03. RAG 훑어보기

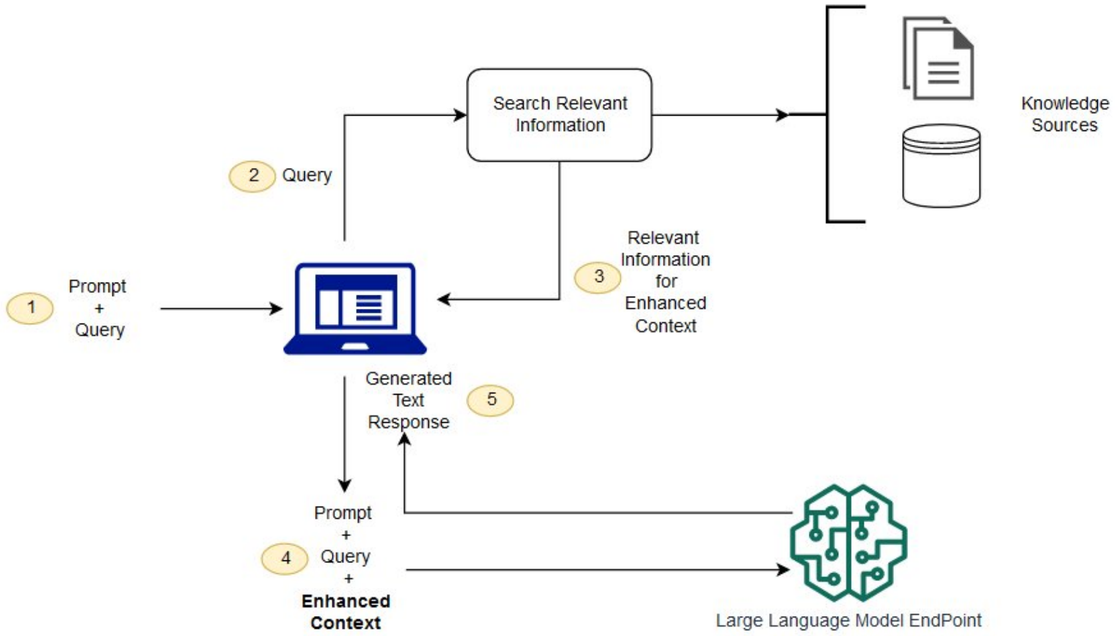
3.1 RAG 개념 ~ 3.3 RAG 구현 시 필요한 것

발표 일시 : 2025.04.08(화)
발표자 : 권선옥

3.1 RAG 개념

RAG

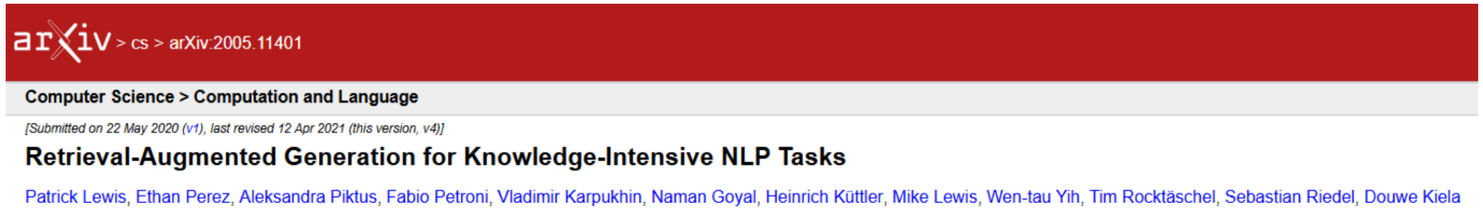
- RAG(Retrieval-Augmented Generation) : 검색 증강 생성(LLM의 정확성, 신뢰성을 높이는데 도움)
  - (Retrieval) LLM이 텍스트를 생성할 때 관련 정보 탐색
  - (Augmented) 외부 정보 검색 기능으로 정보에 대한 맥락을 강화하거나 확장
  - (Generation) 탐색한 정보를 활용하여 새로운 텍스트를 생성



출처 : https://aws.amazon.com/ko/what-is/retrieval-augmented-generation/

3.1 RAG 개념

RAG 논문



☆ 논문의 주요 내용

- RAG 모델이 최신 매개변수 기반 seq2seq 모델보다 더 구체적이고 다양하고 사실적인 언어를 생성하는 것을 확인했다.
- 학습된 검색(retrieval) 구성 요소에 대한 효과성을 입증했으며, 재학습 없이도 검색 인덱스를 핫스왑(실시간 교체)하여 모델을 업데이트할 수 있는 방법을 보였다.

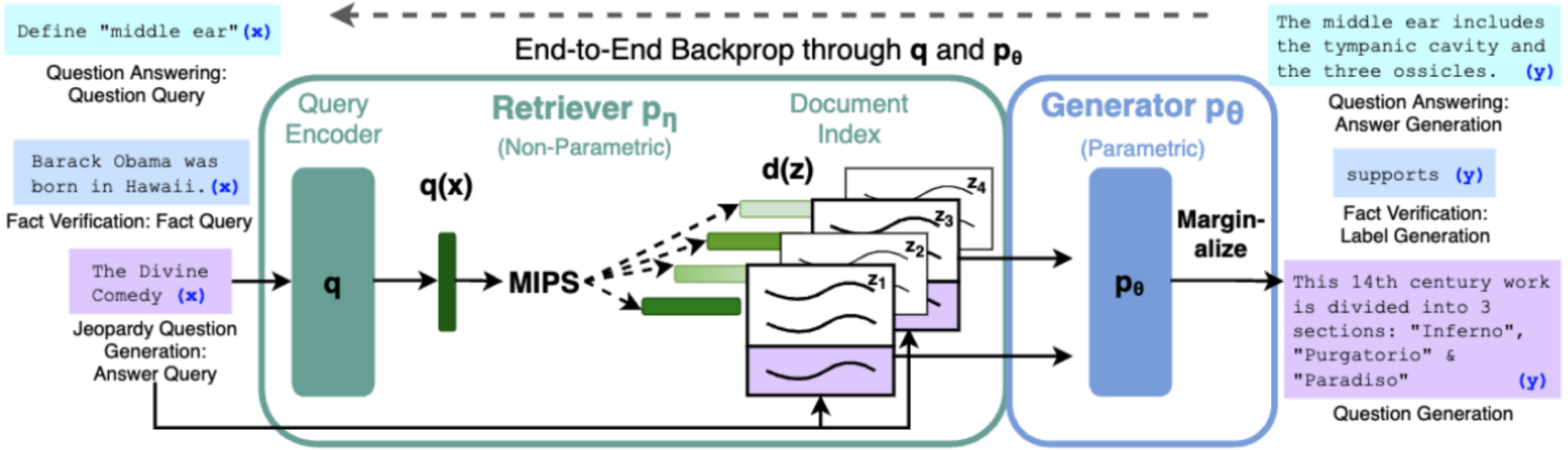
구분	검색기 (Retriever)	생성기 (Generator)
모델 유형	비매개변수적 (Non-Parametric)	매개변수적 (Parametric)
학습 방식	문서 인덱스를 활용하여 검색	훈련된 뉴럴 네트워크를 기반으로 생성
정보 출처	외부 문서 데이터베이스 (검색 기반)	내부 학습된 가중치 (딥러닝 모델)
변경 가능성	문서 인덱스가 변경되면 즉시 업데이트 가능	학습된 파라미터를 변경해야 함

출처 : https://arxiv.org/abs/2005.11401

3.1 RAG 개념

RAG

- 사전 학습된 검색기(Query Encoder + Document Index)와 사전 학습된 seq2seq2 모델(Generator)를 결합하고 한번에 파인튜닝한다.
- 쿼리 x 에 대해 최대 내적 탐색(MIPS)을 사용하여 상위 K개의 문서 z를 찾는다.
  - 내적 값이 클수록 유사도 높음
- 최종 예측 y를 위해, z를 잠재 변수(latent variable)로 간주하고, 서로 다른 문서가 주어진 상태에서 seq2seq 예측을 주변화(marginalize)한다.
  - 잠재 변수 : 관찰되지 않은 변수(hidden) → 최종 예측에 영향을 미치는 숨겨진 요인으로 작용함
  - 주변화 : 특정 변수 하나에 대한 확률값 추정을 위해 나머지 변수를 모두 적분(확률을 합산 or 평균)하여 제거하는 방법(=주변 변수화 한다)



출처 : https://arxiv.org/abs/2005.11401

3.1 RAG 개념

RAG

- Table 1 - 4가지 오픈 도메인 QA 작업 모두에서 다른 모델보다 우수한 성능을 보임
  - closed-book(파라미터 기반)의 생성 유연성 + open-book(검색 기반)의 성능을 결합
- Table 2 - 생성과 분류 테스트에서 BART를 앞섰고, SOTA 모델과 유사한 성능을 보임
  - 기존 모델이 정답 생성에 필요한 특정 정보를 포함한 'gold passages'에 접근 가능했던 반면, RAG는 접근하지 않고도 (Wikipedia 사용)근접한 성능을 냄

  → 검색된 문서가 부족하거나 정답을 포함하지 않아도 합리적인 답변을 생성

Table 1: Open-Domain QA Test Scores. For TQA, left column uses the standard test set for Open-Domain QA, right column uses the TQA-Wiki test set. See Appendix D for further details.

	Model	NQ	TQA	WQ	CT
Closed Book	T5-11B [52]	34.5	- /50.1	37.4	-
	T5-11B+SSM[52]	36.6	- /60.5	44.7	-
Open Book	REALM [20]	40.4	- / -	40.7	46.8
	DPR [26]	41.5	**57.9**/ -	41.1	50.6
	RAG-Token	44.1	55.2/66.1	**45.5**	50.0
	RAG-Seq.	**44.5**	56.8/**68.0**	45.2	**52.2**

Table 2: Generation and classification Test Scores. MS-MARCO SotA is [4], FEVER-3 is [68] and FEVER-2 is [57] *Uses gold context/evidence. Best model without gold access underlined.

Model	Jeopardy B-1	Jeopardy QB-1	MSMARCO R-L	MSMARCO B-1	FVR3 Label Acc.	FVR2 Label Acc.
SotA	-	-	**49.8***	**49.9***	**76.8**	**92.2***
BART	15.1	19.7	38.2	41.6	64.0	81.1
RAG-Tok.	**17.3**	**22.2**	40.1	41.5	72.5	89.5
RAG-Seq.	14.7	21.4	40.8	44.2		

출처 : https://arxiv.org/abs/2005.11401

3.2 RAG 구현 과정

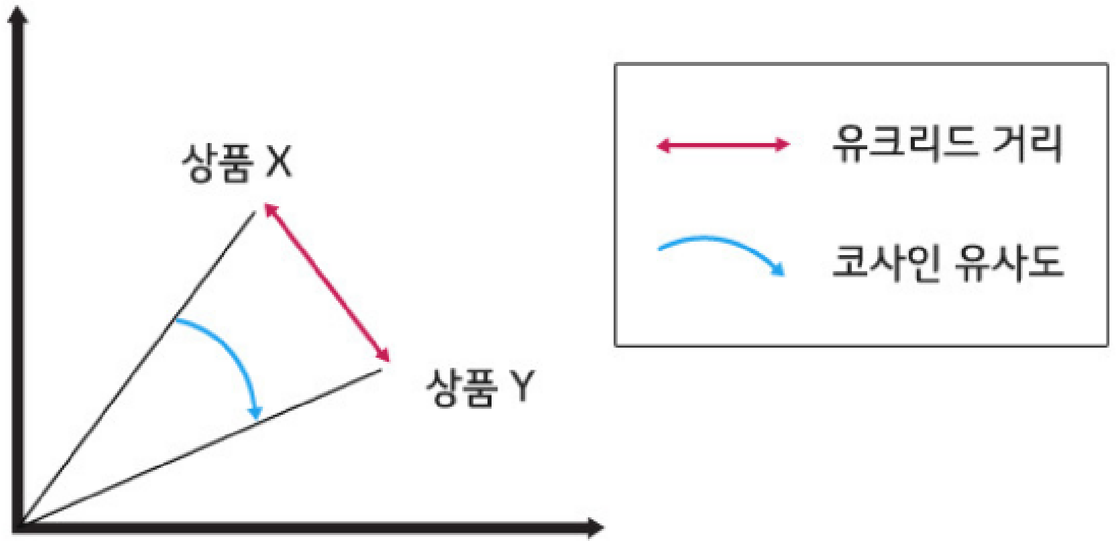
3.2.1 정보 검색

- 단계
  - 1. 질문 입력 : 쿼리(query)를 입력한다.
  - 2. 검색 : 해당 쿼리와 관련된 정보를 DB 에서 탐색한다.
  - 3. 유사도 계산 : 쿼리와 DB의 문서 간의 유사도를 계산한다.
    - ■keyword search : 사용자가 입력한 쿼리가 명시적으로 나타난 경우만 탐색(의도, 문맥과 상관 없는 결과 반환되기도 함)
    - ■semantic search : 단어의 의미, 문맥을 이해하여 관련성 높은 결과 반환
  - 4. 랭킹 처리 : 질문과 가장 관련이 높은 검색 결과 순으로 나열
    - ■유사도 계산을 통해 사용자의 질문이나 요구에 가장 적합한 정보 결정

3.2 RAG 구현 과정

3.2.2 (심화)정보 검색

- 벡터와 유사도
  - 벡터(vector) : 방향과 크기를 나타내는 값
  - 유사도
    - ■코사인 유사도 : 두 벡터 간의 각도를 계산하여 유사도 측정
    - ■유클리드 유사도(유클리드 거리) : 두 점 사이의 '직선 거리'를 계산하여 유사도 측정



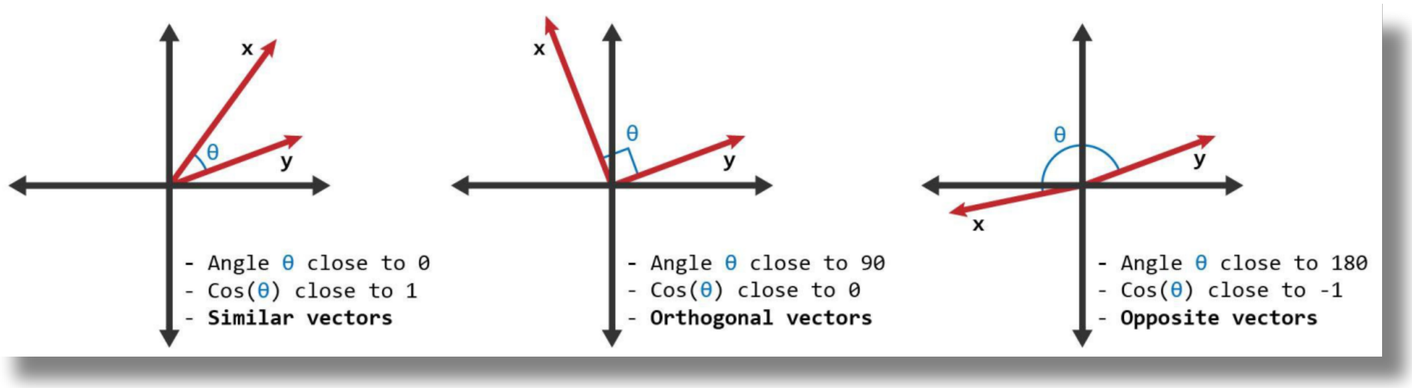
출처 : https://www.lgcns.com/blog/cns-tech/ai-data/15526/

3.2 RAG 구현 과정

3.2.1 정보 검색

- 유사도 계산 : 문서 사이의 관련성이나 유사성을 수치로 표현하는 방법
  - 대표적으로 '코사인 유사도'를 사용함
    - ■유사도는 -1과 +1 사이의 값.
    - ■벡터 사이의 각도가 작을수록 코사인 값이 더 커져 코사인 유사도가 더 크다는 것을 나타냄

$$similarity(A, B) = cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|}$$



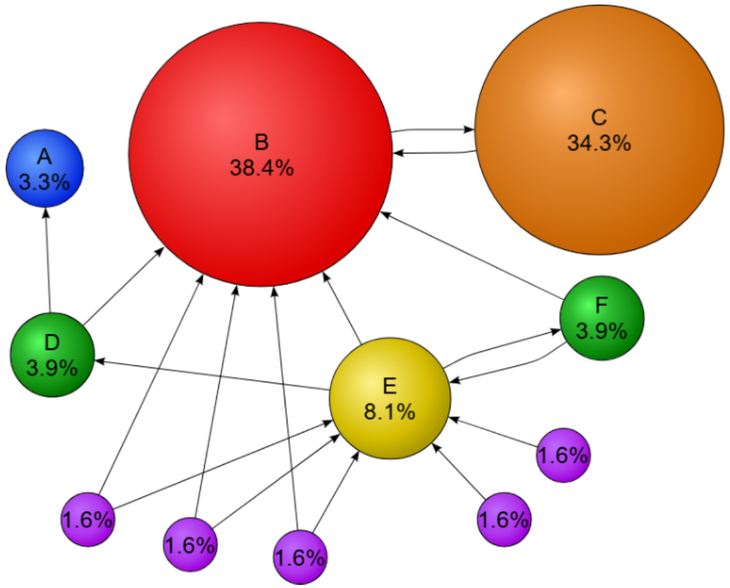
출처 :
https://www.learndatasci.com/glossary/cosine-similarity/

3.2 RAG 구현 과정

3.2.2 (심화)정보 검색

- 검색 결과 랭킹 처리
  - 검색 엔진이 사용자의 쿼리에 가장 관련성이 높은 문서를 결정하고 이를 순서대로 나열하는 방법(랭킹)
- 랭킹 방법
  - 페이지랭크(PageRank) : Google의 검색 결과에 랭킹을 매기는 알고리즘(가장 일반적)
    - ■영향력 있는 페이지가 인용할수록 페이지랭크가 올라감

      (ex. 아래 이미지의 'C'는 영향력 있는 'B'가 인용하여 랭크가 높아짐)



출처: https://en.wikipedia.org/wiki/PageRank

3.2 RAG 구현 과정

3.2.2 (심화)정보 검색

- **TF-IDF(Term Frequency - Inverse Document Frequency)**
  - 특정 단어가 나타나는 단어 빈도(TF)와
  - 그 단어가 전체 문서에서 얼마나 드물게 나타나는지에 대한 문서 빈도(IDF)를 계산하여 랭크를 매기는 방법
  - TF-IDF 값이 높은 문서는 쿼리와 더 관련이 높음(해당 단어가 특정 문서에서는 많이 등장하지만, 다른 문서에서는 거의 등장하지 않는다는 뜻)
  - ex.
    - 'RAG'라는 단어가 한 권의 책에 10회 출현 → TF = 10
    - 모든 책(30권)에서 'RAG'가 2개의 책에서만 출현 → IDF = log(30/2) = log(15)
    - TF-IDF = 10 * log(15) = 10×1.1761=11.761

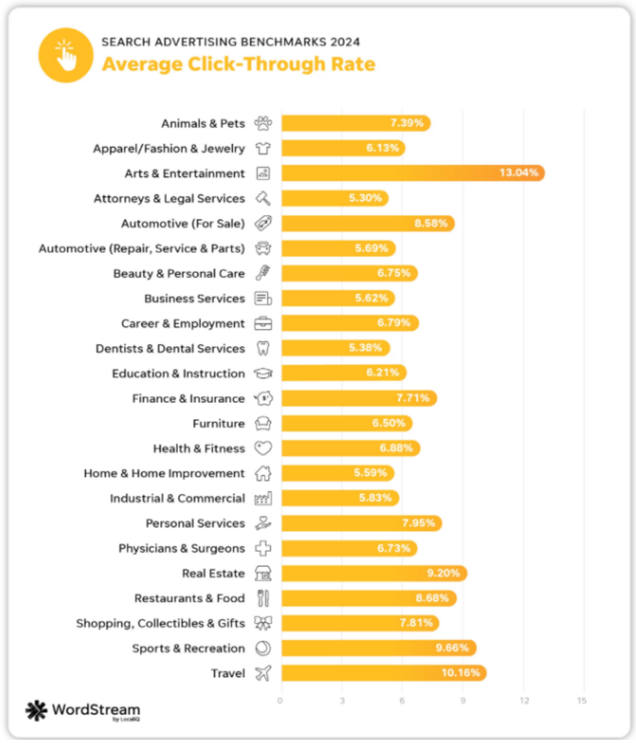
출처: https://en.wikipedia.org/wiki/PageRank

3.2 RAG 구현 과정

3.2.2 (심화)정보 검색

- **클릭률(Click-Through Rate, CTR)**
  - 링크가 얼마나 자주 클릭되는지를 측정하는 지표(높을수록 해당 링크에 대한 관심이 높다는 의미)
  - (클릭수 / 노출수) * 100
  - ex. 100명이 사이트를 조회하고, 5명만 상품을 구매함 ⇒ (5/100) * 100 = 5%

The average click-through rate in Google Ads in 2024 is 6.42%.

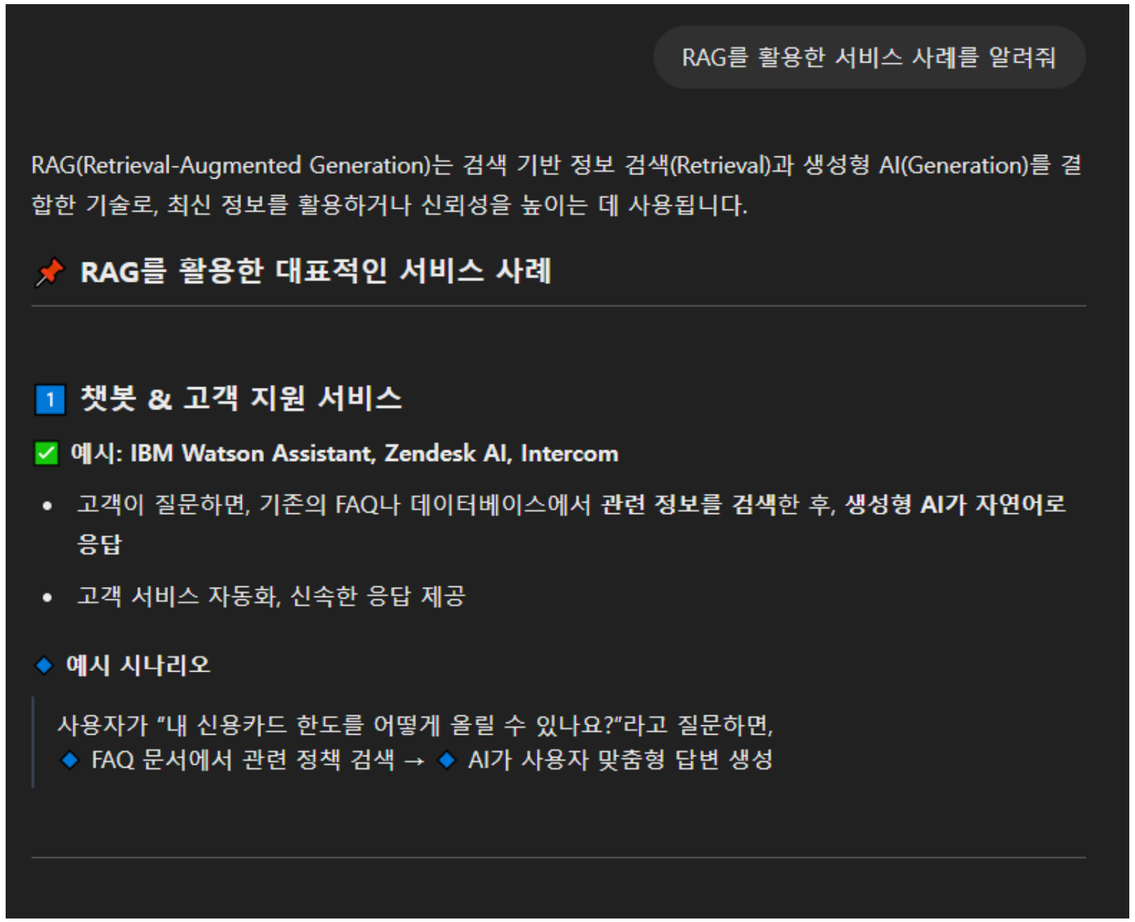


출처: https://www.wordstream.com/blog/2024-google-ads-benchmarks

3.2 RAG 구현 과정

3.2.3 텍스트 생성

- 사용자의 질문과 검색 결과로 텍스트를 생성
  - 단순히 정답만 제시하는 것이 아닌 검색 결과를 함께 제시함으로써 구체적인 결과 생성



3.3 RAG 구현 시 필요한 것

3.3.1 데이터

- RAG에서 사용 가능한 데이터는 특별히 정해진 형식이 없음
- 중요한 것은 데이터의 규범, 규제를 고려하는 것(개인정보 포함 여부, 저작권법 침해 여부 등)
- 획득한 데이터는 두 가지 방법으로 사용됨
  - semantic search : 단어의 의미, 문맥을 이해하여 관련성 높은 결과 반환
  - vector search : 쿼리 벡터와 가장 유사한 벡터를 기반 → embedding 필요!

3.3 RAG 구현 시 필요한 것

3.3.1 데이터

- 임베딩(embedding) : 복잡한 데이터를 간단한 형태(숫자)로 바꾸는 것

1. Word2Vec : 단어 → 벡터로 변환

```python
from gensim.models import Word2Vec

training_data = [
    ['어제', '아침', '비', '많이', '내렸다']
]

word2vec_model = Word2Vec(sentences=training_data, min_count=1)

word_vector = word2vec_model.wv['어제']
word_vector
```

[2] ✓ 1.2s

```
array([-0.00713902,  0.00124103, -0.00717672, -0.00224462,  0.0037193 ,
        0.00583312,  0.00119818,  0.00210273, -0.00411039,  0.00722533,
       -0.00630704,  0.00464722, -0.00821997,  0.00203647, -0.00497705,
       -0.00424769, -0.00310898,  0.00565521,  0.0057984 , -0.00497465,
```

2. GloVe(Global Vectors for Word Representation)
   - 단어의 의미 → 벡터로 변환

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

glove_path = 'glove.6B/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")

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_vector
```

[4] ✓ 0.0s

```
C:\Users\tjsdh\AppData\Local\Temp\ipykernel_20564\624368931.py:11: DeprecationWarning: Ca
  glove2word2vec(glove_path, word2vec_output_file)

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

3. OpenAI 모델

```python
from openai import OpenAI

client = OpenAI(
    api_key = '
)

document = ['제프리 힌튼', '교수', '토론토 대학', '사임']

response = client.embeddings.create(
    input = document,
    model = "text-embedding-ada-002"
)

response
```

[5] ✓ 2.2s Python

```
CreateEmbeddingResponse(data=[Embedding(embedding=[-0.025345299392938614, -0.019261321052908897, -0.007729417644441128, -0.016329949721693993, -0.00801287591457367, 0.028065
```

3.3 RAG 구현 시 필요한 것

3.3.2 벡터 DB

- 벡터 DB : 벡터 저장소(데이터 간의 '유사성'을 바탕으로 검색하는 데 사용됨)
  - ex. 사용자 질문 : "배송 상태를 어떻게 확인할 수 있나요?"
    - ■query_vector : 질문의 벡터 표현
    - ■top_k : 가장 유사한 문서의 수

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

3.3 RAG 구현 시 필요한 것

3.3.2 벡터 DB

- 벡터 DB 종류

벡터 DB	유료/무료	특징
Pinecone (파인콘)	유료 (무료 티어 제공)	클라우드 기반 벡터 검색, 간단한 API
Milvus (밀버스)	오픈소스 (무료) + 유료 옵션	높은 수준의 제어 가능, 대규모 데이터 지원
Qdrant (쿼드런트)	오픈소스 (무료) + 유료 옵션	Rust 기반, 고성능 검색 엔진, 클라우드 & 온프레미스 지원
Chroma (크로마)	오픈소스 (무료)	간단한 사용법, AI 및 RAG 최적화
Elasticsearch (엘라스틱서치)	오픈소스 (무료) + 유료 옵션	기존 검색 엔진 + 벡터 검색 기능 추가
Faiss (파이스)	오픈소스 (무료)	Facebook AI 연구팀 개발, GPU 가속 지원

3.3 RAG 구현 시 필요한 것

3.3.3 framework

LangChain(LLM 밀키트)

- 개념 : LLM을 활용한 애플리케이션 개발에 특화된 오픈소스 프레임워크*
  - *앱 개발 시 기본적인 것을 제공하는 소프트웨어 도구 모음
- 출시 : 2022년 10월 말, 오픈 소스로 출시됨
- 상징 : 앵무새(인간의 언어를 따라서 말할 수 있다는 점) + 사슬⛓ (언어 모델과 언어 모델을 활용할 수 있는 다양한 도구를 결합)
- RAG 구현을 위해 '정보 검색' & '텍스트 생성'(LLM) → '정보 검색'을 위한 준비 필요
  - 임베딩, 유사도 검색, 랭킹 처리 → 랭체인으로 모두 가능

핵심 요약

- RAG는 정보를 찾고(Retrieval), 외부 정보 검색으로 맥락을 확장(Augmented)하여, 새로운 텍스트를 생성(Generation)하는 기술이다.
- RAG는 정보 탐색 + 텍스트 생성 단계로 구성되며, 정보 탐색에는 쿼리 입력 -> DB 검색 -> 유사도 계산 -> 랭킹 처리 단계를 거친다.
- 유사도 계산은 문서 사이의 관련성이나 유사성을 수치로 표현하는 방법으로 코사인 유사도가 대표적이다.
- 랭킹 방법에는 페이지랭크, TF-IDF, 클릭률 등이 있다.
- RAG 활용 서비스 개발을 위해서는 데이터, 벡터 DB, framework가 필요하며 데이터는 임베딩을 통해 벡터로 변환해야한다.