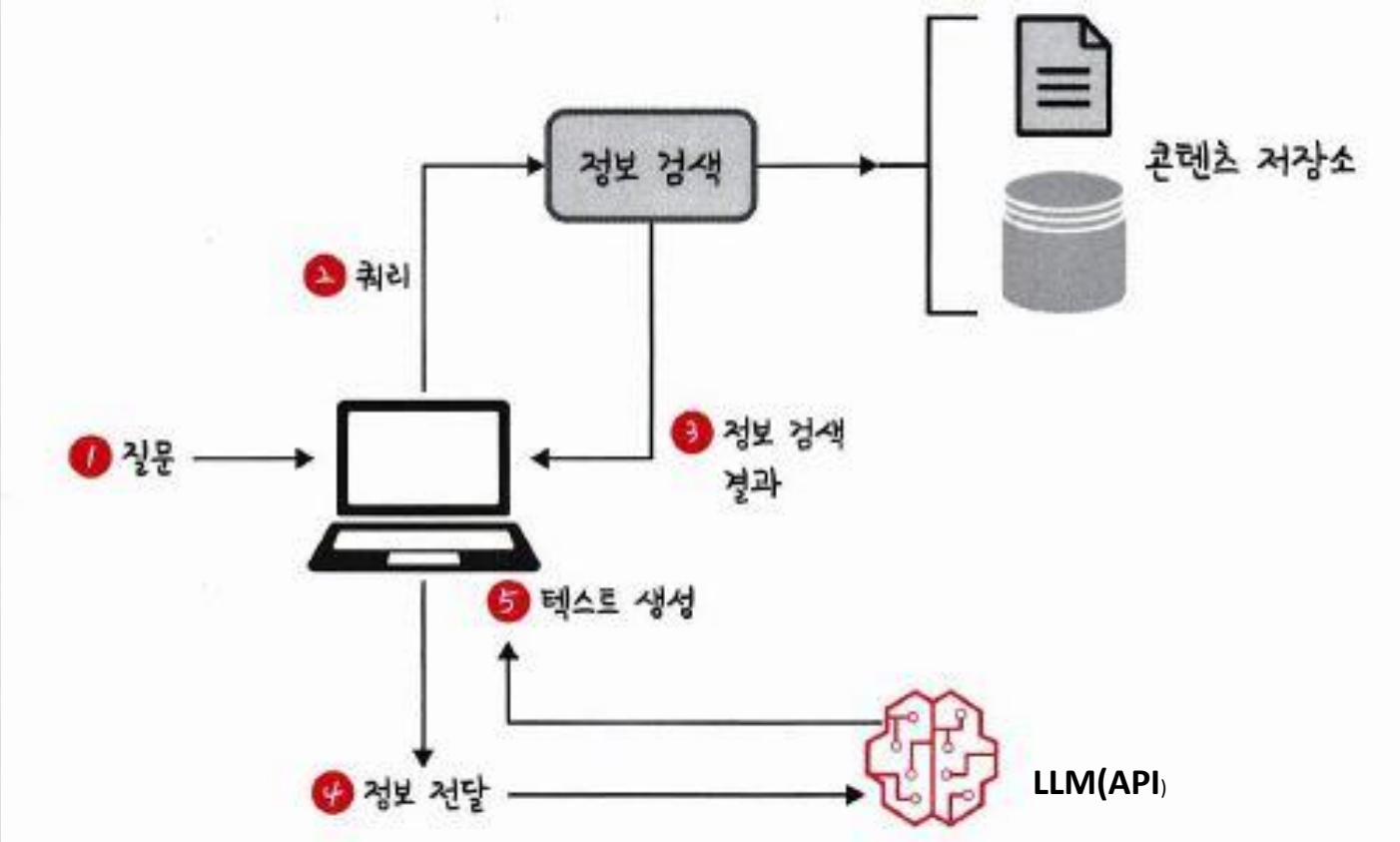
RAG(Retrieval Augmented Generation)

- 텍스트 생성 단계
  - 정보 전달:
    - 선택된 문서의 내용이 모델에 전달
    - 정확히는 사용자의 질문과 정보 검색 결과가 전달
    - 이 단계에서 모델은 문서의 정보를 활용하여 질문에 대한 의미를 이해
  - 텍스트 생성
    - 모델은 이제 전달받은 정보를 바탕으로 질문에 대한 답변을 생성
    - 이 과정은 LLM에 의해 처리되며, 문서에서 얻은 지식과 모델이 이미 학습한 정보를 결합하여 답변을 만듬
    - 이후 생성된 답변을 사용자에게 제공

RAG(Retrieval-Augmented Generation)

RAG(Retrieval Augmented Generation)

- RAG 동작 방식



퓨샷 러닝(Few Shot Learning)

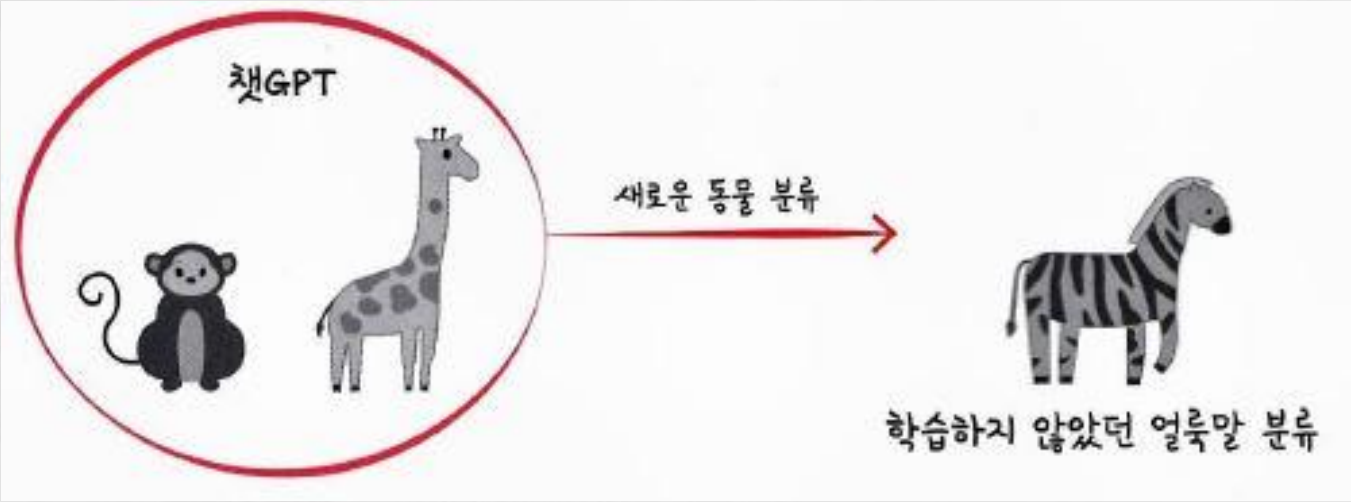
퓨샷 러닝(Few Shot Learning)

- 매우 적은 양의 데이터로 학습하는 능력을 가리킴
- 여기서 중요한 것은 모델이 기존에 학습한 지식을 바탕으로 매우 제한된 예시로부터 새로운 작업에 빠르게 적응하는 것
- 퓨샷 러닝과 함께 나오는 단어로 제로샷 러닝(Zero Shot Learning), 원샷 러닝(One Shot Learning)이 있는데, 데이터의 양에 따라 제로샷, 원샷, 퓨샷으로 나눔

퓨샷 러닝(Few Shot Learning)

퓨샷 러닝(Few Shot Learning)

- 모델이 학습 과정에서 결코 보지 못한 데이터에 대해 예측을 수행할 수 있는 것을 제로샷 러닝(Zero Shot Learning)
- 거대 언어 모델과 같이 학습한 데이 터가 꽤 방대하면서도 모델이 높은 수준의 추상적 사고와 일반화 능력을 갖춰야 함



퓨샷 러닝(Few Shot Learning)

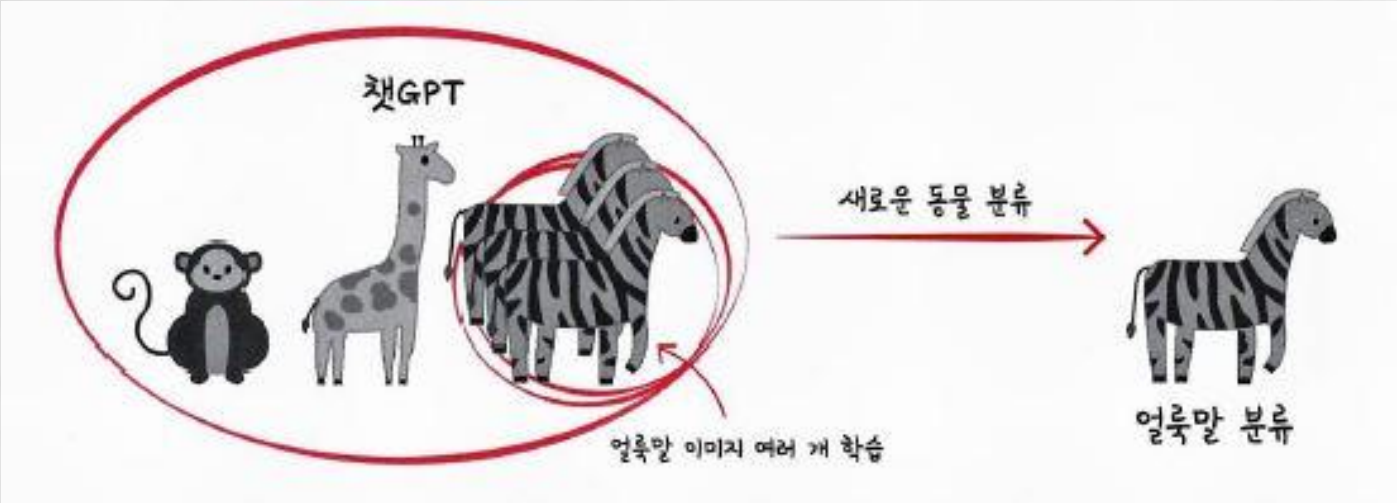
퓨샷 러닝(Few Shot Learning)

- 이미지 한 개만 학습했을 뿐인데도, 잘 분류할 수 있는 것을 원샷 러닝 (One Shot Learning)
- 이 방법은 특히 데이터가 매우 제한적인 상황에서 유용

퓨샷 러닝(Few Shot Learning)

퓨샷 러닝(Few Shot Learning)

- 이미지 여러 개를 학습한 이후 모델이 잘 분류하는 것을 퓨샷 러닝 (Few Shot Learning)



퓨샷 러닝(Few Shot Learning)

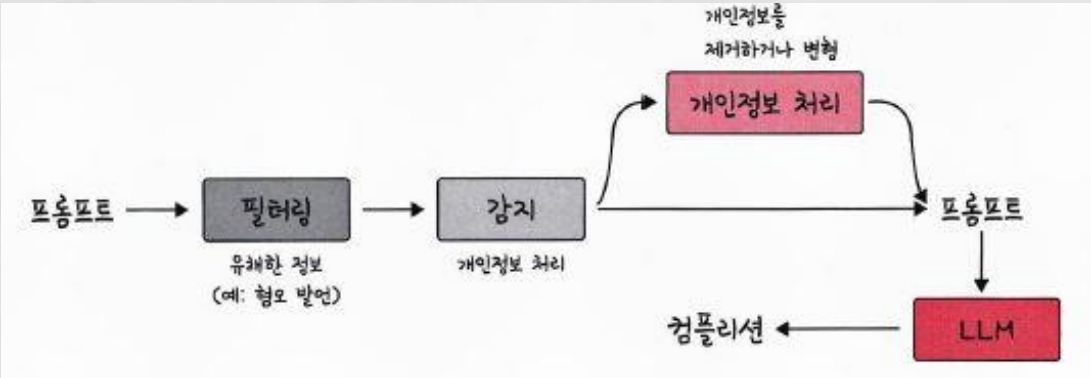
퓨샷 러닝(Few Shot Learning)

- 원샷 러닝이나 퓨샷 러닝은 특정 작업이나 분야에서 충분한 양의 학습 데이터를 확보하기 어려울 때 유용
- 특정한 의료 이미지나 희귀 언어 데이터의 경우 충분한 학습 자료를 얻기 어려울 수 있음
- 이때 원샷/퓨샷 러닝을 사용하면 효과적인 작업을 처리할 수 있음
- 실제로 적은 수의 예제에서 학습된 모델은 종종 새롭고 다양한 데이터를 일반화하는 데 어려움을 겪을 수 있으므로 서비스로 활용할 때는 유의

LLM 활용 시 주의 사항

정보 필터링

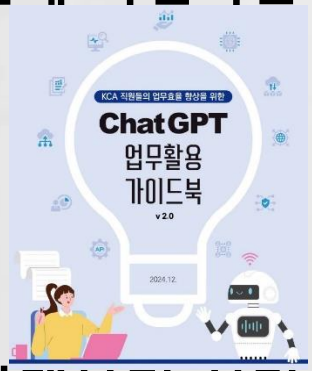
- LLM을 이용하는 사용자의 질문은 **반드시 필터링**
- 기업 내부 직원들이 사용할 경우는 덜하겠지만 일반인을 상대로 서비스 하는 경우, 어떤 내용이 입력될지 알 수 없음
- 따라서 이러한 경우 반드시 입력 및 출력 텍스트를 필터링
- 특히 개인정보가 입력되지 않도록 필터링하는 것이 중요



LLM 활용 시 주의 사항

법적인 규제

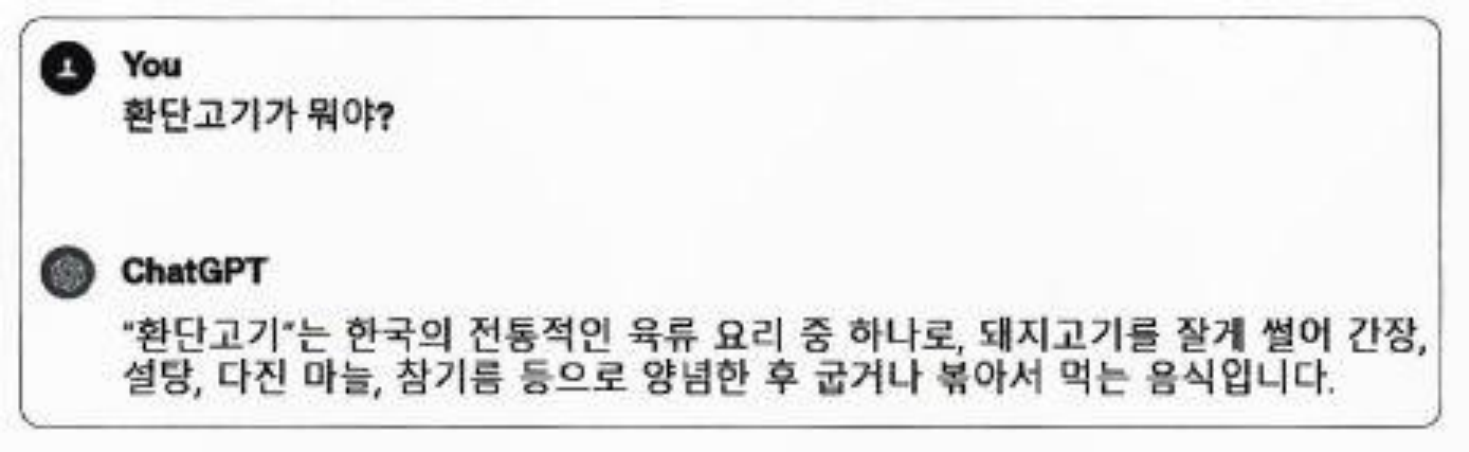
- 특히 공공기관 및 금융산업에 해당
- 기관들은 산업 특성상 국가에서 규정하는 법규 및 권고 사항을 지켜야 함
- 따라서 정부가 정의해 놓은 규제가 어떤 것들이 있는지 사전에 확인하는 것이 필요
- 그뿐만 아니라 기업의 보안팀에서도 정의한 규정들도 있음
- 이것들도 잘 확인해서 준수할 수 있도록 아키텍처 디자인 단계부터 신경써야 함



LLM 활용 시 주의 사항

할루시네이션

- 할루시네이션(hallucination)은 AI 모델, 특히 언어 모델이 부정확하거나 관련 없는 정보를 생성하는 현상
- LLM이 생성한 답변의 부정확한 할루시네이션 현상을 최소화
- 할루시네이션은 정보 검색 결과만 정확하다면 어느 정도 해결할 수 있음



LLM 활용 시 주의 사항

할루시네이션

- LLM이 정확한 답변만 할 수 있도록 temperature라는 파라미터를 0으로 설정
- 정확한 답변이 중요할수록 낮은 값(0), 창의적인 답변이 필요할수록 높은 값(1)

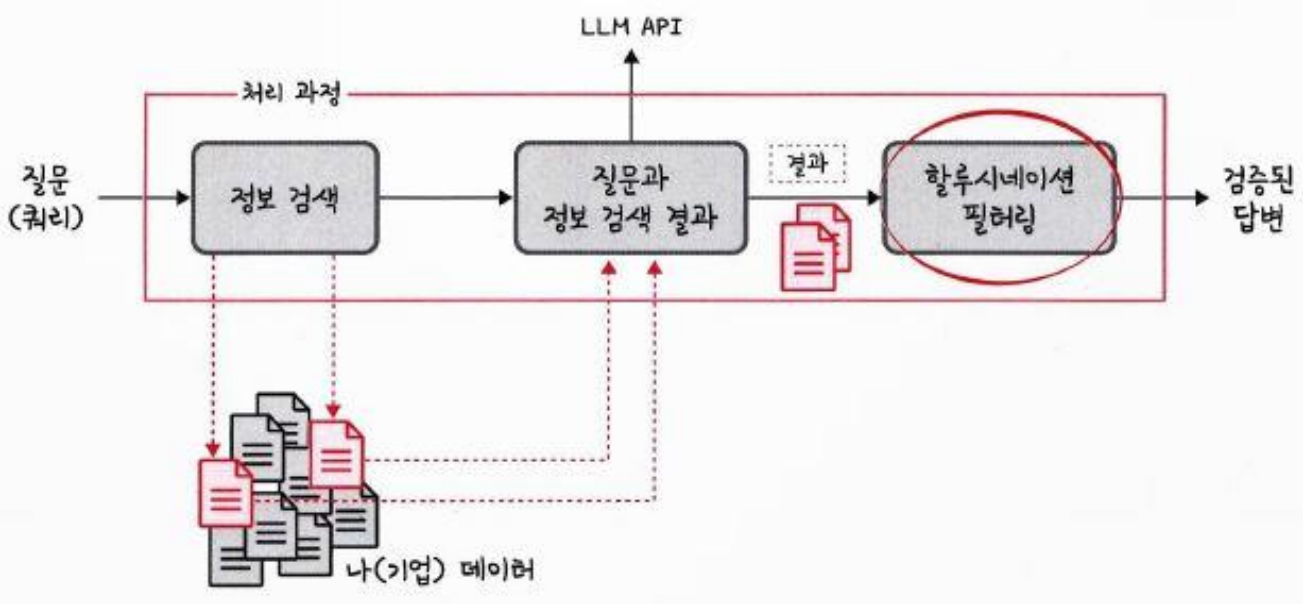
```
from openai import OpenAI
client = OpenAI(
  api_key="sk- " #openai 키 입력
)

prompt = "한국의 야구팀 3개만 알려줘?"
response = client.chat.completions.create(
  model = "gpt-4",
  messages= [{'role': 'user', 'content': prompt}],
  temperature=0,
  max_tokens=300,
  top_p=1,
  frequency_penalty=0,
  presence_penalty=0,
)
print(response.choices[0].message.content)
```

LLM 활용 시 주의 사항

할루시네이션

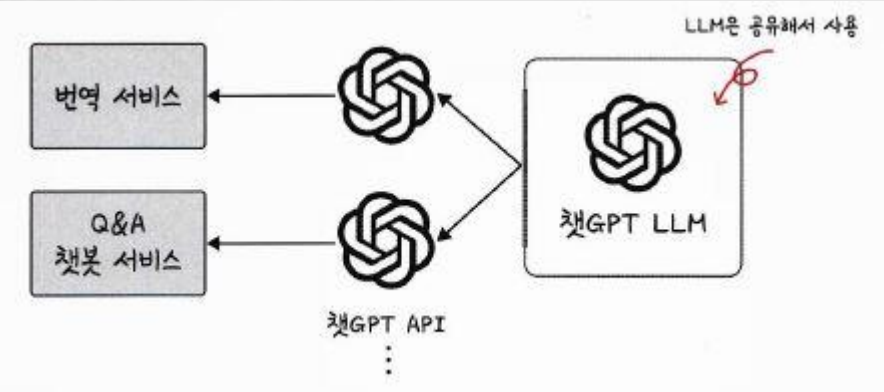
- LLM 구현 과정 중 마지막에 할루시네이션 필터링을 추가함으로써 할루시네이션을 방지로 조정
- 할루시네이션제거를 위한 필터링



LLM 활용 시 주의 사항

보안

- LLM 모델의 경우, 모든 사용자가 모델을 공유해서 사용하다 보니 '내가 입력한 데이터가 학습에 활용되지 않을까?' 혹은 '내 데이터가 LLM까지 넘어가면서 보안에 문제는 없을까?'라는 의문을 가질 수 있음
- 보안(특히 네트워크 측면의 보안)을 강화하기 위해서는 마이크로소프트 애저(Azure) 오픈 AI를 사용하는(정확히는 프라이빗 엔드포인트(private endpoint)) 방법도 고려



LLM의 한계

편향과 공정성

- LLM이 주로 남성 엔지니어의 데이터를 학습했다면, '엔지니어'라는 단어에 대해 남성 이미지를 더 자주 연상시키는 문장을 생성할 수 있음
- 예를 들어 엔지니어가 포함 된 질문을 한다면 남성에 더 우호적인 텍스트를 생성할 가능성이 높음
- 실제로 아마존은 직원 채용 시 'AI 면접' 과정이 있었지만 남성에 더 우호적으로 채점하는 것으로 밝혀지면서 AI 면접을 폐지했다고 함

LLM의 한계

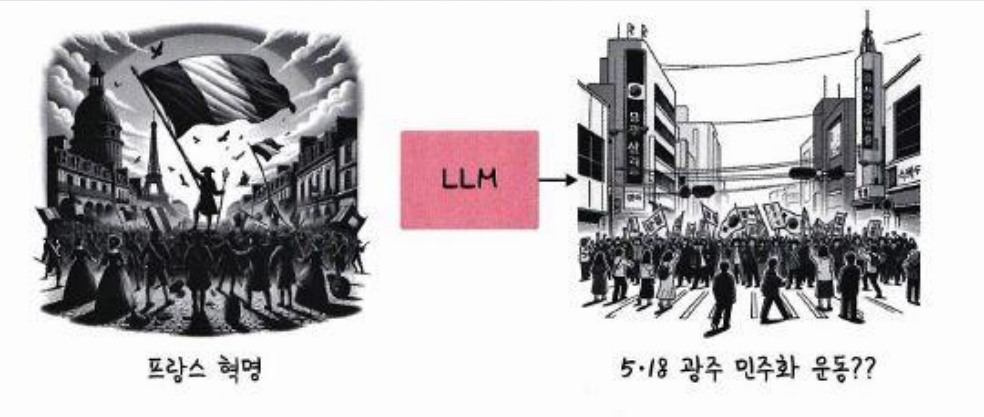
투명성

- LLM이 어떤 질문에 대해 특정 대답을 하는 경우 왜 그런 대답을 했는지 그 이유를 사용자에게 설명하는 능력이 부족
- 분명 학습한 데이터를 기반으로 답변을 한 것 은 맞지만 그렇다고 LLM이 학습한 데이터를 그대로 사용자에게 보여주는 것은 아님
- LLM 자체적으로 어느 정도 가공을 하는데 그 가공 과정을 인간은 이해할 수 없음
- 자율 주행 자동차는 복잡한 딥러닝 알고리즘을 사용하여 주변 환경을 인식하고 결정(언제 브레이크를 밟을지, 언제 우회전할지 등)이며 차량이 특정 상황에서 어떻게 행동할지 예측하기 어렵다는 점

LLM의 한계

데이터 의존성

- LLM은 특정 언어, 특정 분야(예: 문학)에 한정되어서 데이터를 학습하지 않음
- 그렇기 때문에 지금과 같이 다양한 질문에도 답변할 수 있는 것
- 모델이 특정 국가의 소설로만 학습을 할 경우 다른 지역의 문화적 맥락을 반영한 텍스트를 생성하는 데 한계



LLM의 한계

정보의 일반화

- '데이터 의존성'과 반대되는 것으로 LLM이 너무 다양한 데이터를 학습했기 때문에 특정 산업(예: 여행)에 특화된 질문을 할 경우 정밀한 답변을 얻지 못할 수 있음
- 예를 들면 건축에 관한 구체적 질문하면 홈페이지에서 확인하라는 답변

LLM의 한계

오류 가능성와 개인정보보호

- 오류 가능성
  - LLM이 잘못된 정보를 기반으로 문서를 작성할 수 있으며 이는 가짜 뉴스의 확산과 같은 문제를 야기할 수 있음
  - 대표적인 것이 앞서 말한 할루시네이션 현상
- 개인정보보호
  - 학습 데이터에 포함된 개인정보(예: 메신저 대화 내용에 포함된 이름, 위치, 개인적 선호 등)를 모델이 학습
  - 이를 생성 과정에서 노출시킬 수 있음

LLM의 한계

새로운 정보의 결여와 기업 내 데이터 미활용

- 새로운 정보의 결여
  - 모델이 2025년까지의 데이터로 훈련되었다면, 그 이후에 발생한 사건이나 정보에 대해서는 알지 못하고 과거의 정보를 기반으로 응답할 수 있음
- 기업 내 데이터 미활용
  - 기업에서 LLM을 그대로 사용하지 못하는 이유는, LLM을 그대로 기업에서 활용할 경우 기업이 가진 데이터를 활용할 수 없기 때문
  - LLM이 학습한 데이터에 각 기업에서만 가지고 있는 데이터가 포함되어 있지 않음

RAG 개념

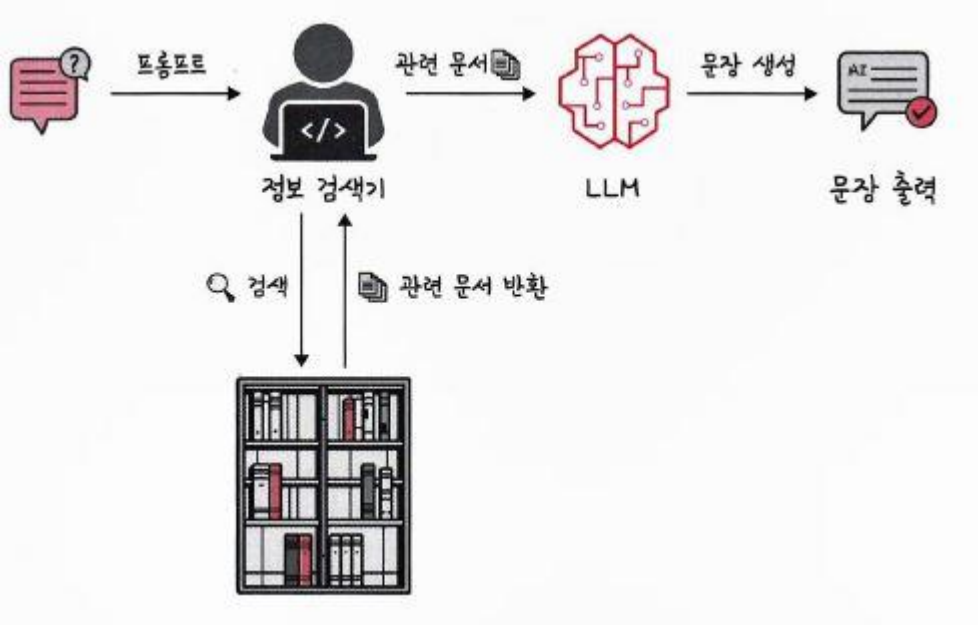
RAG 개념

- RAG(Retrieval-Augmented Generation)는 LLM이 텍스트를 생성할 때 관련 정보를 찾아 보고(retrieval), 그 정보를 활용하여 새로운 텍스트를 만드는(generation) 기술
- 예를 들어 RAG를 사용하는 LLM은 특정 질문에 답하기 위해 인터넷에서 정보를 검색하고, 그 정보를 바탕으로 상세하고 정확한 답변을 생성할 수 있음

RAG 개념

RAG 개념

- RAG는 큰 데이터베이스나 인터넷과 같은 정보의 원천에서 필요한 사실이나 데이터를 찾아내고, 그것을 기반으로 텍스트를 만드는 기술
- 이 방식은 LLM이 더 정확하고 신뢰할 수 있는 내용을 생성하도록 도와줌



RAG 구현 과정

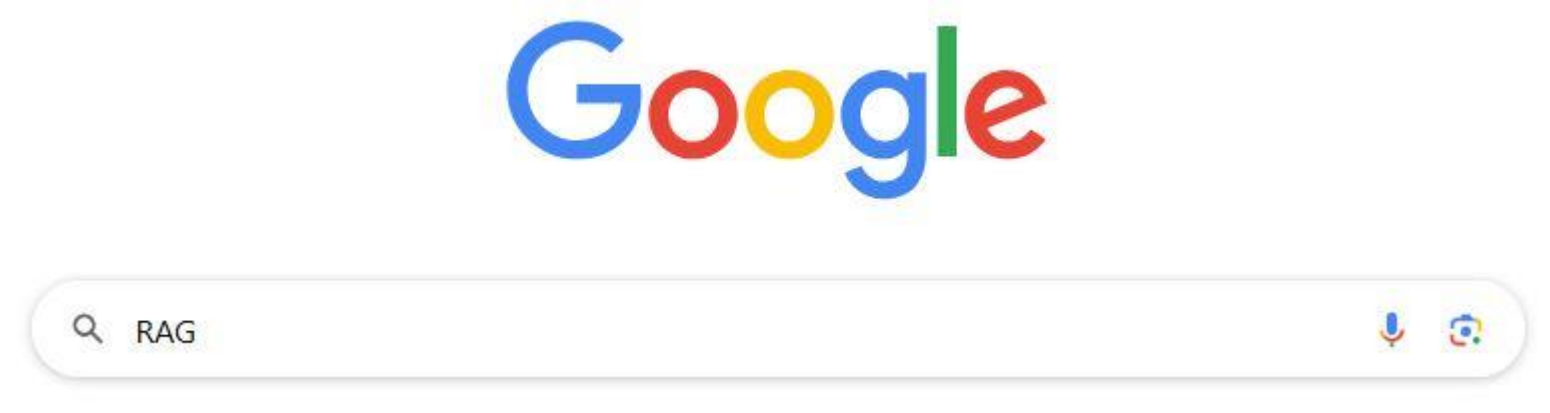
RAG 구현 과정

- RAG는 '정보 검색'과 '텍스트 생성'이라는 두 가지 단계를 결합한 기술
  - 정보 검색(retrieval)은 AI가 대규모 정보로부터 관련 데이터를 찾는 과정
  - 텍스트 생성(generation)은 찾은 정보를 기반으로 새로운 텍스트를 만드는 과정
- 정보 검색
  - 질문 입력
  - 검색
  - 유사도 검색
  - 랭킹 처리

RAG 구현 과정

정보 검색

- 질문 입력
  - 사용자는 필요한 정보를 찾기 위해 구글, 빙이나 네이버에 질문을 하거나 키워드를 입력. 이것을 '쿼리(query)'

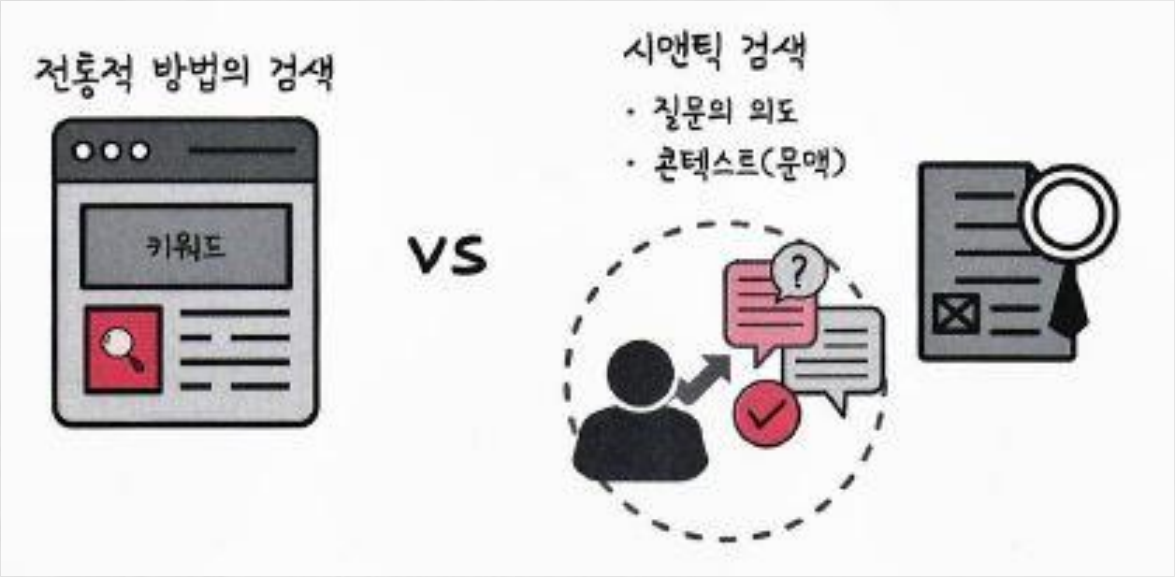


- 검색
  - 검색 엔진은 해당 쿼리와 관련된 정보를 데이터베이스나 인터넷에서 찾음

RAG 구현 과정

정보 검색

- 유사도 검색
  - 이후 검색 엔진은 쿼리와 데이터베이스(혹은 인터넷)에 있는 문서들 사이의 유사도를 계산
  - 이 과정은 키워드 검색과 시맨틱 검색을 모두 포함



RAG 구현 과정

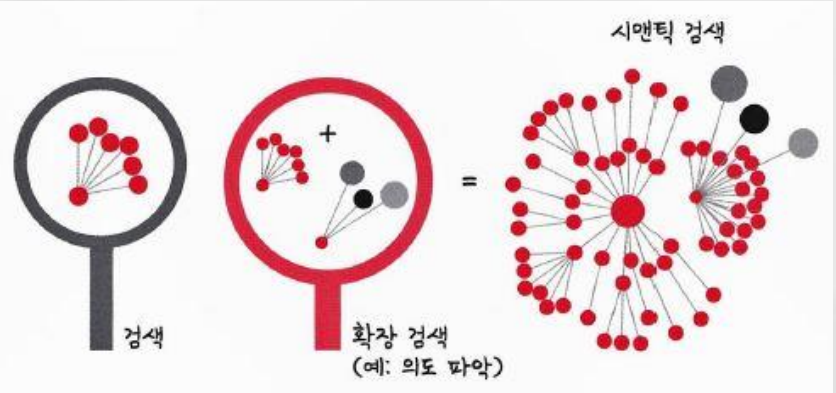
정보 검색

- 키워드 검색(keyword search)
  - 사용자가 입력한 단어나 구를 데이터베이스나 인터넷에서 직접 찾는 방식
  - 즉, 검색 엔진은 사용자가 입력한 키워드가 문서 내에 명시 적으로 나타나는 경우만 해당 문서를 결과로 반환
  - 커피숍'을 검색하면, '커피숍'이라는 단어가 포함된 모든 웹페이지를 찾아서 결과로 보여 줌
  - 하지만 이 방식은 사용자의 질문 의도나 문맥을 파악하지 못하고 단순히 키워드의 존재 여부만을 기준으로 하기 때문에 때때로 관련 없는 결과를 보여 주기도 함

RAG 구현 과정

정보 검색

- 시맨틱 검색(semantic search)
  - 단어의 의미와 문맥을 이해하여 보다 관련성 높은 결과 를 제공하는 기술
  - 이 방식은 단어의 의미, 동의어, 주제, 사용자의 검색 의도 등을 고려하기 때문에 단어가 문서에 직접적으로 나타나지 않더라도 문맥상 관련 있는 결과를 찾아낼 수 있음
  - '가장 가까운 커피숍'을 검색하면, 시맨틱 검색은 '가까운'의 의미를 이해하고 사용자의 위치를 고려하여 주변의 커피숍을 찾아서 보여 줌
  - 시맨틱 검색은 키워드 검색보다 훨씬 복잡하며 의미를 정확히 파악하고 문맥을 이해하기 위해 고도의 알고리즘과 자연어 처리 기술 필요



RAG 구현 과정

정보 검색

- 랭킹 처리 – 유사도 계산
  - 검색 결과로 찾아낸 문서들 중에서 어떤 것이 질문과 가장 관련이 높은지를 결정
  - 가장 관련이 높다고 판단되는 문서부터 순서대로 나열
  - 모델이 생성할 텍스트와 가장 관련이 높은 정보를 선택하는 과정
  - 이 과정의 목표는 사용자의 질문이나 요구에 가장 적합하고 유용한 정보를 찾는 것
  - RAG에서 랭킹을 매기는 원리
    - 유사도 계산
    - 문맥과 의도 파악
    - 랭킹 산출

RAG 구현 과정

정보 검색

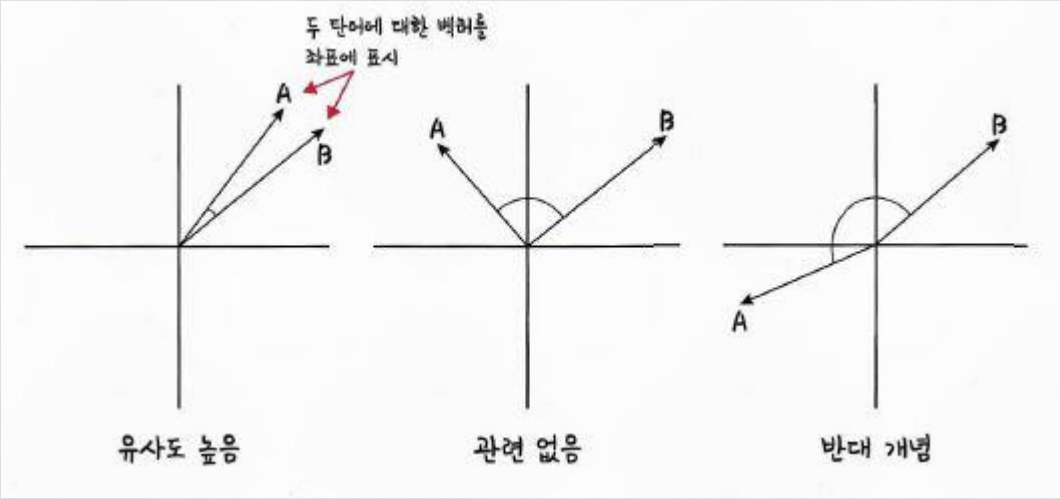
- 랭킹 처리 – 유사도 계산
  - 문서나 단어 사이의 관련성이나 유사성을 수치로 표현하는 방법
  - 예를 들어 우리가 '사과'와 '수박'이라는 단어를 생각해보면, 이 두 단어는 서로 관련성이 있음
  - 검색엔진은 이러한 연관성을 이해하기 위해 유사도 계산
  - 예를 들어 '사과'라는 단어는 [1,0] 이라는 벡터로, '수박'이라는 단어는 [1, 1]이라는 벡터로 표현
  - 여기서 첫 번째 숫자는 '과일'과 관련된 정도를, 두 번째 숫자는 '단맛'과 관련된 정도를 나타냄

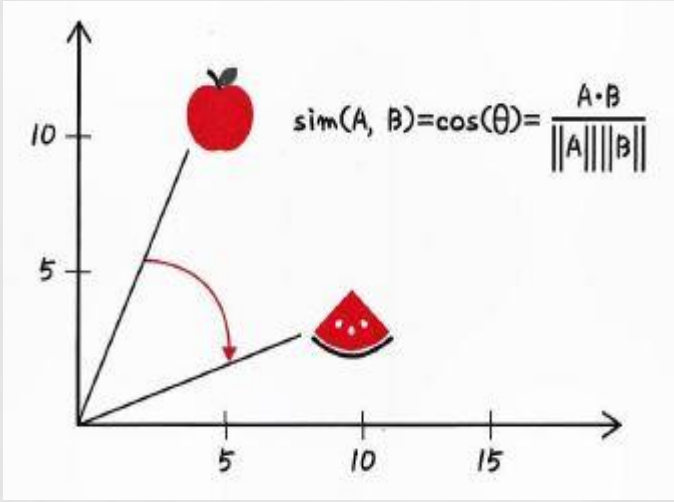
RAG 구현 과정

정보 검색

- 랭킹 처리 – 유사도 계산
  - 두 번째 숫자는 '단맛'과 관련된 정도를 나타냄
  - 그런 다음, 이 벡터들 사이의 거리나 각도를 계산하여 두 단어가 얼마나 유사한지 측정

    (벡터는 좌표에 표현할 수 있기 때문에 각도 계산이 가능)
  - 벡터가 가리키는 방향이 비슷하거나 거리가 가까울수록 두 단어가 유사하다는 의미





RAG 구현 과정

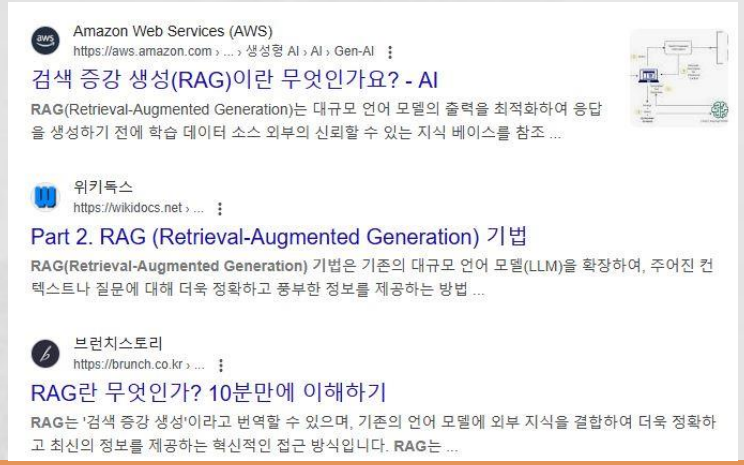
정보 검색

- 랭킹 처리
  - 문맥과 의도 파악
    - 모델은 쿼리의 문맥과 의도를 고려하여 검색된 문서가 얼마나 관련이 있는지 판단
  - 랭킹 산출
    - 유사도, 문맥, 정보 품질 등 다양한 요소를 종합하여 각 문서에 최종 랭킹을 매김

RAG 구현 과정

텍스트 생성

- (검색 엔진의 경우)결과 반환
  - 랭킹이 매겨진 문서 리스트를 사용자에게 보여 줌
  - 사용자는 이 리스트를 보고 원하는 정보를 선택할 수 있음
  - 일반적으로 검색 결과를 보여주면 첫 번째부터 클릭해보다가 원하는 정보가 없으면 페이지를 넘기면서 모든 결과를 클릭
  - 하지만 '랭킹 처리'에서 페이지를 넘길수록 원하는 정보를 찾지 못할 확률이 더 높음



RAG 구현 과정

텍스트 생성

- (LLM의 경우)텍스트 생성
  - 사용자의 질문과 검색 결과로 텍스트를 생성
  - 정보 검색을 통해 수집된 정보를 LLM에 넘기면 LLM에서는 텍스트를 생성
  - LLM은 제공된 정보를 활용하여 구체적이고 정보에 기반한 텍스트를 생성

RAG 구현 시 필요한 것

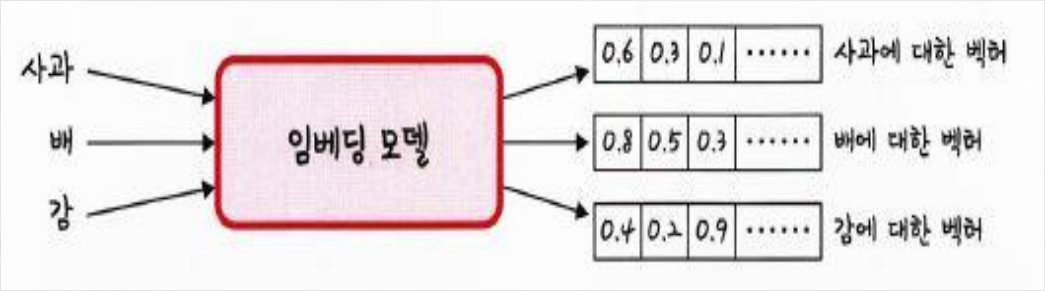
데이터

- 획득한 데이터는 크게 두 가지 방법으로 사용
- 시맨틱 검색
- 백터 검색
  - 임베딩
  - 임베딩 모델
  - 오픈 AI 임베딩 모델

RAG 구현 시 필요한 것

임베딩이란

- 임베딩(embedding)은 복잡한 데이터를 간단한 형태로 바꾸는 것
- 컴퓨터가 이해할 수 있도록 정보를 숫자(벡터)로 바꾸는 것
- 변환되는 첫 번째 숫자는 '과일 (fruit)'을, 두 번째 숫자는 '기술 (technology)'을 나타냄
  - '사과'는 [1.0, 0.이으로 변환
  - '바나나'는 [0.9, 0.1]로 변환
  - '컴퓨터'는 [0.1, 0.이로 변환



RAG 구현 시 필요한 것

임베딩모델

- 임베딩을 위해서는 임베딩 모델이라는 것이 필요
- Word2Vec
  - Word2Vec은 단어를 컴퓨터가 이해할 수 있는 숫자인 벡터로 변환하는 모델
  - 단어 사이의 관계를 이해하고 비슷한 의미를 가진 단어들을 찾을 수 있음

```
from gensim.models import Word2Vec

#훈련에 사용된 데이터
training_data = [
    ['강아지', '고양이', '두', '마리', '계단', '위', '앉아', '있다']
]
#word2vec 사용하여 벡터로 변환
word2vec_model = Word2Vec(sentences=training_data, min_count=1)

word_vector = word2vec_model.wv['강아지']  #강아지를 벡터로 변환
word_vector
```

```
array([ 8.1681199e-03, -4.4430327e-03, 8.9854337e-03, 8.2536647e-03,
-4.4352221e-03, 3.0310510e-04, 4.2744912e-03, -3.9263200e-03, -5.5599655e-
03, -6.5123225e-03, -6.7073823e-04, -2.9592158e-04, 4.4630850e-03,
-2.4740540e-03, -1.7260908e-04, 2.4618758e-03, 4.8675989e-03, -3.0808449e-
05, -6.3394094e-03, -9.2608072e-03, 2.6657581e-05, 6.6618943e-03,
1.4660227e-03, -8.9665223e-03, -7.9386048e-03, 6.5519023e-03, -3.7856805e-
03, 6.2549924e-03, -6.6810320e-03, 8.4796622e-03, -6.5163244e-03,
3.2880199e-03, -1.0569858e-03, -6.7875278e-03, -3.2875966e-03, -1.1614120e-
03, -5.4709399e-03, -1¦2113475e-03, -7.5633135e-03, 2.6466595e-03,
9.0701487e-03, -2.3772502e-03, -9.7651005e-04, 3.5135616e-03, 8.6650876e-03,
-5.9218528e-03, -6.8875779e-03, -2.9329848e-03, 9.1476962e-03, 8.6626766e-
04, -8.6784009e-03, -1.4469790e-03, 9.4794659e-03, -7.5494875e-03,
-5.3580985e-03, 9.3165627e-03, -8.9737261e-03, 3.8259076e-03, 6.6544057e-04,
6.6607012e-03, 8.3127534e-03, -2.8507852e-03, -3.9923131e-03, 8.8979173e-03,
2.0896459e-03, 6.2489416e-03, -9.4457148e-03, 9.5901238e-03, -1.3483083e-
03, -6.0521150e-03, 2.9925345e-03, -4.5661093e-04, 4.7064926e-03,
-2.2830211e-03, -4.1378425e-03, 2.2778988e-03, 8.3543835e-03, -4.9956059e-
03, 2.6686788e-03, -7.9905549e-03, -6.7733466e-03, -4.6766878e-04,
-8.7677278e-03, 2.7894378e-03, 1.5985954e-03, -2.3196924e-03, 5.0037908e-03,
9.7487867e-03, 8.4542679e-03, -1.8802249e-03, 2.0581519e-03, -4.0036892e-03,
-8.2414057e-03, 6.2779556e-03, -1.9491815e-03, -6.6620467e-04, -1.7713320e-
03, -4.5356657e-03, 4.0617096e-03, -4.2701806e-03], dtype=float32)
```

RAG 구현 시 필요한 것

임베딩모델

- Glove
  - GloVe(Global Vectors for Word Representation)는 단어의 의미를 숫자 벡터로 변환하는 방법 중 하나
  - 전체 텍스트에서 단어들이 얼마나 자주 함께 나타나는지를 보고 이 정보를 사용해서 각 단어를 벡터로 표현

```
#gensim은 자연어 처리를 위한 파이썬 라이브러리로, 문서 유사성 분석을 위해 사용됩니다.
!pip install genism

from gensim.models import KeyedVectors
from gensim.scripts.glove2word2vec import glove2word2vec

#사전에 구글 드라이브에 'glove.6B.100d.txt' 파일을 업로드해야 합니다. 업로드 방법
은 부록을 참조해주세요. 또는 https://nlp.stanford.edu/projects/glove 사이트에서
'glove.6B.zip' 파일을 내려받으면 됩니다.
glove_path = '/content/sample_data/glove.6B.100d.txt'

with open(glove_path, 'w') as f:
    f.write("cat 0.5 0.3 0.2\n")
    f.write("dog 0.4 0.7 0.8\n")

#GloVe 파일 형식을 word2vec 형식으로 변환
word2vec_output_file = glove_path + '.word2vec'
glove2word2vec(glove_path, word2vec_output_file)

model = KeyedVectors.load_word2vec_format(word2vec_output_file,
binary=False)
cat_vector = model['cat']  #'cat'에 대한 벡터
cat_vector
```

```
array([0.5, 0.3, 0.2], dtype=float32)
```

RAG 구현 시 필요한 것

임베딩모델

- 오픈 AI 임베딩 모델
  - 다른 모델로 오픈AI에서 제공하는 임베딩 모델
  - 오픈AI 임베딩은 한국어 지원은 물론, RAG에서 정보 검색과 랭크에 있어서도 우월한 성능을 자랑

```
from openai import OpenAI

client = OpenAI(
    api_key = "sk-",  #openai 키 입력
)
document = ['제프리 힌튼','교수','토론토 대학','사임']
#'제프리 힌튼','교수','토론토 대학','사임'을 벡터로 변환
response = client.embeddings.create(
    input=document,
    #오픈AI에서 제공하는 임베딩 모델
    model="text-embedding-ada-002"
)
response
```

```
CreateEmbeddingResponse(data=[Embedding(embedding=[-0.025099867954850197,
-0.019271383062005043, -0.007503656204789877, -0.01672401651740074,
-0.007787466049194336, 0.0280348788946867, -0.005880402401089668,
0.0039110383950173855, -0.00293500954285264, 0.006724909879267216,
--중간 생략--
0.006888267584145069, -0.01740266941487789], index=3, object='embedding')],
model='text-embedding-ada-002-v2', object='list', usage=Usage(prompt_
tokens=23, total_tokens=23))
```

RAG 구현 시 필요한 것

백터 데이터 베이스

- 벡터 데이터베이스는 말 그대로 벡터를 저장하는 저장소
- 벡터는 여러 숫자로 이루어진 데이터의 나열을 말함
- 상품에 대한 사람의 선호도를 여러 숫자로 표현한 것이 벡터
- 숫자의 나열을 저장하는 곳 이 벡터 데이터베이스
- 벡터 데이터베이스는 단순히 벡터를 저장하는 것 외에도 데이터를 관리하며, 검색
- 즉, 일반 데이터베이스와 달리 벡터 데이터베이스는 데이터의 정확한 값 대신 데이터 간의 '유사성'을 바탕으로 검색하는 데 사용

RAG 구현 시 필요한 것

백터 데이터 베이스

- 사용자 질문: “배송 상태를 어떻게 확인할 수 있죠?“
- 임베딩 처리 과정을 거쳐 벡터로 변환
- 벡터 데이터베이스에 다음과 같이 질의

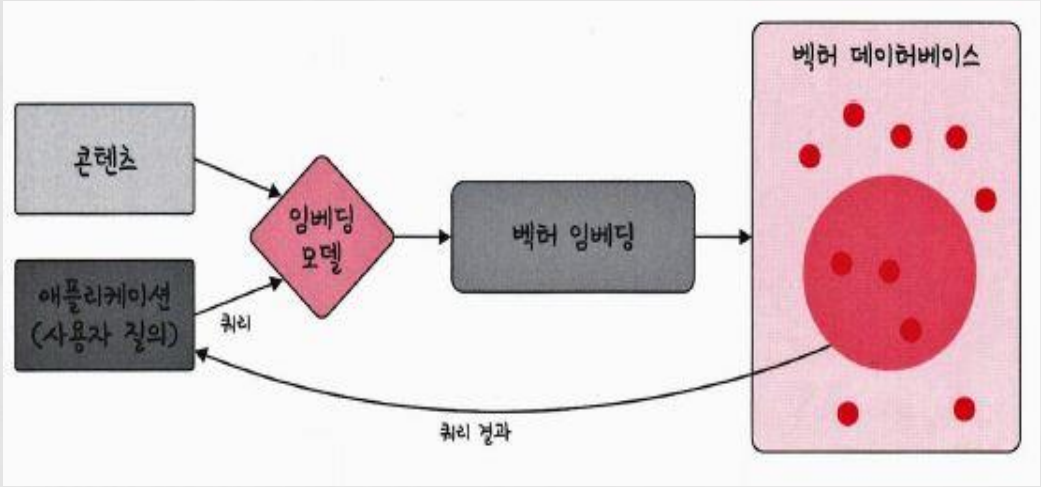
```
POST /search
{
  "query_vector": [0.13, -0.24, 0.33, ..., 0.78],
  "top_k": 5
}
```

- "query_vector''는 질문의 벡터 표현이고, "top_k"는 반환 받고자 하는 가장 유사한 문서의 수

RAG 구현 시 필요한 것

백터 데이터베이스

- 벡터 데이터베이스는 저장된 문서 벡터들과 질문 벡터 간의 유사성을 계산
- 유사성 검사는 코사인 유사도나 유클리드 거리와 같은 것들을 사용
- 유사성 검사 결과에 따라 점수가 가장 높은(랭크가 가장 높은) 상위 N개의 벡터들이 반환되고 사용자



RAG 구현 시 필요한 것

백터 데이터베이스

- 파인콘(Pinecone)
  - 파인콘(Pinecone)은 머신러닝과 인공지능 애플리케이션을 위해 설계된 벡터 데이터베이스
  - 복잡한 데이터(예: 텍스트, 이미지, 소리 등)를 숫자인 벡터로 바꾸어 저장
  - 이 벡터들 사이의 유사성을 기반으로 빠르고 정확하게 검색할 수 있게 해주는 데이터베이스
  - 파인콘을 사용함으로 개발자는 복잡한 벡터 검색 기능을 빠르고 손쉽게 구현

RAG 구현 시 필요한 것

백터 데이터베이스

- 밀버스(Milvus)
  - 밀버스(Milvus)는 클라우드에서 사용 가능한 오픈 소스
  - 밀버스는 다양한 유형의 벡터 데이터를 다루도록 최적화
  - 예를 들어 이미지, 텍스트, 오디오 파일을 벡터 형태로 변환하여 밀버스에 저장하고 유사성 검색을 통해 관련 데이터를 빠르게 찾아낼 수 있음
  - 밀버스는 특정 클라우드 제조사(예: AWS, 마이크로소프트)에 종속적이지 않기 때문에 유연성이 높은 반면 사용자가 직접 인프라를 설정하고 관리

RAG 구현 시 필요한 것

백터 데이터베이스

- 쿼드런트(Qdrant)
  - 쿼드런트(Qdrant)는 오픈적으로 저장하고 검색할 수 있는 기능을 제공
  - 여기서 '차원'은 벡터 내의 개별 요소 의 수를 의미
  - 각 요소는 데이터의 특성을 나타냄
  - 사람의 프로필 데이터에는 나이, 키, 몸무게, 취미, 선호하는 음식 등 수많은 특성이 있을 수 있는데 이러한 개별적인 것들이 하나의 차원을 나타냄
  - 프로필에 나이, 키, 몸무게만 있다면 이것은 3차원 벡터
  - 고차원 벡터를 다루는 데 효율적이라고 했으니 수십, 수백 차원의 데이터를 다루는 데 효과적인 것

RAG 구현 시 필요한 것

백터 데이터베이스

- 크로마(Chroma)
  - 크로마(Chroma)는 주로 LLM을 위해 설계된 오픈 소스 벡터 데이터베이스
  - 크로마는 모델이 생성하는 텍스트 데이터의 벡터를 저장하고 검색하는데 특화
  - 크로마는 텍스트 데이터와 언어 모델에 특화되었기 때문에 다른 유형의 데이터 (예: 이미지, 오디오)를 다루는 데는 다른 벡터 데이터베이스만큼 효과적이지 않음

RAG 구현 시 필요한 것

백터 데이터베이스

- 엘라스틱서치(Elasticsearch)
  - 엘라스틱서치(Elasticsearch)는 강력한 검색과 데이터 분석 기능을 제공하는 검색 엔진
  - 원래는 텍스트 기반 검색에 특화된 기능을 제공했지만 최근에는 벡터 데이터를 처리하는 기능도 추가
  - 설정과 관리가 복잡할 수 있으며 대규모의 시스 템에서는 전문 지식이 필요

RAG 구현 시 필요한 것

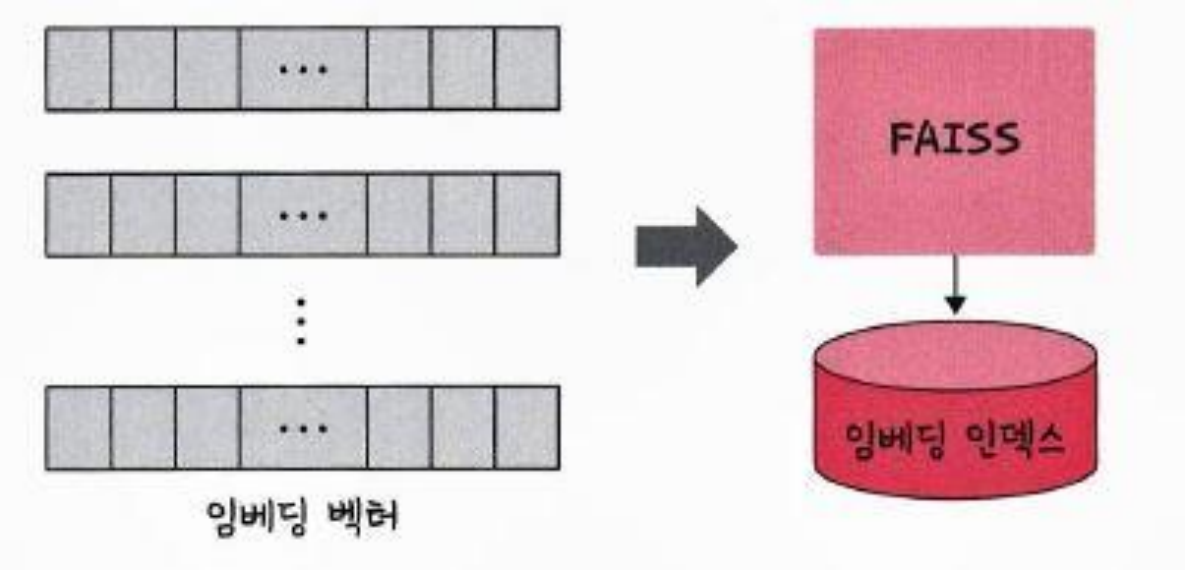
백터 데이터베이스

- 파이스(FAISS)
  - 파이스(FAISS)는 Facebook AI Research(FAIR)에서 개발한 라이브러리로 이미지를 비롯 해 다양한 데이터를 벡터로 바꾸고
  - 이 벡터들 중에서 비슷한 것들을 신속하게 찾아낼 수 있음(Find in a Set of Images)
  - 검색 엔진의 경우 빠른 검색을 위해 인덱스라는 것을 사용
  - 인덱스는 책에 있는 색인과 비슷한 역할
  - 파이스 역시 벡터 인덱스 기능을 제공
  - 색인을 통해 원하는 내용이나 단어를 빠르게 찾을 수 있듯이 벡터 인덱스도 많은 양의 데이터 중에서 원하는 정보를 신속하게 찾는 데 도움

RAG 구현 시 필요한 것

백터 데이터베이스

- 파이스(FAISS)
  - GPU를 사용하여 더 빠른 검색 성능을 제공하지만 효과적인 사용을 위해서는 인덱스 구조에 대한 사전 지식이 필요
  - 파이스를 잘 다루기 위해서는 인덱스와 관련된 지식을 사전에 학습해뒤어야 함



RAG 구현 시 필요한 것

백터 데이터베이스

- 벡터 데이터베이스의 특징 및 장단점 비교

데이터베이스	특징	장점	단점
파인콘	• 간단한 API • 빠른 검색 성능	• 클라우드 기반으로 쉬운 확장성 • 높은 가용성 및 보안성	• 제어에 제한이 있을 수 있음
밀버스	• 오픈 소스 • 고성능 • 광범위한 AI 애플리케이션 지원	• 무료로 사용 가능 • 높은 수준의 제어가 가능	• 관리와 유지보수 필요
쿼드런트	• 오픈 소스 • 고성능 • 유연한 데이터 모델링 및 고급 필터링 기능	• 벡터 및 스칼라 데이터 모두 지원 • 복잡한 검색 쿼리 가능	• 커뮤니티 지원이 밀버스나 엘라스틱서치에 비해 제한적
크로마	• LLM을 위한 벡터 데이터베이스	• 텍스트 데이터와 언어 모델에 특화된 기능을 제공	• 이미지나 오디오 데이터 같은 다른 유형의 벡터 데이터 처리에는 덜 최적화되어 있음
엘라스틱서치	• 널리 사용되는 검색 엔진 • 벡터 검색과 전통적인 텍스트 검색을 모두 지원	• 다양한 플러그인 및 통합 옵션	• 벡터 검색에 대해 다른 전문 벡터 데이터베이스만큼 강력하지 않을 수 있음 • 데이터 사이즈가 커지면 그에 따라 리소스 사용량이 높을 수 있음
파이스	• 오픈 소스 • 고성능	• 무료로 사용 가능 • GPU와 결합하여 빠른 검색 기능 제공	• 인덱스에 대한 사전 지식이 필요

RAG 구현 시 필요한 것

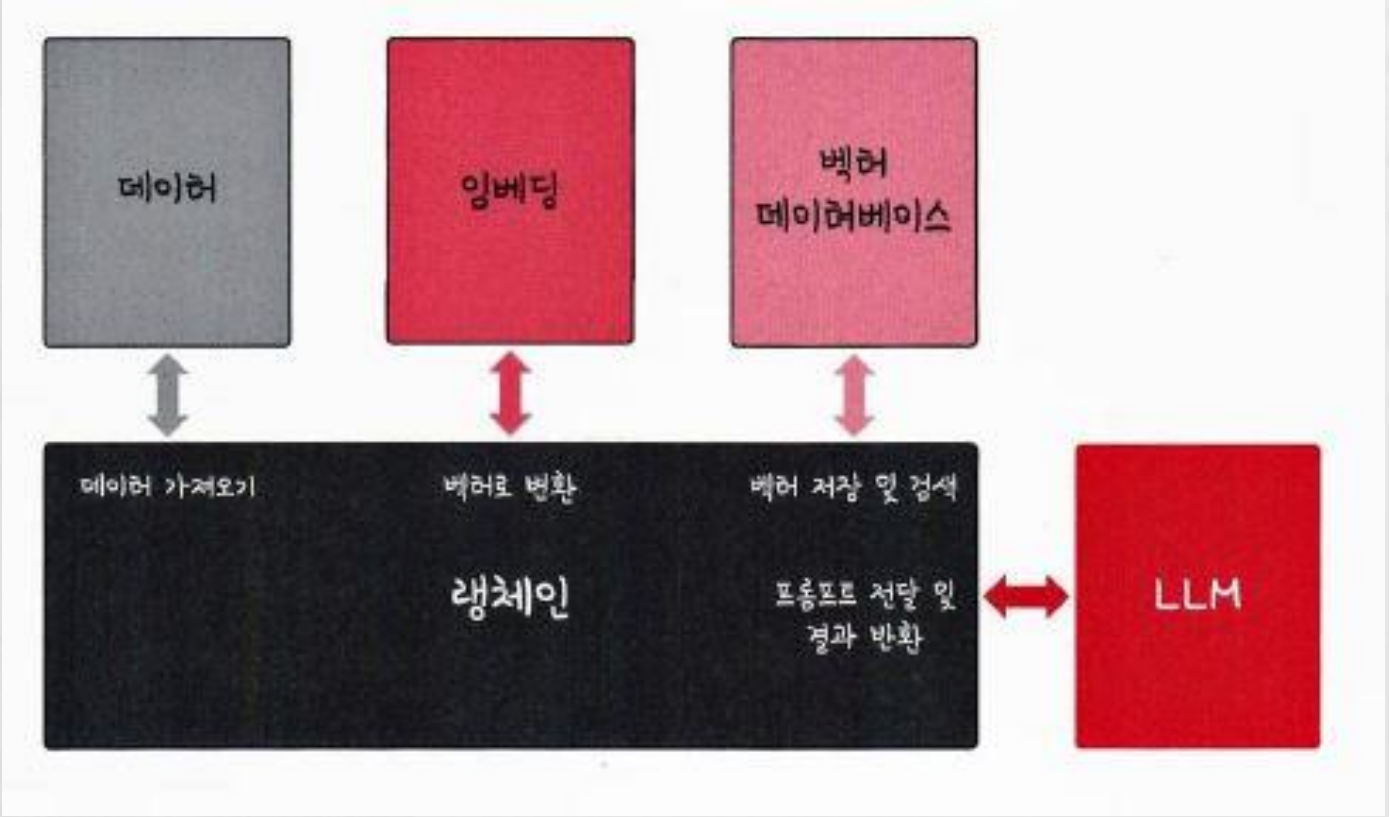
프레임워크(랭체인)

- 실제로 LLM을 이용하여 서비스를 개발하는 것
- 랭체인(LangChain)은 언어 모델을 위한 프레임워크
- 프레임워크는 마치 레고 블록과도 같음
- 레고 블록에는 여러 블록들과 조립 설명서가 들어 있어서 복잡한 모형을
- 빠르게 만들 수 있음
- 프레임워크도 컴퓨터 프로그램을 만들 때 필요한 많은 기본적인 부품들이 미리 준비되어 있고 어떻게 조립해야 하는지에 대한 가이드가 제공

RAG 구현 시 필요한 것

프레임워크(랭체인)

- 랭체인은 LLM을 활용하여 손쉽게 서비스를 개발할 수 있는 도구



랭체인 개념

랭체인 개념

- 랭체인은 LLM을 활용하기 위해 필요한 모듈(라이브러리)
- 랭체인은 챗GPT와 거의 같은 시기에 등장
- 개발자인 해리슨 체이스(Harrison Chase)는 2022년 10월 말, LLM 열풍이 일어나기 시작한 시점에 랭체인을 처음 오픈 소스로 선보임
- 앵무새는 언어 모델을 상징적으로 나타내는 것
- 앵무새가 인간의 언어를 따라서 말 할 수 있다는 점 때문에 랭체인의 상징처럼 표현
- 사슬(chain)은 언어 모델과 언어 모델을 활용할 수 있는 다양한 도구를 결합시킨다는 의미로 표현

랭체인 개념

랭체인 개념

- RAG를 구현하려면 정보 검색과 텍스트 생성이 필요
- 텍스트 생성은 LLM 몫이기 때문에 우리가 신경 쓸 부분은 정보 검색
- 정보 검색에는 일반적인 데이터베이스가 아닌 벡터 데이터베이스를 사용하기 때문에 임베딩 과정이 필요
- 이후 유사도 검색과 랭킹 처리가 필요
- 즉, 임베딩, 유사도 검색, 랭킹 처리인데, 이 모든 것이 랭체인으로 가능
- 물론 프롬프트 처리와 LLM 연결도 랭체인을 통해 가능

랭체인 개념

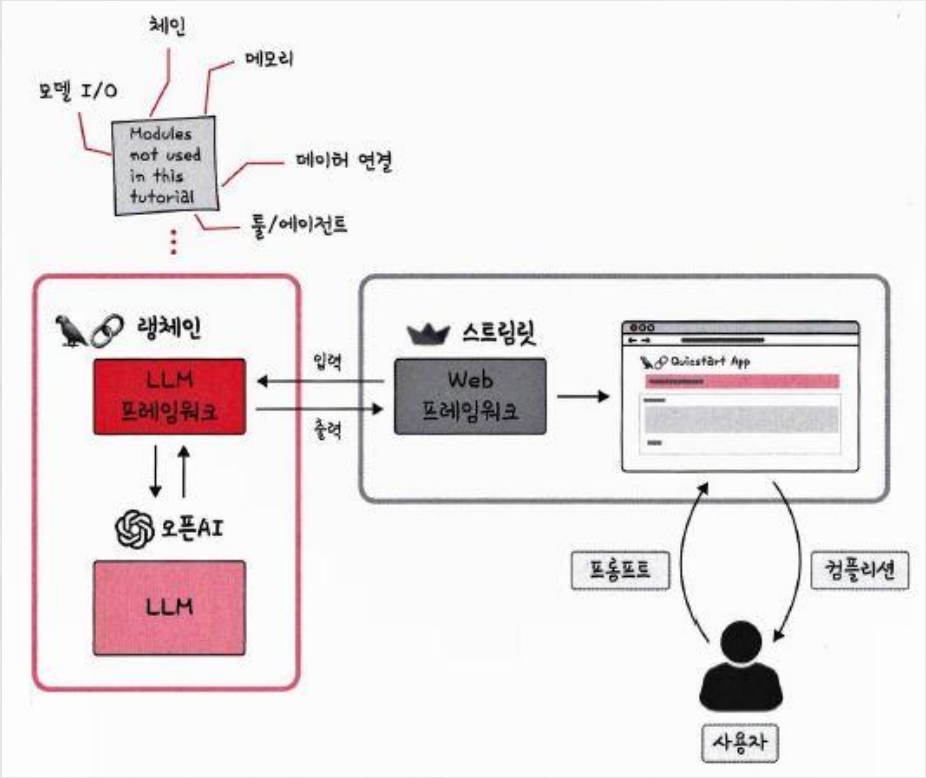
랭체인으로 가능한 것들

- 랭체인의 가장 핵심적인 역할은 이름에서 알 수 있듯이 LLM과 외부 도구를 마치 사슬 처럼 엮어 결합시켜 주는 것

랭체인 개념

랭체인으로 가능한 것들

- 랭체인은 다음 영역에서 편리하게 개발할 수 있도록 도와 줌
- 스트림릿(Streamlit)은 사용자와 상호작용을 할 수 있는 애플리케이션을 쉽게 만들어주는 도구



정리

정리

- LLM 개념
- RAG의 개념
- LangChain 개념