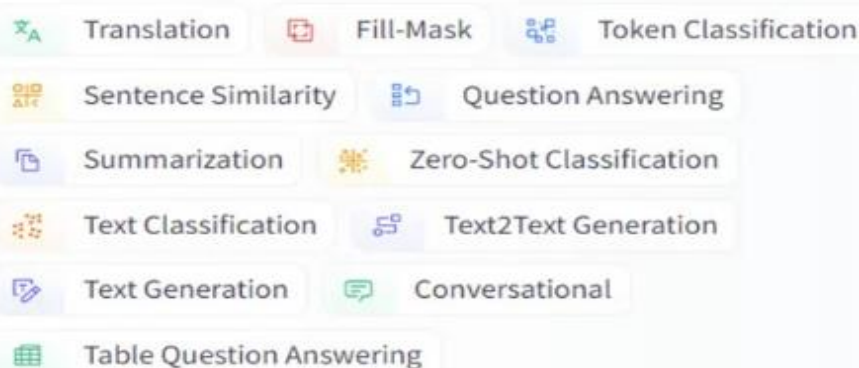


현대의 텍스트 데이터 처리에서 Embedding 기술은 매우 중요한 역할을 하고 있다. 임베딩 기술은 의미 기반 검색(Semantic Search), 추천(Recommendation), 군집화(Clustering) 등 다양한 애플리케이션에서 필수적으로 활용되며, 특히 RAG 프로젝트의 성공을 위해서는 임베딩에 대한 이해가 필수적이다. 임베딩은 문서 전처리와 RAG 파이프라인의 핵심 요소로 자리 잡고 있으며, 데이터의 품질과 검색 효율성을 크게 좌우한다.

1. Embedding의 필요성

자연어 처리의 세부 분야

Natural Language Processing



분야는 이렇게 많지만, 자연어 처리의 과정은 크게 두가지로 나누어 볼 수 있다.

AI 모델은 기본적으로 하나의 “함수”로 작동하며, 숫자 형태의 입력과 출력만을 처리할 수 있다. 그러나 사람이 입력한 텍스트는 본질적으로 숫자가 아니기 때문에, 이를 AI 모델이 이해할 수 있는 숫자 형태로 변환해야 한다. 이를 위해 자연어 처리(NLP)과정에서 AI가 자연어를 이해할 수 있도록 해야 한다.

자연어 처리의 과정



Tokenize

분석하고자 하는 문서를 '단위'로 나누기

	long hair	black	human	gender	cat	dog	house
cat	0.6	0.9	0.1	0.4	-0.7	-0.3	-0.2
kitten	0.5	0.8	-0.1	0.2	-0.6	-0.5	-0.1
dog	0.7	-0.1	0.4	0.3	-0.4	-0.1	-0.3
houses	-0.0	-0.4	-0.5	0.1	-0.9	0.3	0.0

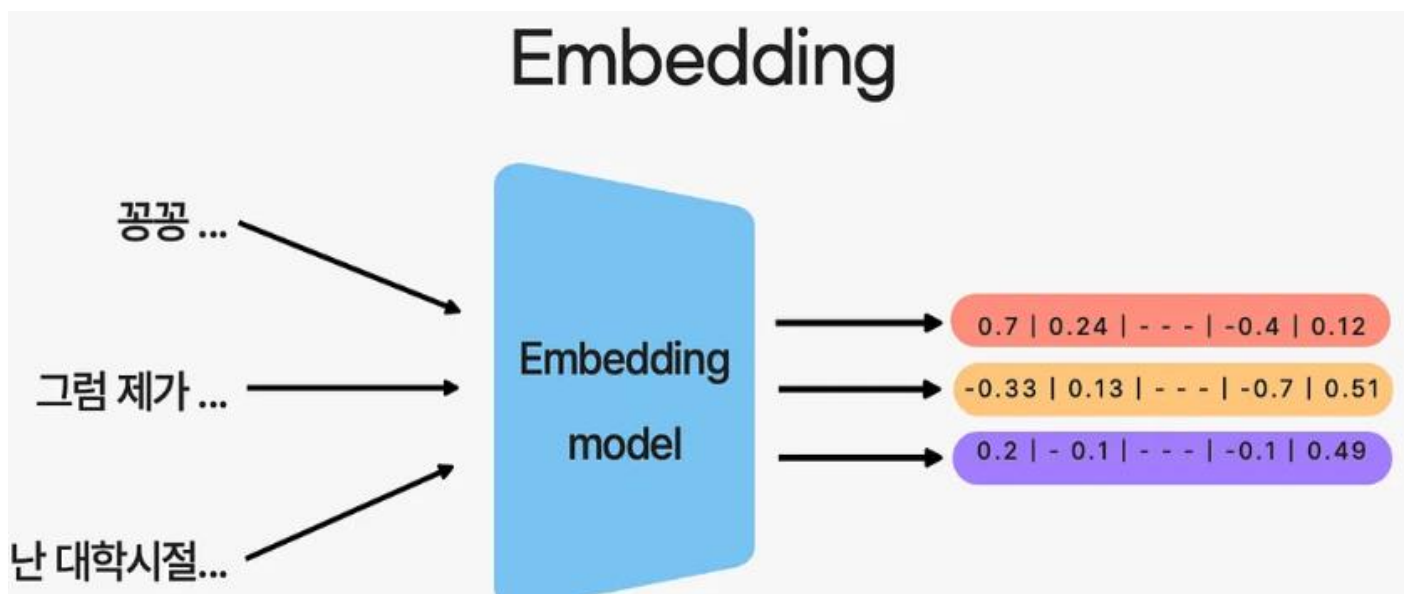
Embedding

분석의 단위를, 컴퓨터가 이해할 수 있는 '숫자'로 만들기

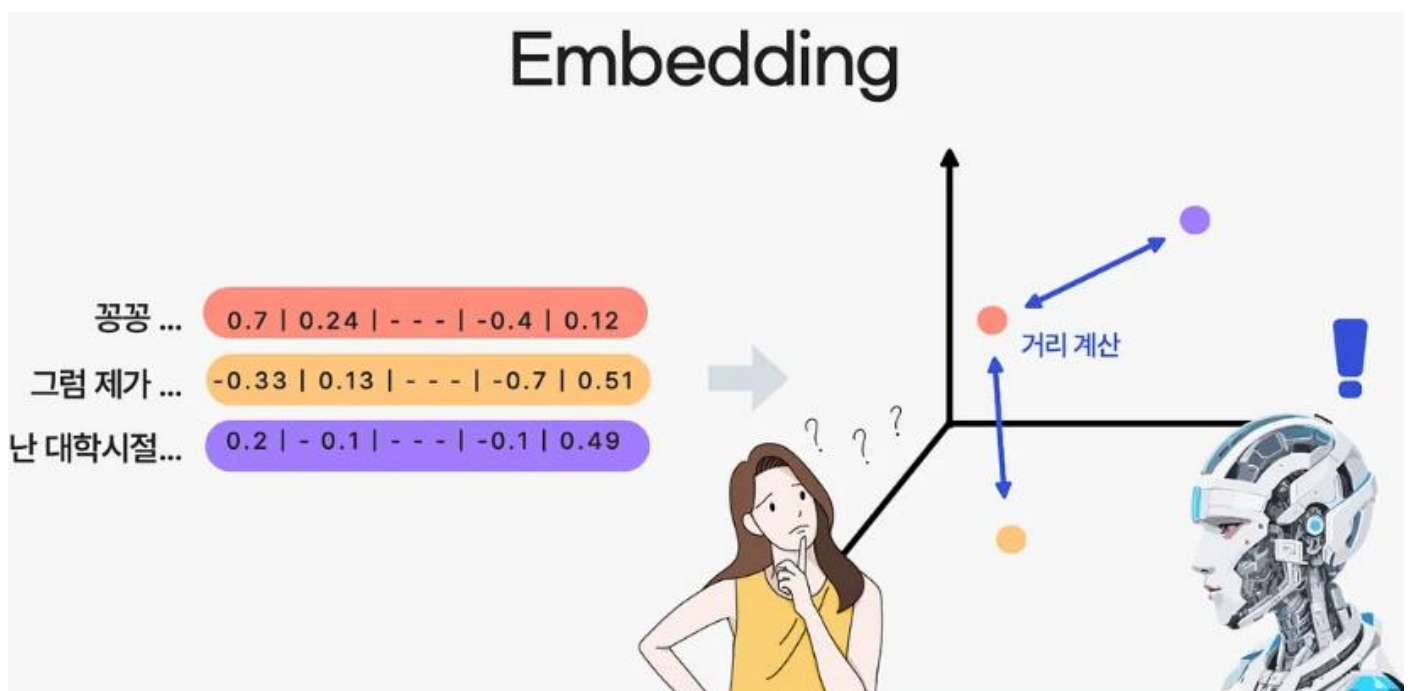
분석하고자 하는 문장이나 문서를 정해진 단위로 나누는 **토큰나이징(Tokenize)** 과정과 이렇게 나눈 단위를 **컴퓨터가 이해할 수 있는 숫자 형태로 변환하는 임베딩(Embedding)** 과정으로 나누어진다.

2. Embedding 이란?

데이터를 벡터 형태로 표현하는 것을 의미한다. 특히 텍스트나 이미지를 컴퓨터가 이해할 수 있는 형태로 변환하여, **의미와 정보를 벡터 공간에 담아내는 과정**이다. 임베딩은 단어의 의미, 문맥, 이미지의 특징 등 다양한 정보를 표현하며, 이를 통해 유사도 계산, 분류, 검색 등 다양한 작업에 활용한다.



임베딩(Embedding)은 텍스트를 “**실수 벡터 형태**”로 표현한 결과를 의미한다. 특정한 단어, 문장 또는 문서를 임베딩 생성 모델에 입력하면, 일정 수의 실수로 구성된 벡터가 출력된다.



임베딩의 결과는 사람이 직접 관찰하고 그 의미를 파악하기는 어렵지만, 컴퓨터는 서로 다른 단어 또는 문서에서 추출된 임베딩 간의 거리를 계산하여 이들 간의 의미적 관계를 이해할 수 있다.

특징	Word (Token) Embedding	Sentence/Document Embedding
단위	단어	문장/문서
대표 기법	Word2Vec, GloVe, FastText	Doc2Vec, Universal Sentence Encoder, BERT
문맥 정보 반영	제한적	우수
의미 표현	단어의 미세한 의미 차이 잘 반영	문장/문서 전체의 의미 잘 반영
계산 비용	낮음	높음
어휘 확장성	우수	제한적

3. 과거와 현재의 임베딩 기법: One-hot Encoding vs. Learned Embedding

One-hot Encoding					
	request	feature	how	issue	credit
request →	1	0	0	0	0
feature →	0	1	0	0	0
how →	0	0	1	0	0
issue →	0	0	0	1	0
credit →	0	0	0	0	1

과거에는 텍스트 데이터를 벡터로 표현하기 위해 One-hot Encoding을 사용했다. One-hot Encoding은 단어를 고유한 벡터로 변환하는 간단한 방법이다.

One-hot Encoding의 과정

1. **단어 카운팅**: 텍스트 데이터에 등장하는 모든 단어들을 카운팅한다.
2. **단어집 생성**: 모든 단어들에 고유한 숫자 인덱스를 부여하여 단어집을 만듭니다.
3. **인코딩**: 각 단어를 해당 인덱스 위치에 1을 배치하고 나머지 위치에 0을 배치하는 벡터로 변환한다.

장점

- **직관적 이해**: 각 단어가 벡터의 특정 위치에 대응되므로, 사람이 직접 보고 이해하기 쉽다.

단점

- **고차원 벡터**: 단어 수가 많아질수록 벡터의 차원이 매우 커진다.
- **희소 벡터**: 벡터 내 대부분의 값이 0으로 채워지므로 매우 희소하다.
- **비효율성**: AI 모델이 이러한 고차원 희소 벡터를 효과적으로 처리하기 어려워, 계산 자원과 시간이 많이 소요된다.

One-hot Encoding의 단점은 단어의 수가 많아질수록 문제가 된다. 예를 들어, 단어가 1만 개라면 벡터의 차원도 1만이 되는데, 이 중 대부분의 값이 0이 된다. 이렇게 희소한 벡터는 효율적이지 않으며, AI 모델이 효과적으로 다루기 어렵다.

Learned Embedding

embedding pairs

request	→	0.007	-0.039	-0.002	-0.024
feature	→	-0.014	-0.011	0.014	-0.017
how	→	0.01	0.004	0.006	-0.037
issue	→	-0.015	-0.035	-0.009	-0.01
credit	→	-0.012	-0.027	-0.001	-0.05

현재는 **Learned Embedding**을 사용하여 텍스트 데이터를 벡터로 변환한다. 이는 대규모 문서 모음집을 사용해 **신경망 구조를 지닌 AI 모델(또는 LLM)**을 학습시켜 얻는다.

Learned Embedding의 과정

1. **대규모 데이터셋 수집**: 다양한 문서와 텍스트 데이터를 수집하여 대규모 데이터셋을 만든다.
2. **모델 학습**: 신경망 구조를 지닌 AI 모델을 사용하여 데이터를 학습한다. 대표적인 모델로는 Word2Vec, GloVe, BERT 등이 있다.
3. **임베딩 생성**: 학습된 모델은 단어들을 벡터로 변환한다. 이 벡터들은 **문맥과 의미를 반영**하여 생성된다.

장점

- **문맥 반영**: 단어의 문맥과 의미적 관계를 반영하므로, 유사한 의미를 가진 단어들은 벡터 공간에서 가깝게 위치한다.
- **효율성**: 벡터의 차원이 상대적으로 낮고, 밀집된 형태로 표현되므로 AI 모델이 더 효과적으로 처리할 수 있다.
- **성능 향상**: 의미적 관계를 반영하여, 검색, 추천, 군집화 등 다양한 애플리케이션에서 높은 성능을 보인다.

Learned Embedding의 학습 방식은 다음과 같다:

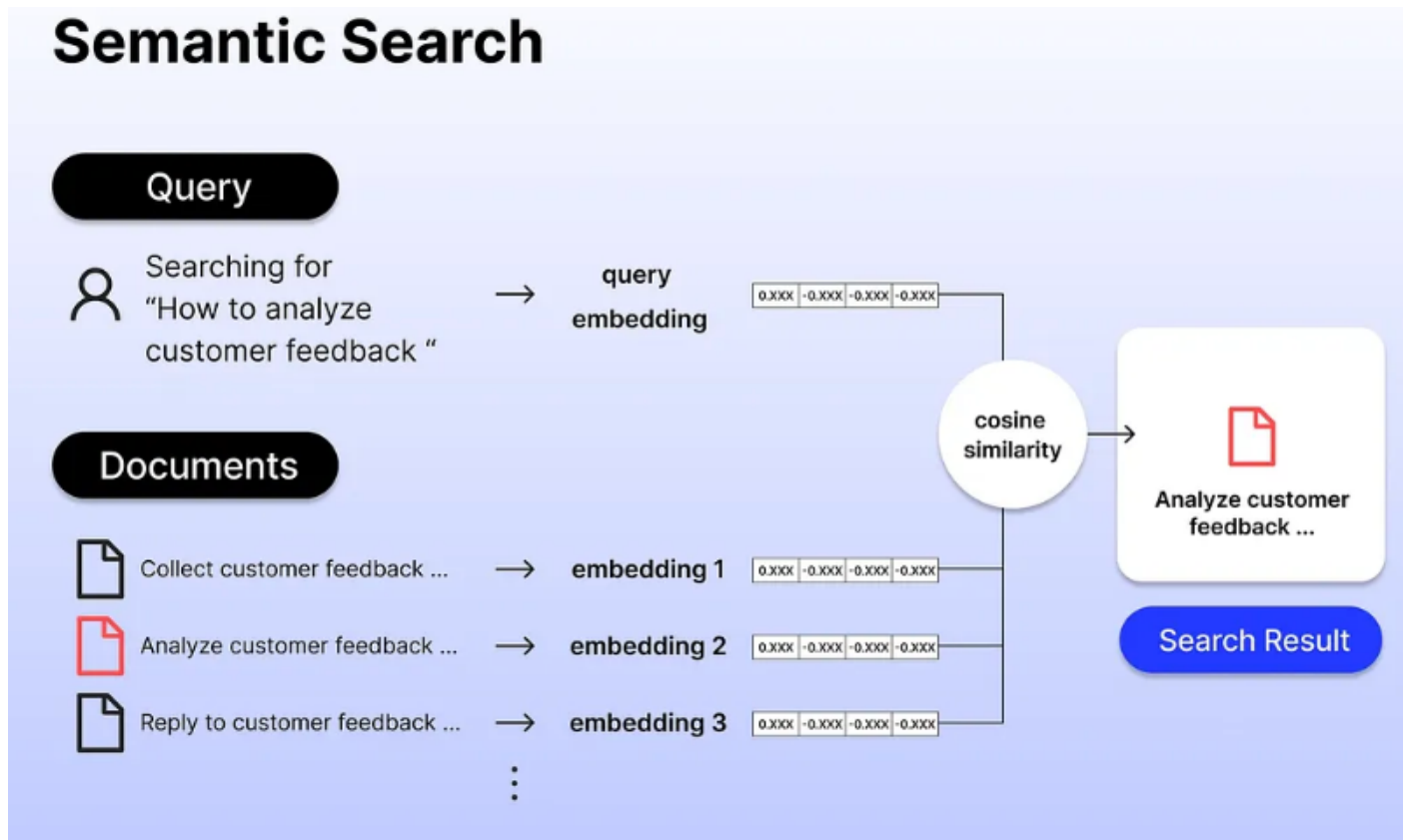
1. **단어 입력**: 모델은 다양한 문맥에서 단어들을 입력으로 받는다.
2. **의미적 관계 학습**: 모델은 문맥 상 유사한 단어들을 가까운 벡터로, 그렇지 않은 단어들을 먼 벡터로 배치하며 의미적 관계를 학습한다.
3. **벡터 출력**: 학습이 완료된 후, 각 단어는 의미적 관계를 반영한 고유한 벡터로 변환된다.

Learned Embedding은 과거의 One-hot Encoding 방식과 비교할 때 많은 장점을 가지고 있다. 단어의 문맥을 반영하여 더 정확한 의미를 전달할 수 있으며, 벡터의 차원이 낮아져 모델의 효율성이 크게 향상된다.

4. Embedding을 언제 사용해야 하는가?

임베딩을 사용해야 하는 주요 사례는 크게 두 가지로 나눌 수 있다. 여러 문서들을 의미적으로 탐색하거나 비교해야 하는 경우, 그리고 LLM(Large Language Models)이 결과를 생성하기 위해 추가적인 정보를 주입해야 하는 경우이다.

Case 1: 여러 문서들을, 의미를 기반으로 탐색하거나 비교해야 하는 경우



Semantic Search는 사용자가 제시한 텍스트 형태의 쿼리(query)와 의미적으로 연관성이 높은 문서들을 찾아 제시하는 기능이다.

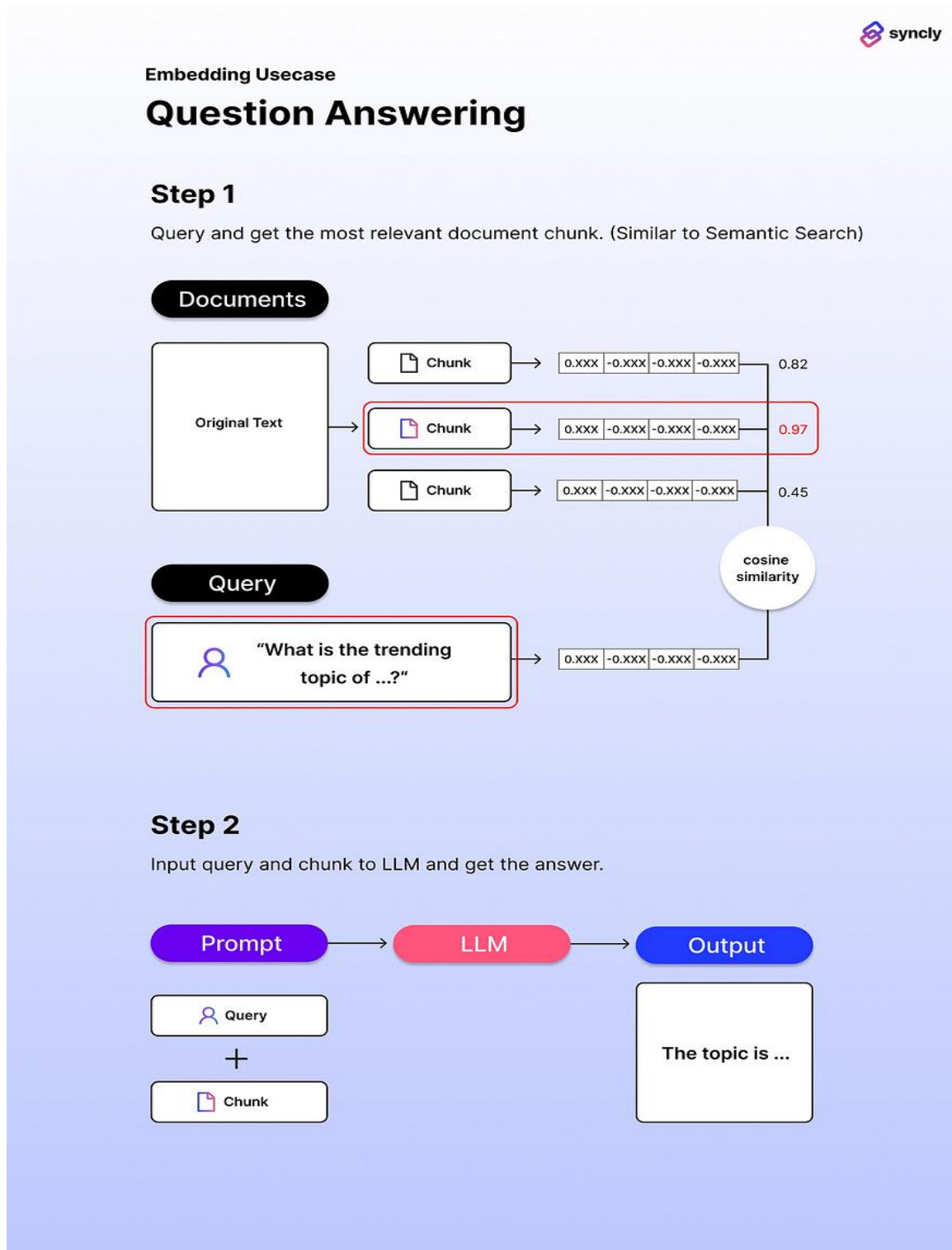
Semantic Search 프로세스

1. **임베딩 계산 및 저장:** 문서 모음집에 포함되어 있는 각각의 문서에 대한 임베딩을 계산하여 별도의 저장소(e.g. 로컬 드라이브, 벡터 데이터베이스 등)에 저장한다.
2. **쿼리 임베딩 계산:** 사용자가 입력한 쿼리 텍스트에 대한 임베딩을 계산한다.
3. **유사도 계산:** 쿼리 임베딩과 각 문서 임베딩 간의 코사인 유사도(cosine similarity)를 계산하고, 그 값을 기준으로 전체 문서들을 내림차순 정렬한다.
4. **결과 반환:** 정렬 결과 중 상위 k 개의 문서 텍스트를 불러와 사용자에게 반환한다.

Case 2: LLM이 결과 생성을 위해 부가적인 정보를 주입해야 하는 경우

LLM은 학습하지 않은(비공개) 정보에 대한 지식을 가지고 있지 않다. 따라서, LLM 으로 하여금 나만이 가지고 있는 정보와 연관된 결과물을 기반으로 어떤 질문에 대한 답을 출력하도록 하고자 한다면, 해당 정보를 담은 텍스트를 LLM의 프롬프트에 함께 포함시켜 요청해야 한다.

이 과정에서 임베딩을 활용하면 더 효과적으로 정보를 처리할 수 있다.



그러나 현재 서비스되고 있는 LLM 서비스들의 경우 prompt에 추가될 수 있는 텍스트의 길이(i.e. token의 총 개수)가 제한되어 있기 때문에, 제시하고자 하는 텍스트의 길이가 매우 길다면, 이를 미리 여러 chunk(덩어리)로 쪼개 놓고 이들 중 주어진 질문과 가장 연관성이 높은 chunk만을 골라서 prompt에 추가해야 한다.

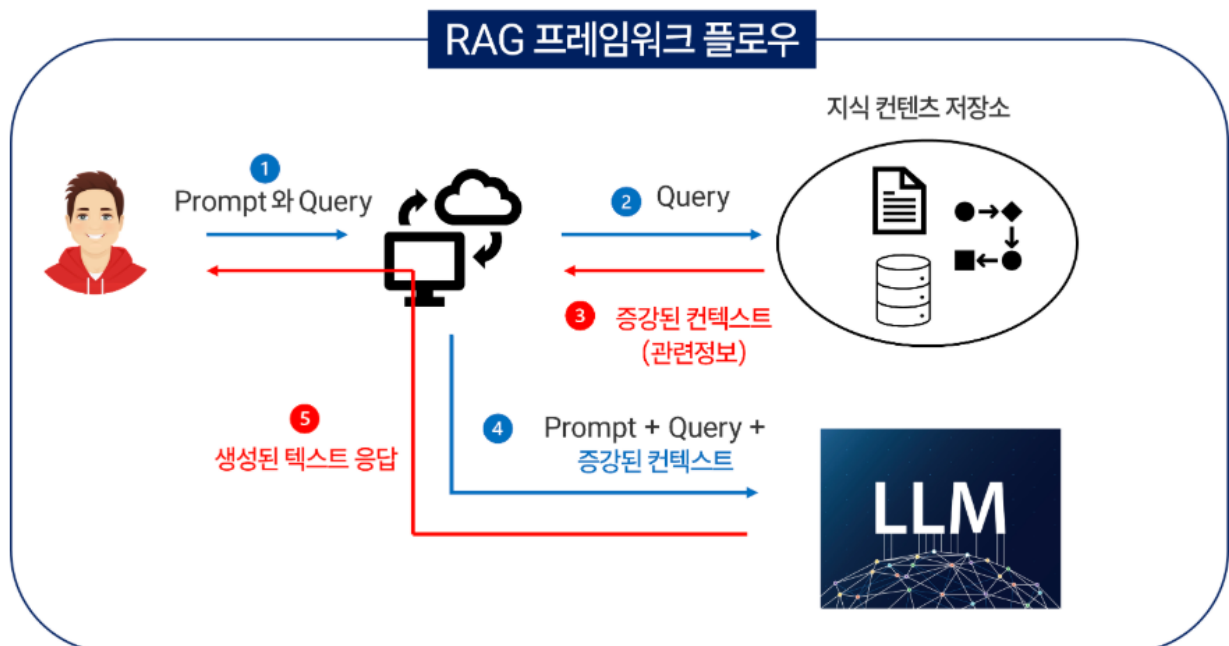
이는 특히 “Question Answering”(질의 응답)과 같은 기능을 구현하고자 할 때 흔히 고려하게 되는 포인트인데, LLM에게 추가로 주입된 정보를 바탕으로 주어진 질문(query)에 대한 답변을 수행하도록 하는 일반적인 Question Answering 프로세스를 간략하게 정리하면 아래와 같다.

이는 **Semantic Search** 프로세스와 어느 정도 유사하다.

Question Answering 프로세스

1. **텍스트 분할 및 임베딩 계산**: 전체 정보를 담은 텍스트를 일정한 길이로 분할하여 여러 개의 텍스트 청크(chunk)를 구성하고, 각각의 청크에 대한 임베딩을 계산하여 별도의 저장소에 저장한다.
2. **쿼리 임베딩 계산**: 질문 내용을 담은 쿼리 텍스트에 대한 임베딩을 계산한다.
3. **유사도 계산**: 쿼리 임베딩과 각 청크 임베딩 간의 코사인 유사도를 계산하고, 그 값을 기준으로 전체 청크들을 내림차순 정렬한다.
4. **프롬프트 구성**: 정렬 결과 중 상위 k 개의 청크를 불러와 프롬프트에 추가한다. k 는 LLM 서비스에서 허용하는 최대 텍스트 길이 내에서 결정된다.
5. **답변 생성**: 완성된 프롬프트를 LLM에 입력하고, 생성된 답변을 반환한다.

이와 같은 방식으로, 임베딩을 활용하여 LLM이 보다 정확하고 관련성 높은 답변을 생성할 수 있도록 도울 수 있다.

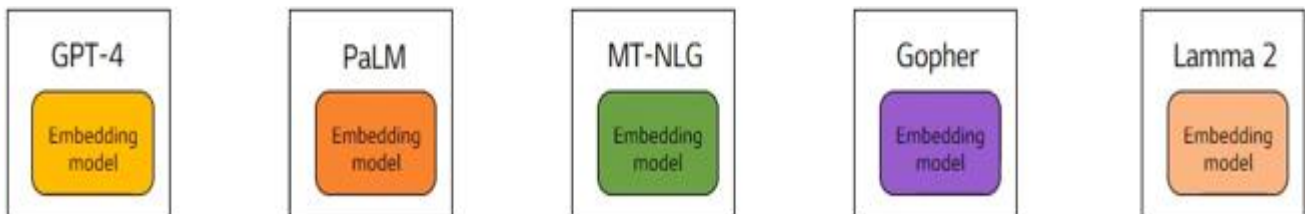


사람의 언어 \neq LLM의 언어

임베딩 (embedding)

사람의 언어

임베딩 (LLM의 언어)



I love you so much.



임베딩 모델 인풋
(인간의 언어)

Embedding
model

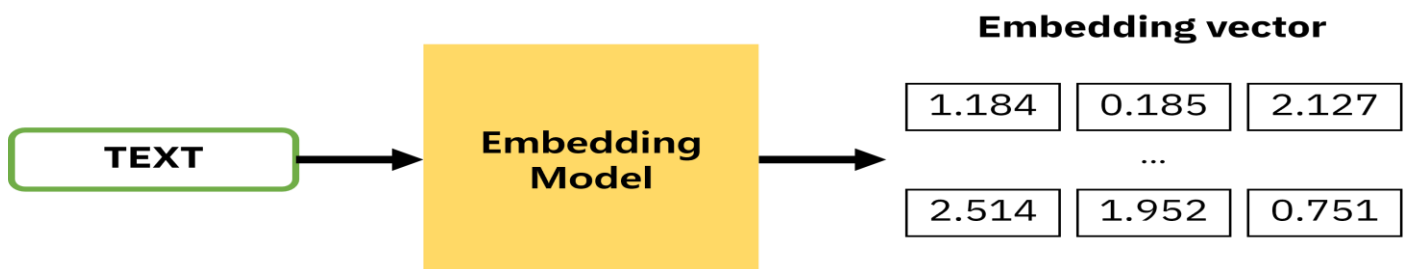
<<시작>> → [0.4810, -0.7042, -0.7409, 0.14624]
 | → [-0.5198, 0.2901, 0.3235, -0.5886]
 love → [1.5240, 2.5387, -1.0701, -0.1190]
 you → [-0.2612, 0.1227, -0.4248, 0.6229]
 so → [-0.7988, 1.6670, 0.0759, -1.2467]
 much → [0.1693, 1.7550, 0.3056, 0.0773]
 . → [-0.0600, 0.9258, -1.2276, 0.4466]
 <<끝>> → [1.3262, -0.8511, 1.4349, -0.6320]

임베딩은 텍스트 데이터를 효율적으로 처리하고 **의미를 수치적으로 표현**하는 강력한 도구이다.

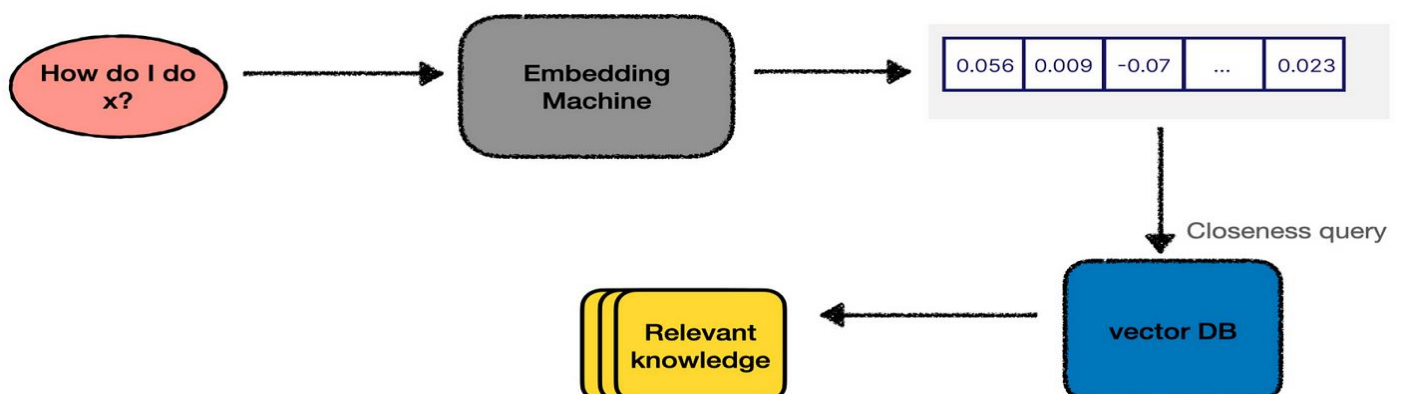
Semantic Search, Recommendation, Clustering과 같은 다양한 애플리케이션에서 임베딩을 활용하여 **텍스트 데이터를 효과적으로 탐색하고 비교**할 수 있다. 또한, 대형 언어 모델(LLM)이 더 나은 결과를 생성할 수 있도록 추가적인 정보를 주입하는 데에도 임베딩은 매우 유용하다. 이러한 임베딩의 활용 방법을 깊이 이해하고 적절하게 적용함으로써, 텍스트 데이터 처리의 효율성과 성능을 크게 향상시킬 수 있다. 따라서, 앞으로의 데이터 분석 및 인공지능 발전에 있어 임베딩 기술의 중요성은 더욱 커질 것이다.



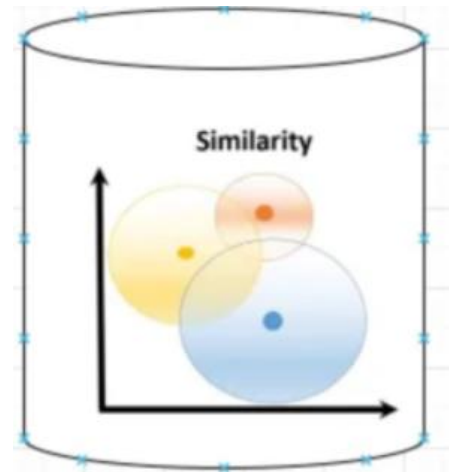
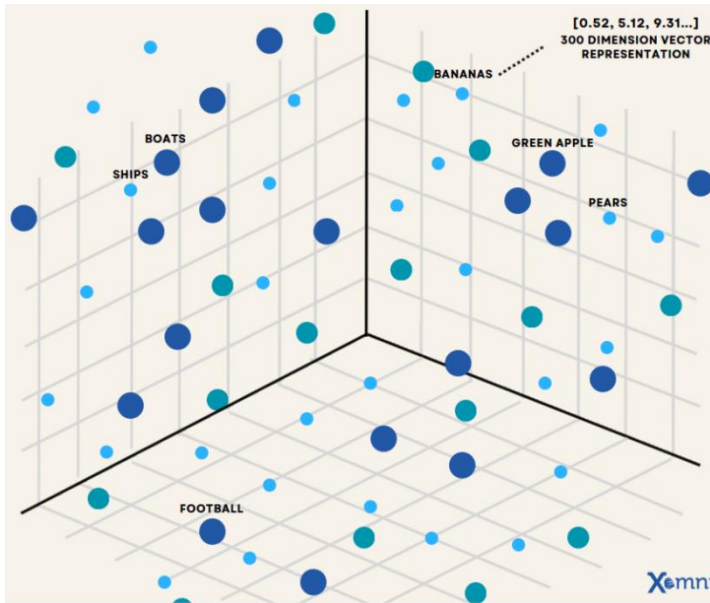
Vector DB는 AI와 머신러닝 모델에서 생성된 벡터를 효과적으로 관리할 수 있도록 설계된 모델이다. 높은 차원의 데이터에서도 빠른 검색 속도, 유사 벡터 검색기능을 지원하며, 대표적으로는 Milvus, Faiss (Facebook AI Similarity Search), Chroma, Qdrant, Pinecone과 같은 vectorDB가 있다. 주로, 이미지 검색, 추천 시스템, 자연어 처리 등 AI 기반의 애플리케이션에 사용되는 DB이다.



Retrieval process

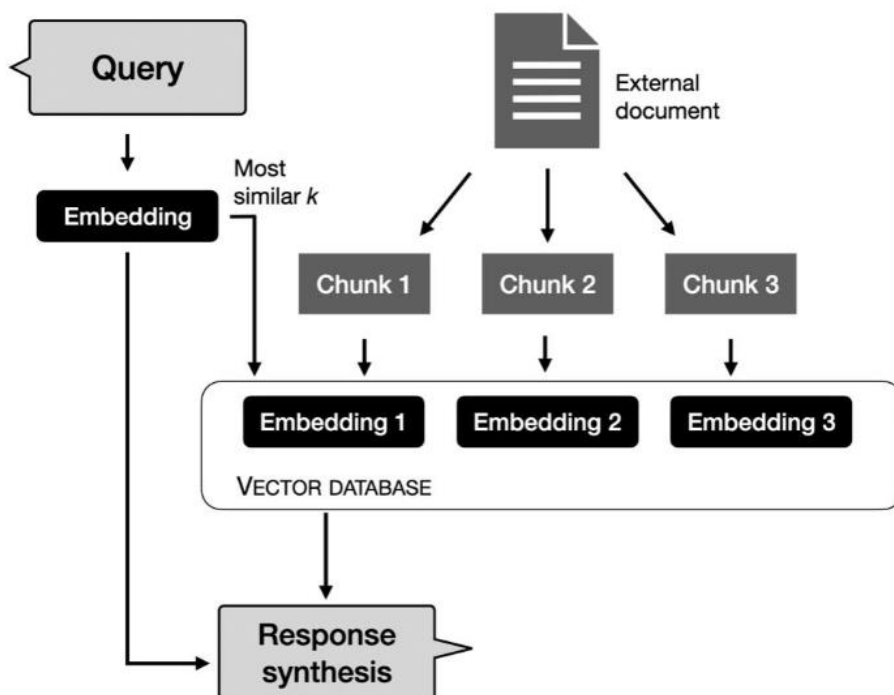


RDB 는 데이터가 테이블(행과 열)으로 구조화되는 반면, 벡터 DB는 벡터 공간에 데이터를 다르며 각 데이터는 고차원 벡터로 표현된다. 즉 벡터DB 는 수학적 특성을 이용해 데이터를 3차원 이상의 다차원 공간에 모델링한다.



청킹(Chunking) : 큰 조각을 작은 조각으로 나누기

청킹은 임베딩을 진행하기 전에 큰 텍스트를 더 작은 단위로 나누는 과정이다. 의미를 보존하면서 작은 단위로 나눠서 검색과 생성 과정에서 더욱 효율적인 이고 정확한 결과를 얻을 수 있다.



[벡터 DB 설치]

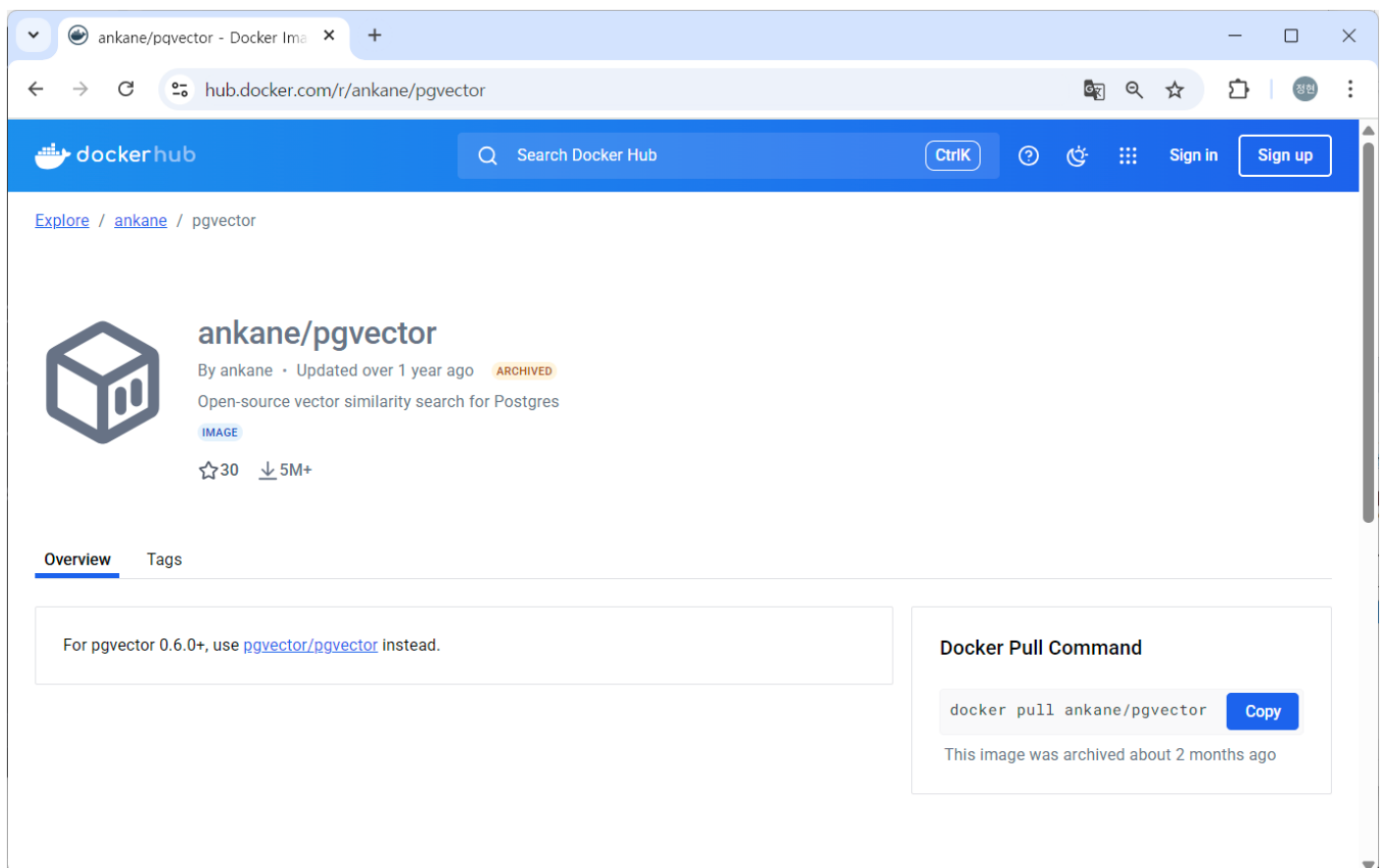
(1) 도커데스크탑을 기동한다.

(2) 볼륨을 생성한다.

```
docker volume create vpg
```

(3) pgvector 라는 벡터DB 를 도커 컨테이너로 기동한다.

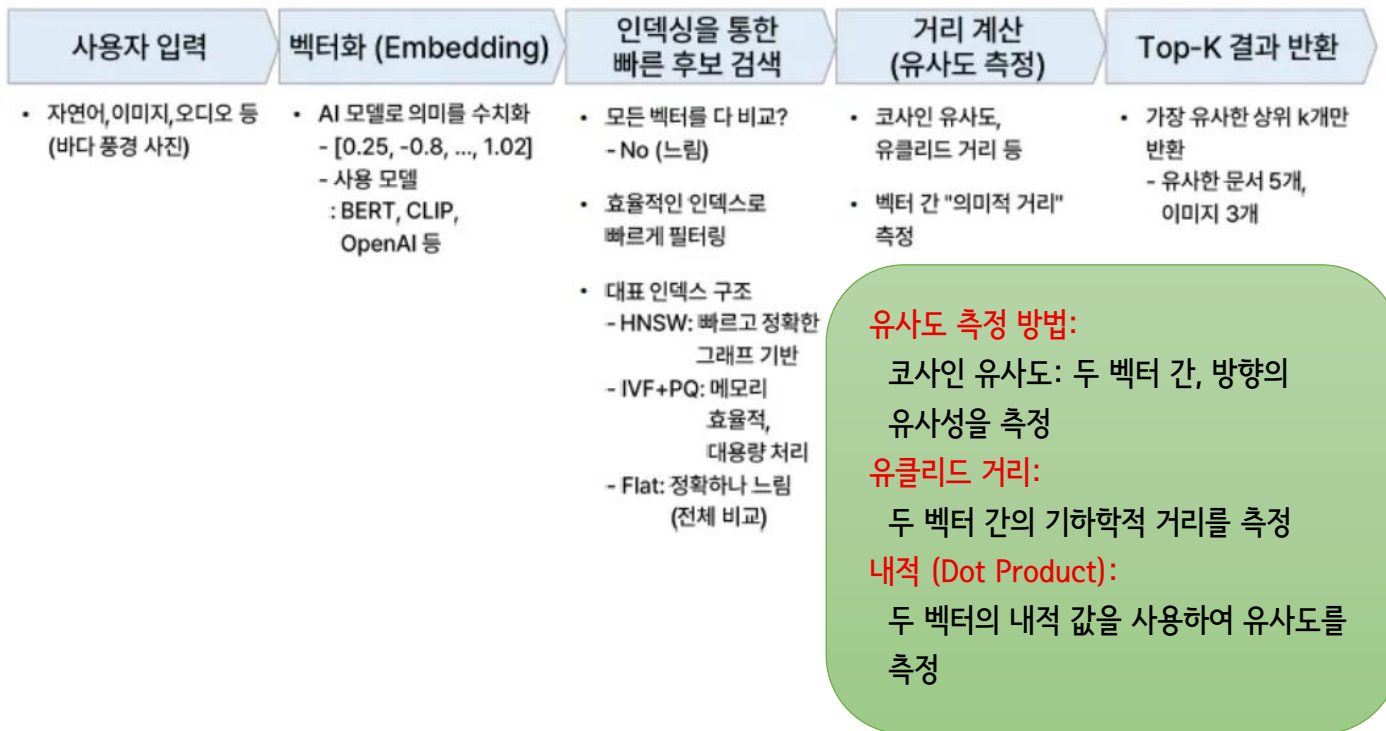
```
docker run -p 5432:5432 -d -e POSTGRES_USER=edu -e POSTGRES_PASSWORD=1234 -e  
POSTGRES_DB=edudb -v vpg:/var/lib/postgresql/data --name edupgvector  
ankane/pgvector
```



PostgreSQL + Pgvector

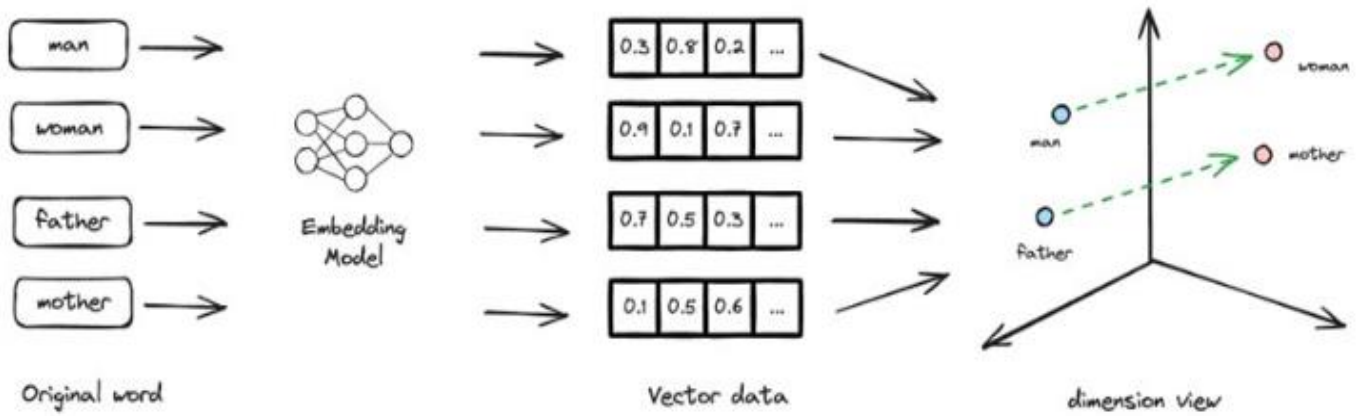


pgvector



전통 DB와 벡터 DB 비교

구분	전통 RDB	벡터 DB
목적	조건 일치 검색	의미 기반 유사 검색
쿼리 방식	SQL	벡터 유사도 기반
인덱스	B-Tree 등	HNSW, IVF 등
반환 결과	일치하는 값	유사한 결과 (확률/유사도 기반)



벡터 임베

[벡터 DB 에 접근할 때 사용하는 클라이언트 프로그램]



dbeaver.io/download/

Download

DBeaver Community 25.0.5

Released on May 18th 2025 ([Milestones](#)).

It is free and open source ([license](#)).

Also you can get it from the [GitHub mirror](#).

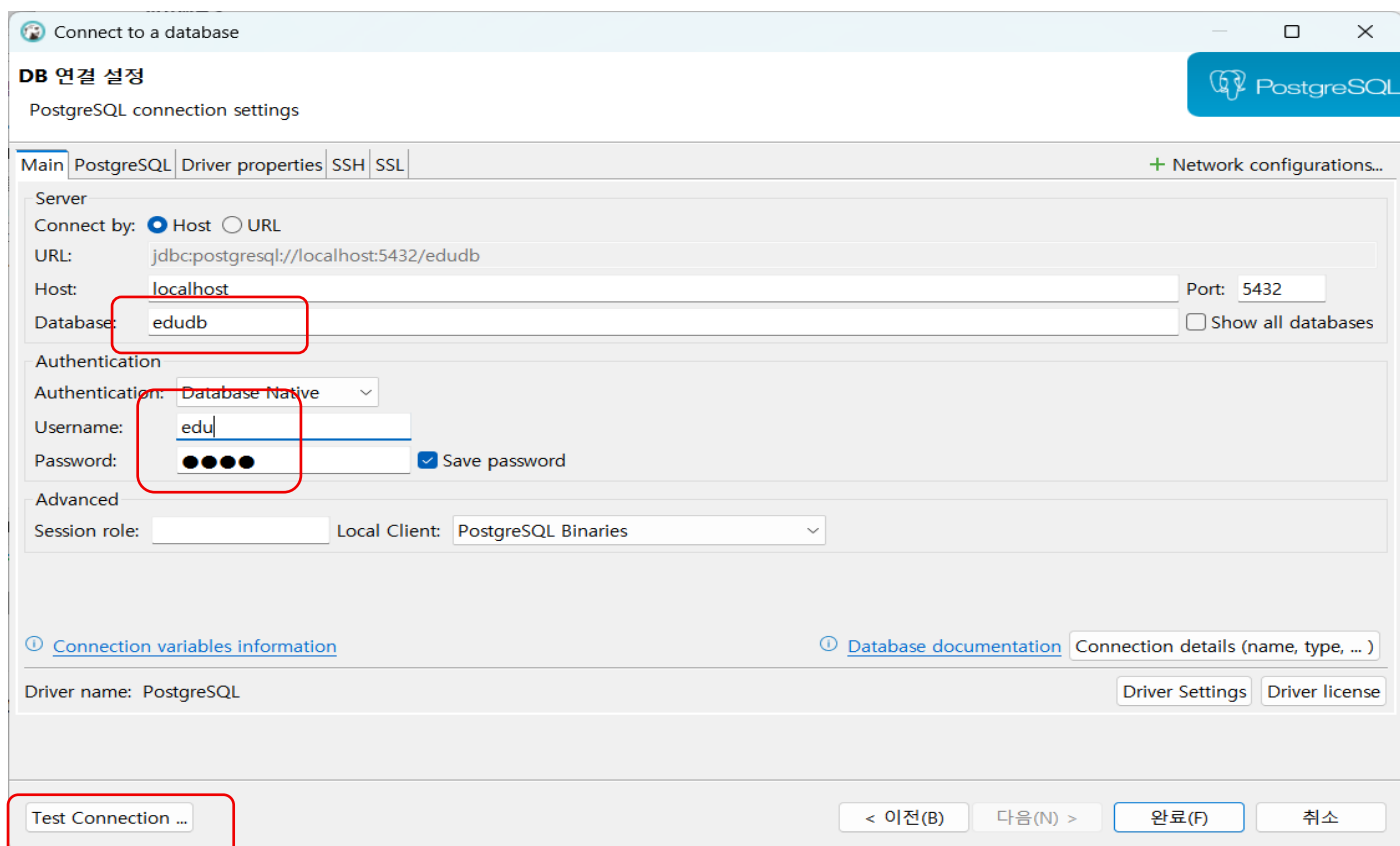
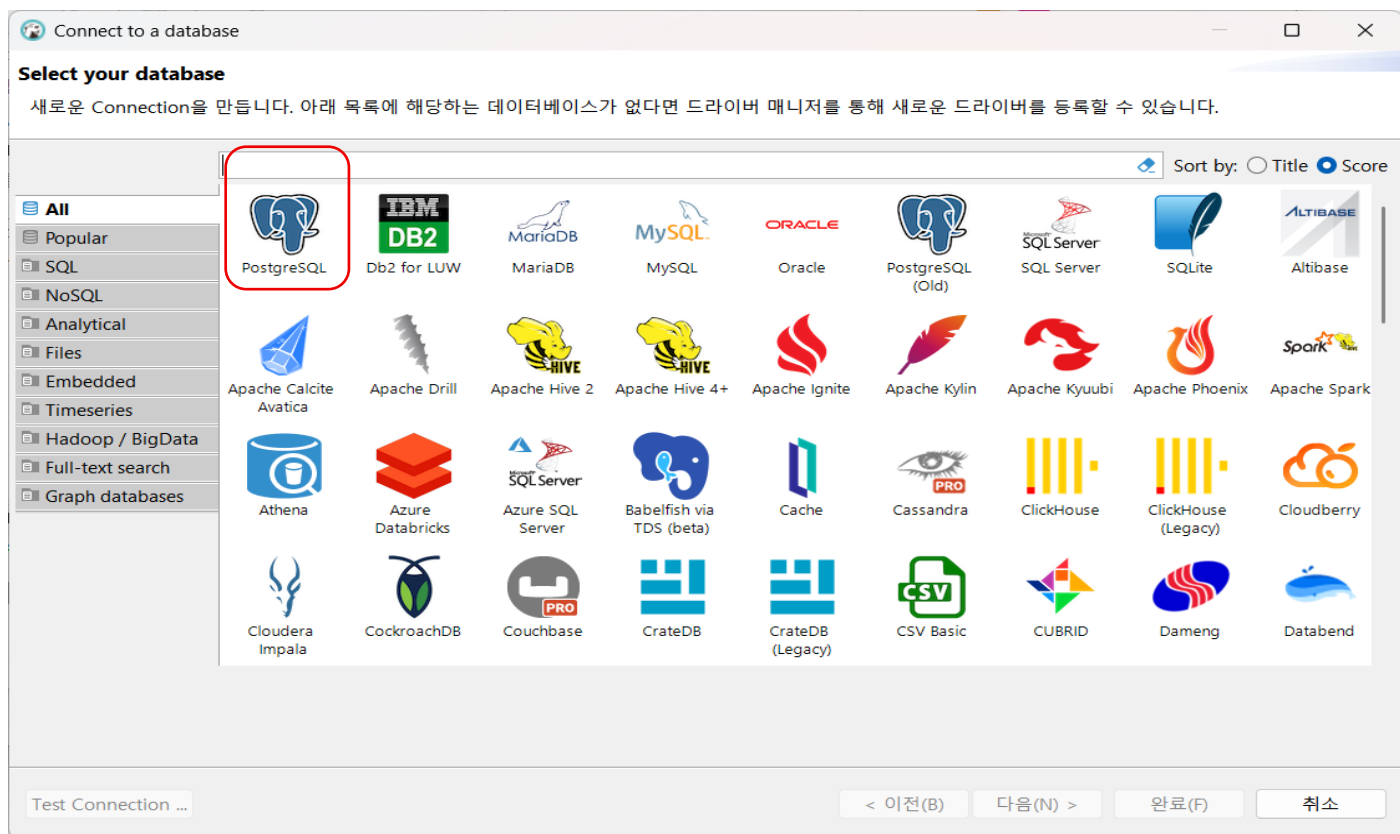
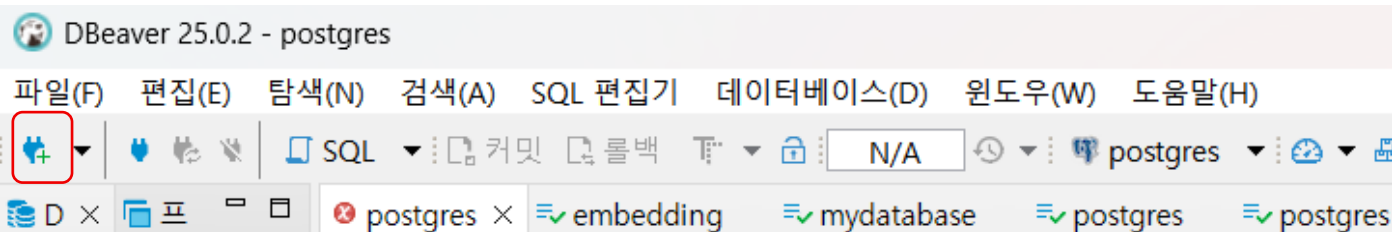
[System requirements](#).

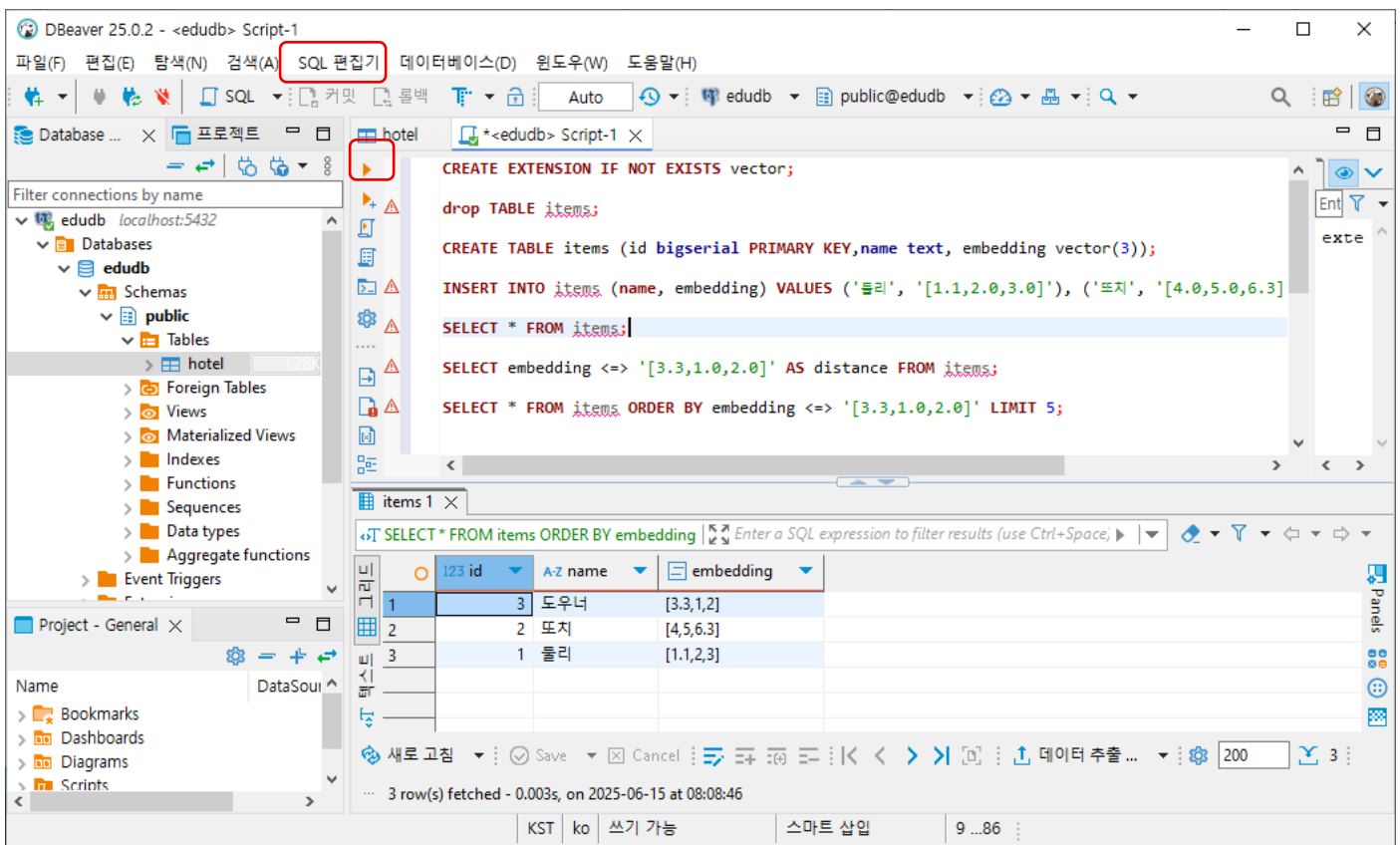
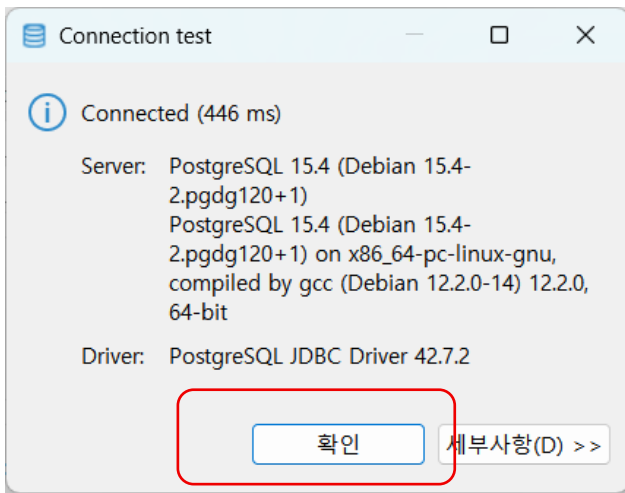
Windows

- [Windows \(installer\)](#)
- [Windows \(zip\)](#)
- [Chocolatey](#) (choco install dbeaver)
- [Install from Microsoft Store](#)

Mac OS X

- [MacOS for Apple Silicon \(dmg\)](#)
- [MacOS for Intel \(dmg\)](#)





CREATE EXTENSION IF NOT EXISTS vector;

CREATE TABLE items (id bigserial PRIMARY KEY,name text, embedding vector(3));

INSERT INTO items (name, embedding) VALUES ('둘리', '[1.1,2.0,3.0]'), ('또치', '[4.0,5.0,6.3]'), ('도우너', '[3.3,1.0,2.0]');

SELECT * FROM items;

id	name	embedding
1	둘리	[1.1,2.3]
2	또치	[4.5,6.3]
3	도우너	[3.3,1.2]

SELECT embedding <=> '[3.3,1.0,2.0]' AS distance FROM items;

Results 1	
SELECT embedding <=> '[3.3,1.0,2.0]' AS distance FROM items	
	distance
1	0.2260360379
2	0.1398406232
3	0

SELECT * FROM items ORDER BY embedding <=> '[3.3,1.0,2.0]' LIMIT 5;

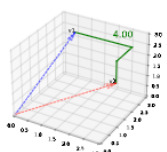
SELECT * FROM items ORDER BY embedding <=> '[3.3,1.0,2.0]' LIMIT 5			
	id	A-Z name	embedding
1	3	도우너	[3.3,1.2]
2	2	또치	[4.5,6.3]
3	1	틀리	[1.1,2.3]

SQL과 임베딩 검색을 동시에 처리할 수 있으며 임베딩 검색에 대한 처리가 기호로 구분되어 있어 매우 깔끔하다.

1. <#> : 내적 계산(inner product)
2. <=> : 코사인 거리 계산
3. <+>: L1 거리 계산(L1 distance)

Distance operators

Manhattan (L1)

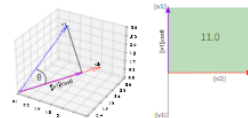


'[1, 2, 3]' <+> '[3, 1, 2]'

or

l1_distance('[1, 2, 3]', '[3, 1, 2]')

Negative inner (dot) product



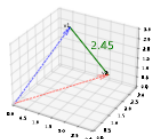
v1 <#> v2

or

inner_product(v1, v2)

* pgvector multiplies the dot product by -1, so 11 becomes -11

Euclidean (L2)

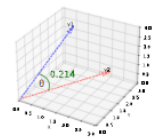


v1 <-> v2

or

l2_distance(v1, v2)

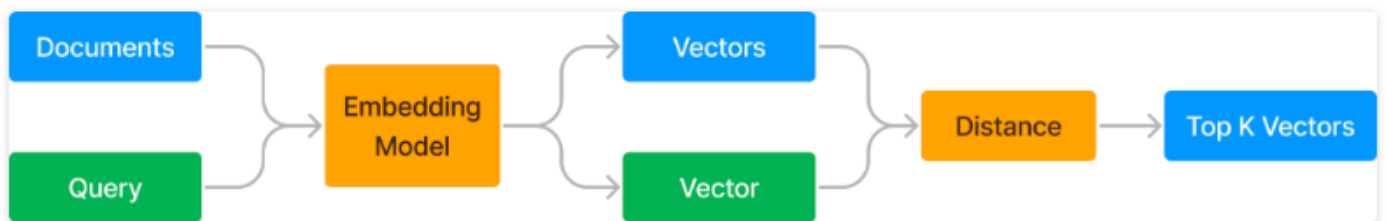
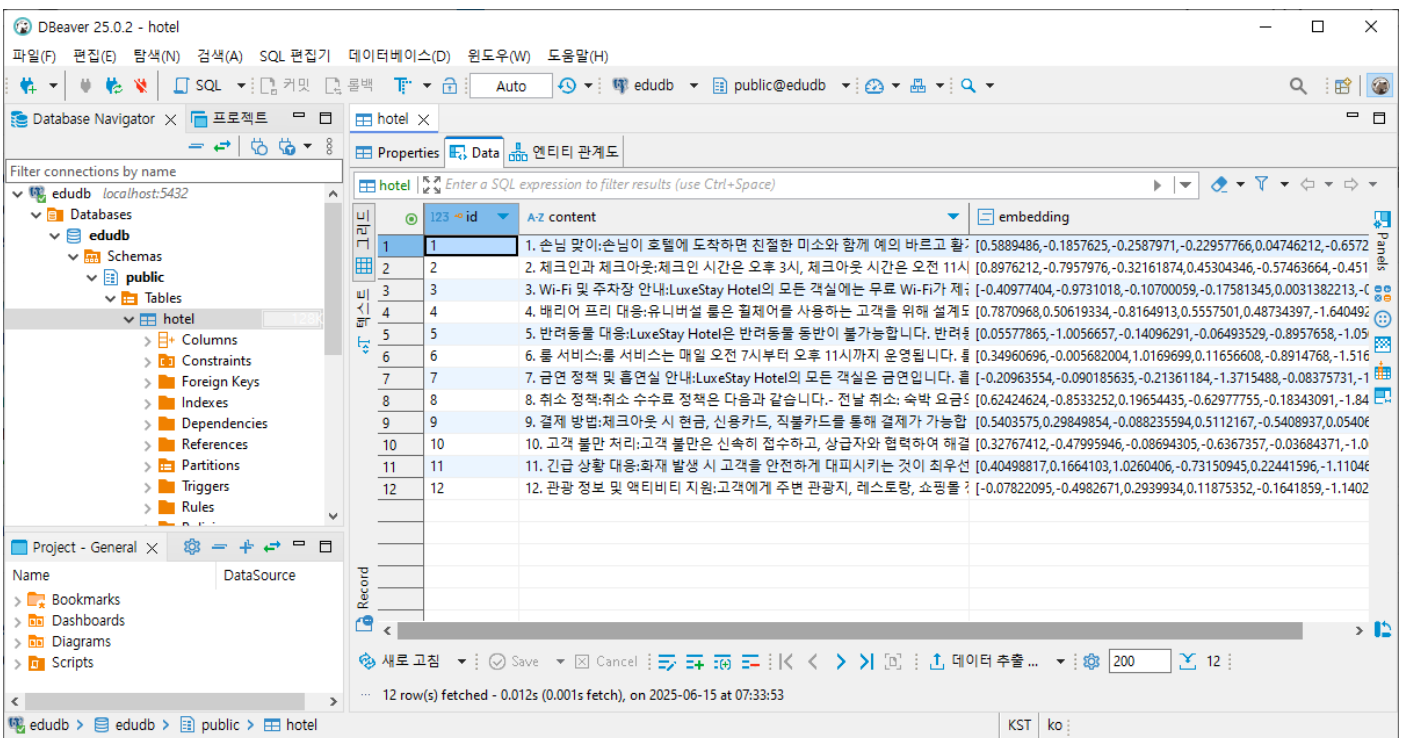
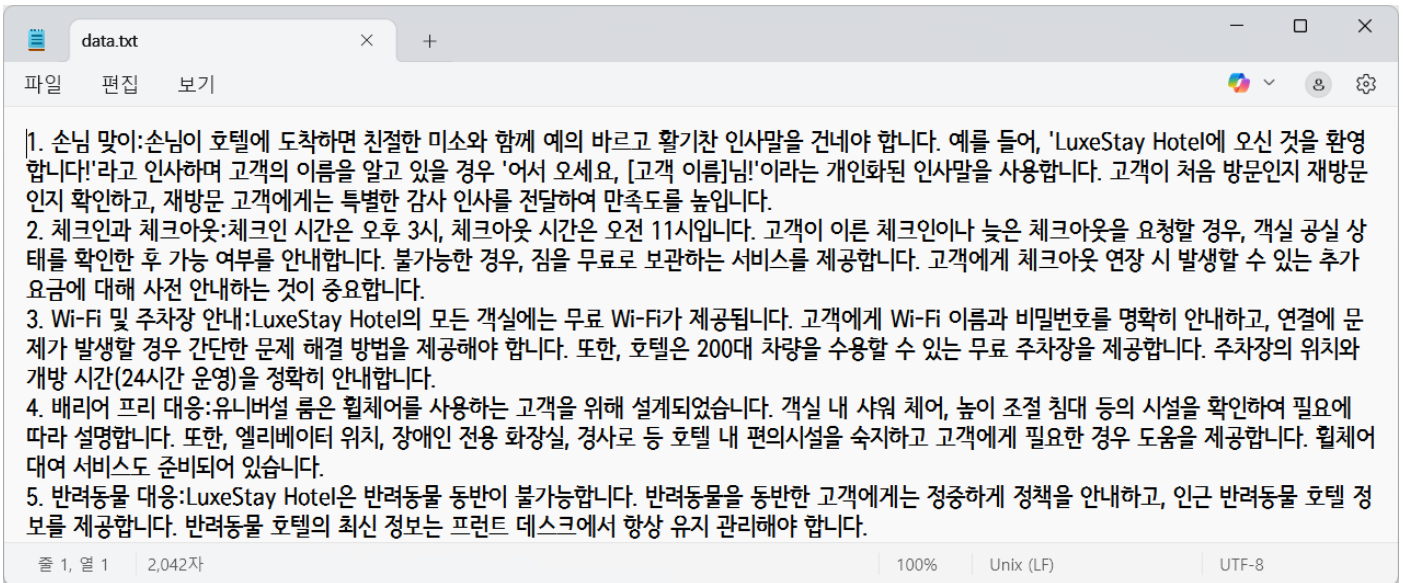
Cosine distance



v1 <=> v2

or

cosine_distance(v1, v2)



대부분의 모델들이 영어에 가장 특화되어 있기 때문에 한국어를 임베딩할 경우 일반적인 오픈 임베딩 모델들은 성능이 나오지 않기 때문에 돈을 주고 openai embedding 이나 anthropic claude embedding 을 사용하는게 가장 좋다. 하지만 오픈 모델을 사용해야 하는 상황이라면 그리고 위 모델들을 사용할 수 없다면 보통 다국어 임베딩 중 한국어를 지원하는 모델을 사용한다. 일단 한국어에 대한 성능이 좋아야 하고, 모델 사이즈가 작을 수록 좋다

1. intfloat/multilingual-e5-large

- Model size : 560M
- Limitations : Long texts will be truncated to at most 512 tokens.

2. BAAI/bge-multilingual-gemma2

- Model size : 9.24B

3. Alibaba-NLP/gte-multilingual-base

- Model Size: 305M
- Embedding Dimension: 768
- Max Input Tokens: 8192

4. jinaai/jina-embeddings-v3

- Model size : 572M
- Handle long input sequences up to 8192 tokens.
- Supports flexible embedding sizes (32, 64, 128, 256, 512, 768, 1024)

5. klue/bert-base

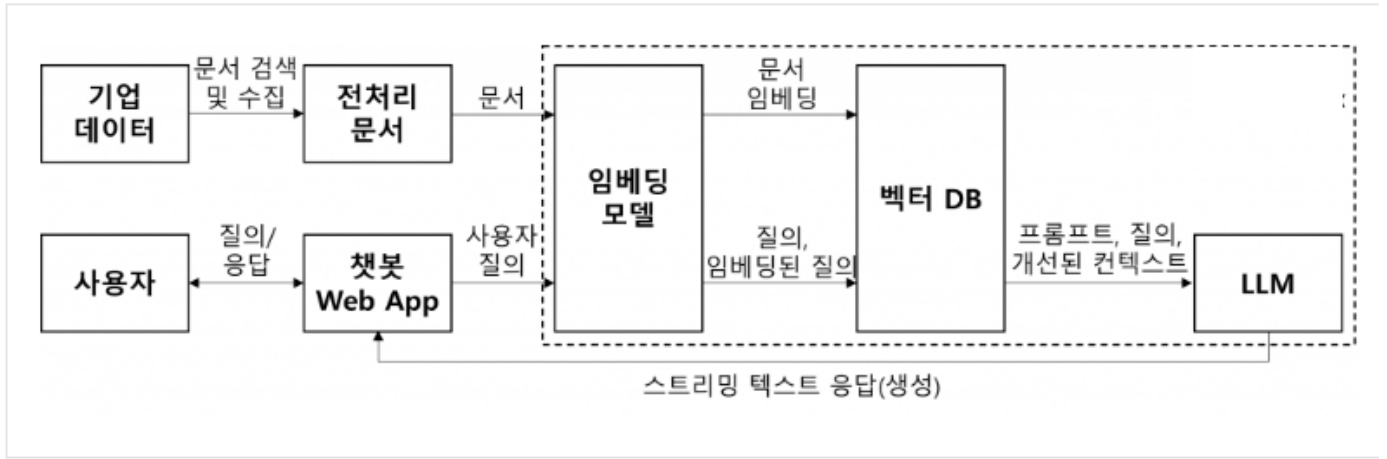
- Model size : 111M

6. klue/roberta-base

- Model size : 111M

7. nlpai-lab/KoE5 · Hugging Face

- Model size : 560M
- Limitations : Long texts will be truncated to at most 512 tokens.



RAG

RAG (Retrieval-Augmented Generation)는 **검색 기반 생성** 모델로, **정보 검색(retrieval)**과 **언어 생성(generation)**을 결합한 기술입니다. 이 기술은 주로 대규모 언어 모델(LLM)이 특정 질문에 대해 더 정확하고 정보에 근거한 답변을 생성할 수 있도록 도와줍니다.

RAG는 크게 두 가지 주요 단계로 나눌 수 있습니다:

1.정보 검색 (Retrieval):

1. 사용자가 입력한 질문에 대해 관련된 정보를 외부 데이터베이스나 문서에서 검색합니다.
2. 이 검색은 특정 질의와 관련된 **문서나 텍스트 조각**을 찾기 위해 **벡터 검색(Vector Search)** 기술을 사용할 수 있습니다. 예를 들어, PGvector나 FAISS와 같은 검색 엔진을 사용하여, 데이터베이스에 저장된 문서들 중 관련도가 높은 정보를 검색합니다.

2.언어 생성 (Generation):

1. 검색된 정보를 바탕으로, ****대형 언어 모델(LLM)****이 사용자에게 더 자연스러운 형태의 답변을 생성합니다.
2. 이 과정에서 모델은 단순히 검색된 정보를 나열하는 것이 아니라, 검색된 정보를 **통합하여 적절한 텍스트**로 생성합니다. 이를 통해 보다 정확하고 구체적인 답변을 제공할 수 있습니다.

응답의 결과를 더 신뢰하는 RAG

Dice survey, what percentage of AI experts said they would consider changing jobs in 2025?

질문 내용 ▾

다이스 조사, AI 전문가의 몇 %가 2025년 중 이직 고려한다고 했어?

현재까지 공개된 자료에 따르면, AI 전문가들이 2025년에 이직을 고려하는 비율에 대한 다이스(Dice)의 설문조사 결과는 확인되지 않습니다. 다이스는 2024년 기술 전문가들을 대상으로 한 설문 조사에서, 지난해 대비 해고된 기술 전문가의 비율이 60% 증가했으며, 실직을 경험한 기술 전문가 중 82%가 즉시 새로운 일자리를 찾기 시작했다고 보고했습니다. 그러나 이 조사에서는 AI 전문가들의 2025년 이직 의향에 대한 구체적인 수치는 제공되지 않았습니다.

OpenAI가 잘못된 답변을 제공할 때 이를

"hallucination"이라고 부릅니다.

AI가 실제로 존재하지 않는 정보나 잘못된 사실을 제시하는 오류를 의미합니다. 이 현상은 특히 대형 언어 모델(LLM)에서 흔하게 발생하며, 모델이 자신감 있게 잘못된 답변을 생성할 때 이를 "hallucination"이라고 표현합니다.

Hallucination : 헛소리, 환각

SPRI AI Brief |
2024-11월호

PDF

RAG(Retrieval-Augmented Generation) 기술을 사용하면 AI의 헛소리(hallucination)를 줄이는 데 도움이 됩니다.

RAG는 외부 데이터베이스나 문서 저장소에서 신뢰할 수 있는 정보를 검색하고, 이를 기반으로 답변을 생성하는 방식입니다. 이렇게 하면 모델이 맥락에 맞는 정보를 활용해 응답을 제공할 수 있어 헛소리를 줄일 수 있습니다.

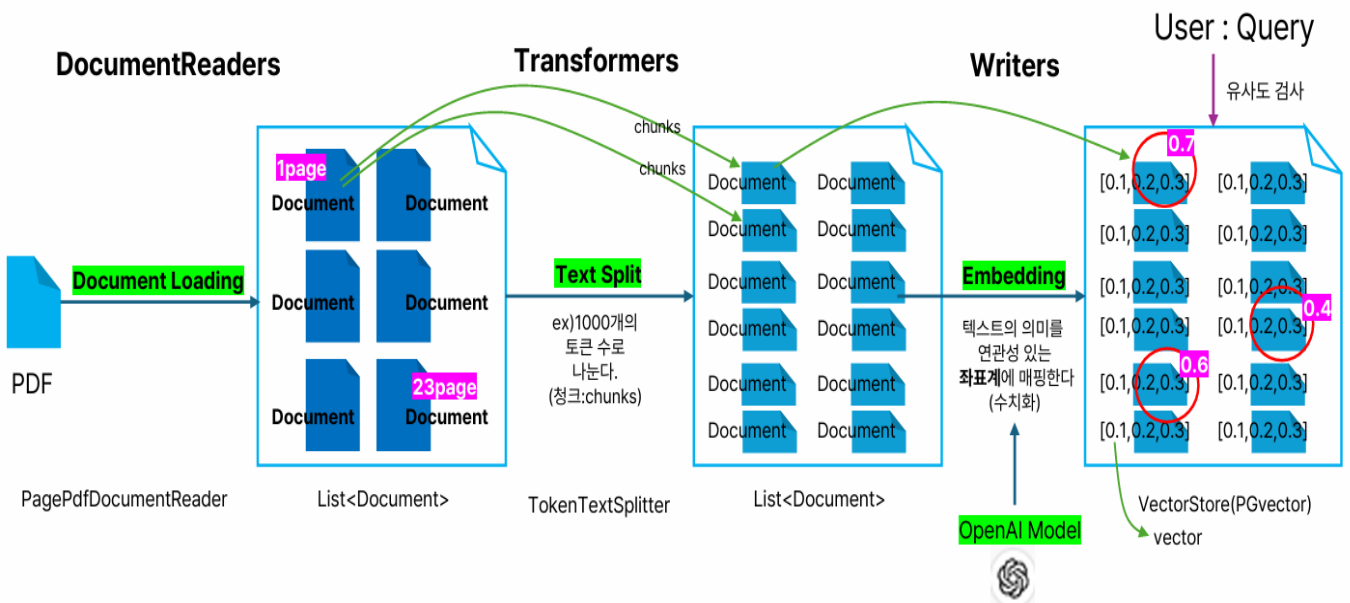
다이스 조사, AI 전문가의 73%는 2025년 중 이직 고려

KEY Contents

- 다이스에 따르면, AI 전문가의 73%는 2025년 이직을 계획 중이며, 58%는 2024년 중 현재보다 더 나은 일자리를 찾을 자신이 있다고 응답해 여타 기술 전문가 대비 직업 전망을 낙관
- AI 전문가들은 여타 기술 전문가 대비 AI 도구 사용에도 적극적이며, 업무에 생성AI가 상당한 영향을 미친다는 응답도 36%로 여타 기술 전문가(22%) 대비 높은 수치를 기록

PDF Embedding

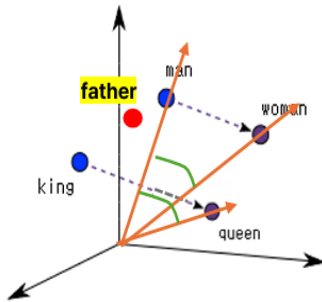
implementation 'org.springframework.ai:spring-ai-pdf-document-reader'



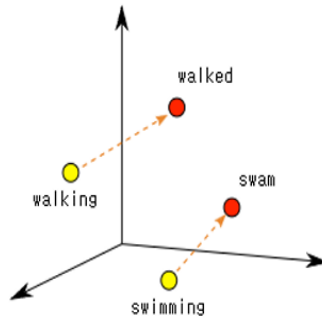
임베딩(Embedding)

단어를 3차원 공간에 수치로 표시하는 방법

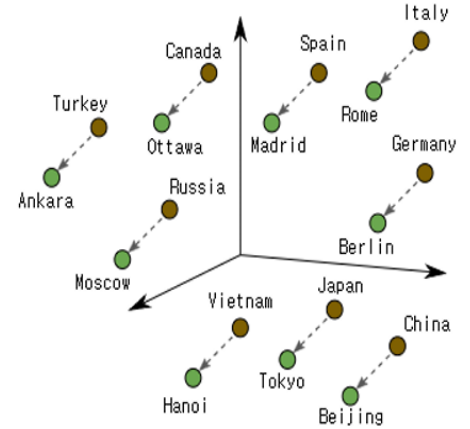
1 x 1536 차원에 표현



단어 간의 의미적 관계



동사의 시제 변화에 따른 관계



나라와 수도 관계

코사인 유사도

코사인 유사도로 실제 확인해야 함:

예를 들어, 임베딩 모델에서 각 단어의 벡터를 다음과 같이 정의했다고 가정합시다:

- `man` : [0.5, 0.6, 0.7]
- `queen` : [0.4, 0.7, 0.6]
- `woman` : [0.5, 0.8, 0.7]

이 경우:

1. `man` ↔ `woman` 의 코사인 유사도:

$$\cos \theta = \frac{\vec{man} \cdot \vec{woman}}{\|\vec{man}\| \|\vec{woman}\|}$$

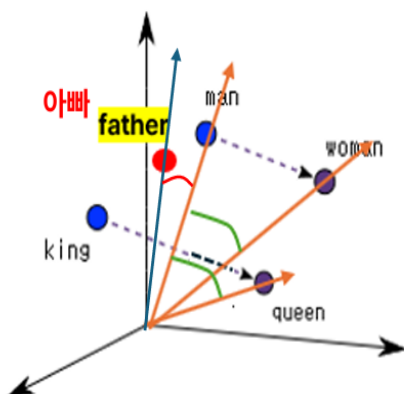
계산 결과: 0.99 (높은 유사도)

2. `man` ↔ `queen` 의 코사인 유사도:

$$\cos \theta = \frac{\vec{man} \cdot \vec{queen}}{\|\vec{man}\| \|\vec{queen}\|}$$

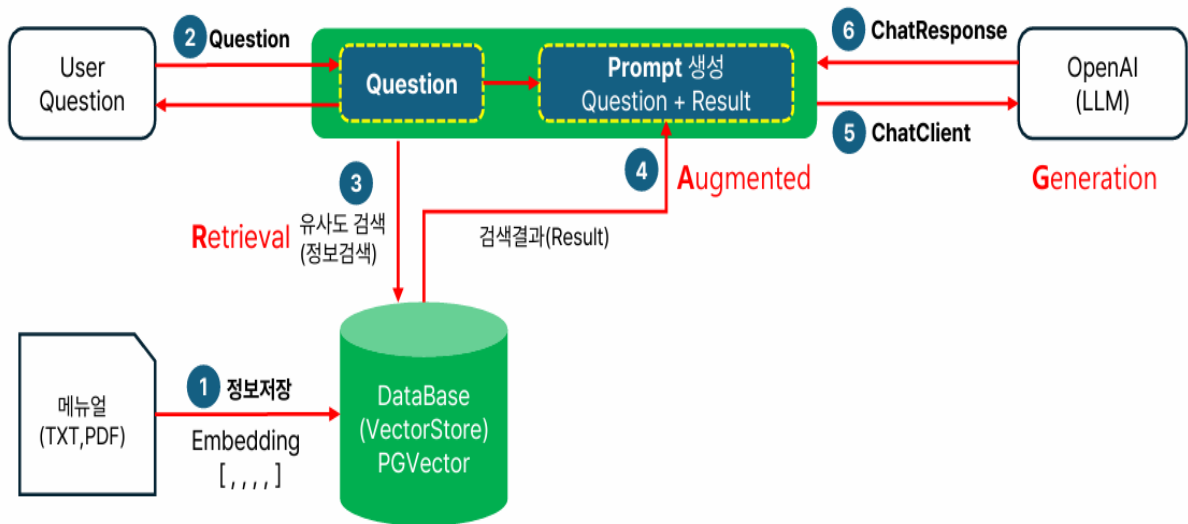
계산 결과: 0.95 (조금 낮은 유사도)

이처럼 계산 결과를 통해 실제로 `man` ↔ `woman` 이 더 가까운 관계라는 것을 알 수 있습니다.



RAG?

RAG는 **Retrieval-Augmented Generation**의 약자로, 정보 검색과 생성 AI를 결합한 기술입니다. RAG는 대규모 언어 모델(LLM)과 데이터베이스를 조합하여, 질문에 대해 더 정확하고 신뢰할 수 있는 답변을 생성하는 데 사용됩니다.



RAG 유사도 검사

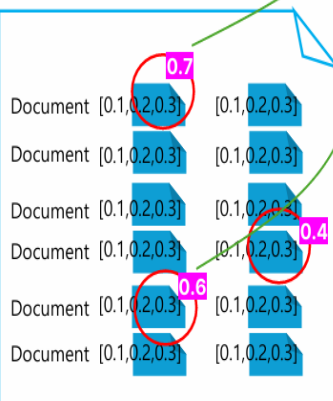
User : Query

: 다이슨 조사, AI 전문가의 몇%가 2025년 중 이직을 고려 한다고 했어?

Embedding

```
List<Document> documents =  
vectorStore.similaritySearch(SearchRequest  
.query(question)  
.withTopK(2));
```

VectorStore(PGvector)



Prompt 만들기

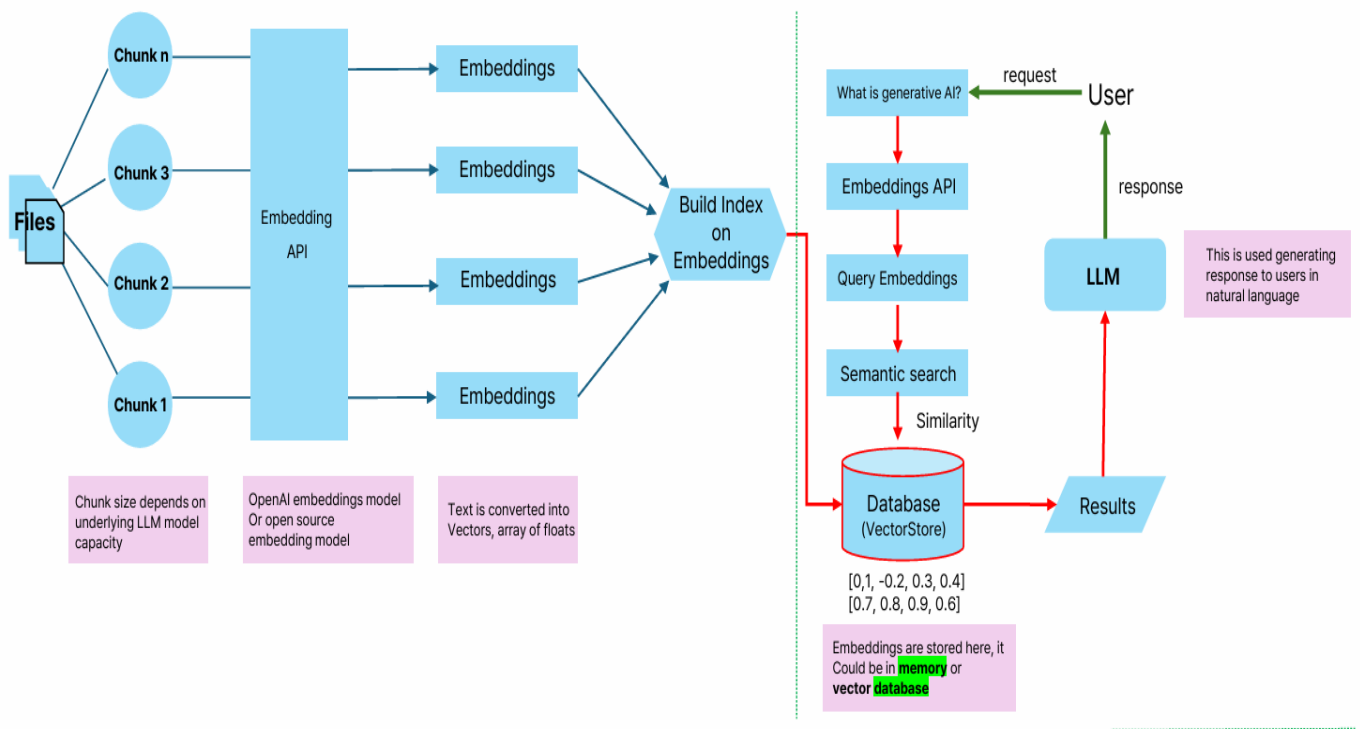
```
private String prompt = ""  
이 질문에 답변하는 것입니다.  
질문에 정확하게 답변하기 위해 문서에 있는 정보를 사용해야 합니다.  
만약 정보가 부족하거나 문서에서 답을 찾을 수 없다면,  
알지 못한다고 간단히 답변하세요.  
  
질문:  
{input}  
  
문서:  
{documents}  
,,,,,  
,
```

ChatClient
ChatModel

OpenAI LLM



RAG Process



RAG기반 호텔 AI 챗봇 서비스

호텔 챗봇은 고객의 질문에 답변하거나 필요한 정보를 제공하는 AI 기반 서비스입니다.

User:

취소정책을 알려주세요

Bot:

호텔의 취소 정책은 다음과 같습니다:

- 전날 취소 시 숙박 요금의 30%가 부과됩니다.
- 당일 취소 시 숙박 요금의 50%가 부과됩니다.
- 연락 없이 취소할 경우 숙박 요금의 100%가 부과됩니다.

이 정책은 모든 예약에 적용되며, 예약 시 고객에게 명확히 고지됩니다. 만약 특별한 사정이 있는 경우, 관리자의 승인을 받아 유연하게 대응할 수 있습니다. 추가적인 문의사항이 있으시면 언제든지 말씀해 주세요.

Type your message...

Send