안동대학교 2024 릴레이 SW단기특강

AI 관점에서의 데이터베이스 및 처리

벡터 데이터베이스

강사 양석환

국립안동대학교
ANDONG NATIONAL UNIVERSITY

벡터 데이터베이스 소개

벡터 데이터베이스 개요

- 벡터 데이터베이스(Vector Database)란?
  - 방대한 양의 고차원 데이터를 벡터 형태로 저장하고 처리하기 위한 특수한 데이터베이스
  - 정보를 벡터로 저장
  - 각 벡터는 특성이나 품질을 기반으로 개체를 설명하는 수학적 데이터 표현을 의미함
    - 벡터 임베딩이라고도 알려진 데이터 객체의 수치 표현
  - 벡터 임베딩의 강력한 기능을 활용하여 이미지, 텍스트 또는 센서 데이터와 같은 비정형 데이터와 반정형 데이터로 구성된 대규모 데이터 세트를 색인하고 검색

벡터 데이터베이스 개요

- **벡터 데이터베이스의 주요 특징과 장점**
  - **고차원 벡터 저장**
    - 데이터 항목은 특성 벡터(feature vector)로 표현됨
    - 특성 벡터를 기반으로 데이터 검색, 분류, 군집화 등의 작업을 수행할 수 있음
    - 수백 개 이상의 차원을 가진 벡터 데이터도 효율적으로 처리할 수 있음
  - **벡터 임베딩 관리**
    - 벡터 임베딩을 관리하기 위해 구축됨
    - 비정형 및 반정형 데이터 관리에 특화되어 있음

벡터 데이터베이스 개요

- 유사성 검색 (Similarity Search): 주어진 쿼리 벡터와 가장 유사한 벡터를 빠르게 검색할 수 있음
- 분산 처리: 대규모 데이터셋에 대한 분산 처리와 병렬 처리를 지원
- 실시간 쿼리: 빠른 응답 시간을 위해 실시간으로 벡터 검색과 분석이 가능함
- 확장 가능성: 확장 가능한 구조로 설계되어 대규모 데이터 세트를 처리할 수 있음
- 동적 데이터 변경: 데이터 변경이 발생해도 유연하게 대응할 수 있음
- 보안 기능: 데이터 보안을 위한 기능 제공

벡터 데이터베이스 개요

- 벡터 데이터베이스의 작동 방식
  - 다양한 알고리즘을 사용하여 벡터 임베딩을 색인하고 쿼리하는 방식으로 작동
  - 빠른 검색이 가능
  - 유사한 벡터 항목을 찾을 수 있음

  - 주요 작동 단계
    1. 색인: 벡터를 주어진 데이터 구조에 매핑하여 색인. 해싱, 양자화, 그래프 기반 기술을 사용할 수 있음
    2. 쿼리: 쿼리 벡터와 인덱스 벡터를 비교하여 최근접 벡터 항목을 결정
       - 코사인 유사성, 유클리드 거리, 점 곱과 같은 유사성 측정 방법을 사용함
    3. 후처리: 최근접 항목의 순위를 다시 매기는 단계. 메타데이터를 기반으로 필터링 수행

벡터 데이터베이스 개요

- 벡터 데이터베이스의 활용 범위
  - AI, 머신 러닝, 이미지 처리, 자연어 처리 등 다양한 분야에서 중요한 역할 수행
  - 특히 머신 러닝 모델의 학습 데이터 관리와 유사 데이터의 검색에 있어서 매우 중요한 역할을 담당
  - 데이터 객체 간의 유사성을 신속하고 효율적으로 파악함으로써 다양한 분석 작업을 수행할 수 있음

벡터 데이터베이스 개요

- 벡터 데이터베이스와 머신 러닝의 결합 방식
  - 머신 러닝 모델을 사용하여 데이터를 벡터 형태로 변환하고, 이를 벡터 데이터베이스에 저장
    - 머신 러닝 모델의 학습 데이터 준비와 유사 데이터의 검색이 효과적으로 이루어질 수 있음
  - 벡터 데이터베이스에서 검색된 결과를 바탕으로 머신 러닝 모델의 학습이나 예측 수행
    - 실시간 예측 시스템 구축에 매우 효과적
  - 벡터 데이터베이스는 데이터 객체 간의 유사성을 효율적으로 파악하고 다양한 분석 작업을 수행할 수 있는 머신 러닝과 인공지능 애플리케이션의 성능을 극대화하는 데 필수적인 기술

벡터 데이터베이스의 필요성

- **LLM은 이미 벡터임베딩 과정이 완료된 데이터를 이용하여 사전학습을 수행한 모델인데 왜 벡터 DB가 필요한가?**

  - **LLM (Large Language Models, 거대 언어 모델)**
    - LLM은 방대한 양의 자연어 데이터를 처리하고 종종 사람이 생성한 텍스트와 구별할 수 없는 응답을 생성할 수 있는 인공 지능 시스템을 가리킴
    - 자기 지도 학습이나 반자기 지도 학습을 사용하여 레이블링 되지 않은 상당한 양의 텍스트로 훈련된 모델
    - 인간의 언어 텍스트를 이해하고 생성할 줄 아는 딥러닝 알고리즘으로 AI 챗봇 기술을 가능하게 하는 주요 요소

벡터 데이터베이스의 필요성

- **왜 벡터 데이터베이스가 필요한가?**
  - LLM은 이미 사전 학습된 모델이지만, 실제 응용 프로그램에서는 여전히 추가적인 데이터 처리와 관리가 필요
  - 벡터 DB는 이러한 작업을 보다 효율적으로 수행할 수 있도록 도와줌
  - 또한, LLM은 사전 학습을 위해 대규모 데이터셋을 사용하지만, 실제 응용 프로그램에서는 특정한 도메인이나 작업에 대해 추가적인 데이터를 필요로 할 수 있음
  - 이러한 추가 데이터를 벡터 DB에 저장하고 관리함으로써 LLM을 더욱 효과적으로 활용할 수 있음

벡터 데이터베이스의 필요성

- **LLM을 위해 벡터 데이터베이스가 지원해 주는 기능**
  - **효율적인 데이터 관리와 검색**
    - **LLM은 텍스트를 이해하고 생성하는 데 탁월하지만, 데이터 관리 측면에서는 한계가 있음**
      - 벡터 DB는 LLM이 생성한 벡터 임베딩을 효율적으로 저장하고 검색할 수 있도록 도와 줌
    - **벡터 DB는 벡터 공간에 데이터를 저장하고 쿼리할 수 있는 구조를 제공함**
      - 기존의 텍스트 데이터베이스에 비해 훨씬 효율적인 데이터 저장 및 검색을 가능하게 함
    - **LLM과 같은 거대한 모델은 수십억 혹은 수백억 개의 파라미터를 가지고 있으며, 이 모델들은 대량의 데이터에 의존함**
      - 벡터 DB를 사용하면 이러한 대용량 데이터에 대한 빠른 접근과 검색이 가능함

벡터 데이터베이스의 필요성

- **빠른 유사성 검색**
  - **벡터 DB가 제공하는 유사성 기반 검색을 통해 사용자는 더 관련성 높은 결과를 빠르게 얻을 수 있음**
    - 벡터 DB는 데이터 간의 유사성을 직접 저장하기 때문에 검색 속도가 훨씬 빠름
  - **LLM은 문장 또는 단어의 의미를 벡터로 표현하며, LLM은 명령 수행을 위하여 텍스트의 의미적 유사성을 파악해야 함**
  - **벡터 DB는 이러한 벡터 표현을 활용하여 유사성 검색을 수행할 수 있음**
    - 예: 특정 단어 또는 문장과 유사한 의미를 가진 다른 단어 또는 문장을 검색하는 것이 가능
  - **이를 통해 LLM이 생성한 결과를 보완하거나 확장할 수 있음**
  - **빠른 유사성 검색 지원은 LLM이 대화 맥락과 관련된 정보를 빠르게 찾아 정확하고 유익한 응답을 생성하는 데 도움이 됨**

- **자연어 처리 지원**
  - **LLM은 자연어 처리를 수행해야 함**
  - **벡터 DB는 자연어 처리에 필요한 다양한 기능을 제공하며, 이를 통해 LLM의 성능을 향상 시킬 수 있음**

벡터 데이터베이스의 필요성

- 데이터 캐싱 및 전처리
  - LLM을 학습시키기 위해서는 막대한 양의 데이터가 필요함
  - 이러한 데이터는 여러 소스에서 수집되며, 이를 벡터로 변환하고 적절하게 저장하는 전처리 과정이 필요함
  - 벡터 DB는 이러한 전처리 및 데이터 캐싱 작업을 효율적으로 수행할 수 있음
    - 예: 웹 크롤링된 텍스트 데이터를 벡터로 변환하여 저장하고, 필요할 때 마다 빠르게 검색할 수 있음
  - 벡터 DB는 대용량 데이터를 효율적으로 저장하고 검색할 수 있기 때문에 LLM의 학습과 추론에 필요한 데이터를 빠르게 제공할 수 있음

- 효율적인 데이터 표현
  - 벡터 DB는 고차원 데이터를 효율적으로 표현함
  - LLM은 텍스트, 이미지, 음성 등 다양한 형태의 데이터를 처리해야 함
  - 벡터 DB는 이러한 다양한 데이터를 공통적인 벡터 표현으로 변환하여 LLM이 데이터 간의 관계를 쉽게 이해하도록 함

벡터 데이터베이스의 필요성

- 메타데이터 관리
  - 벡터 DB는 메타데이터를 저장하고 필터링할 수 있어 데이터 관리를 용이하게 함
  - GPT3 이후의 LLM은 메타학습 등의 기능을 지원함으로써 그 성능을 구현하고 있으므로 메타학습을 지원할 수 있는 메타 데이터의 관리가 중요함

- 개인 맞춤형 추천 지원
  - 벡터 DB는 LLM이 사용자에게 개인 맞춤형 추천을 제공하는 데 도움이 됨
  - 사용자의 과거 행동과 선호도를 벡터 DB에 저장하여 LLM이 사용자에게 가장 관련성이 높고 유용한 정보를 추천할 수 있도록 지원

벡터 데이터베이스의 필요성

- 지식 그래프 구축
  - 벡터 DB는 LLM이 지식 그래프를 구축하는 데 사용될 수 있음
  - 지식 그래프는 개념과 엔티티 간의 관계를 나타내는 정보 네트워크
  - 벡터 DB는 LLM이 개념과 엔티티를 벡터로 표현하고 그들 간의 관계를 저장하여 지식 그래프를 구축하도록 함
  - 이는 LLM이 세상에 대한 이해를 높이고 더욱 지능적인 응답을 생성하는 데 도움이 됨

- 확장성
  - 벡터 DB는 확장 가능한 구조로 설계되어 LLM에서 요구하는 대규모 데이터 세트를 처리할 수 있음

- 보안
  - LLM은 민감한 정보를 포함할 수 있는 수많은 데이터를 사용하기 때문에 강력한 보안 기능이 요구됨
  - 벡터 DB는 자체적으로 보안기능을 제공하기때문에 LLM을 지원하기가 용이함

벡터 데이터베이스의 필요성

- **LLM의 사전 학습과 벡터 데이터베이스의 필요성**
  - **LLM은 이미 벡터 임베딩 과정이 완료된 데이터를 이용하여 사전 학습을 수행하지만, 벡터 DB는 다음과 같은 이유로 여전히 필요함**
    - 새로운 데이터 학습
      - 학습에 사용된 데이터 뿐만 아니라 새로운 데이터에 대해서도 빠르게 검색하고 추론할 수 있어야
      - LLM은 지속적으로 새로운 데이터를 학습하여 성능을 향상시켜야 함
      - 벡터 DB는 새로운 데이터를 효율적으로 저장하고 검색하여 LLM이 새로운 정보를 빠르게 학습하도록 지원
    - 데이터 품질 향상
      - 벡터 DB는 데이터 품질을 향상시키는 데 사용될 수 있음
      - 벡터 DB는 데이터의 이상치나 오류를 식별하고 제거하는 데 도움이 되며, LLM의 학습 성능을 향상시킴

벡터 데이터베이스의 필요성

- 데이터 탐색 및 분석
  - 벡터 DB는 LLM이 데이터를 탐색하고 분석하는 데 도움이 됨
  - 벡터 DB는 데이터의 숨겨진 패턴과 관계를 발견하여 LLM이 세상에 대한 이해를 높이고 더욱 지능적인 응답을 생성하도록 지원함

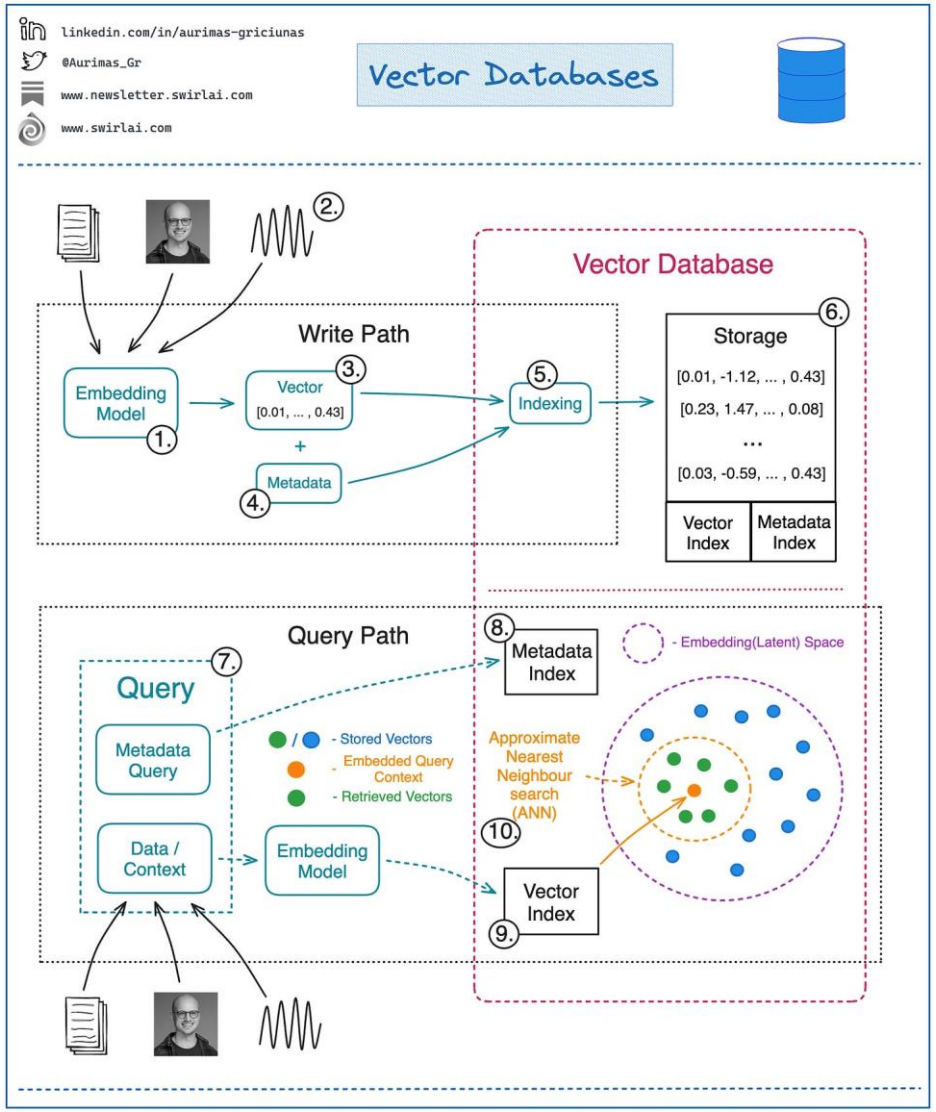
벡터 데이터베이스의 필요성

- 결론적으로
  - 벡터 DB는 인공지능 기술, 특히 LLM의 발전에 필수적인 도구
  - LLM이 학습한 데이터는 매우 방대하지만, 모든 데이터를 저장하고 관리하기는 어려우며 벡터 DB를 이용하면 이러한 문제를 해결할 수 있음
  - 벡터 DB는 LLM이 데이터를 빠르고 효율적으로 처리하고 이해하도록 하여 더욱 정확하고 유익한 응답을 생성하도록 도와 줌
  - LLM이 학습한 데이터는 다양한 형태와 형식을 가지고 있으며, 벡터 DB는 이러한 다양한 형태와 형식의 데이터를 효과적으로 저장하고 관리할 수 있음
  - 벡터 DB를 이용하면 새로운 데이터를 추가하거나 기존 데이터를 수정함으로써 LLM의 지속적인 학습과 데이터 품질 향상에도 중요한 역할을 수행함

벡터 데이터베이스의 작동 방식

- **데이터 쓰기/업데이트하기**

1. 벡터 임베딩을 생성하는 데 사용할 머신러닝 모델을 선택
2. 텍스트, 이미지, 오디오, 표 형식 등 모든 유형의 정보를 임베드 할 수 있음. 임베딩 모델은 데이터 유형에 따라 선택
3. 임베딩 모델을 통해 전처리된 데이터를 실행하여 데이터의 벡터 표현 획득
4. 대부분의 최신 벡터 데이터베이스는 벡터 임베딩과 함께 추가 메타데이터를 저장할 수 있으며, 이 데이터는 나중에 ANN 검색 결과를 사전 필터링하거나 사후 필터링하는 데 사용됨

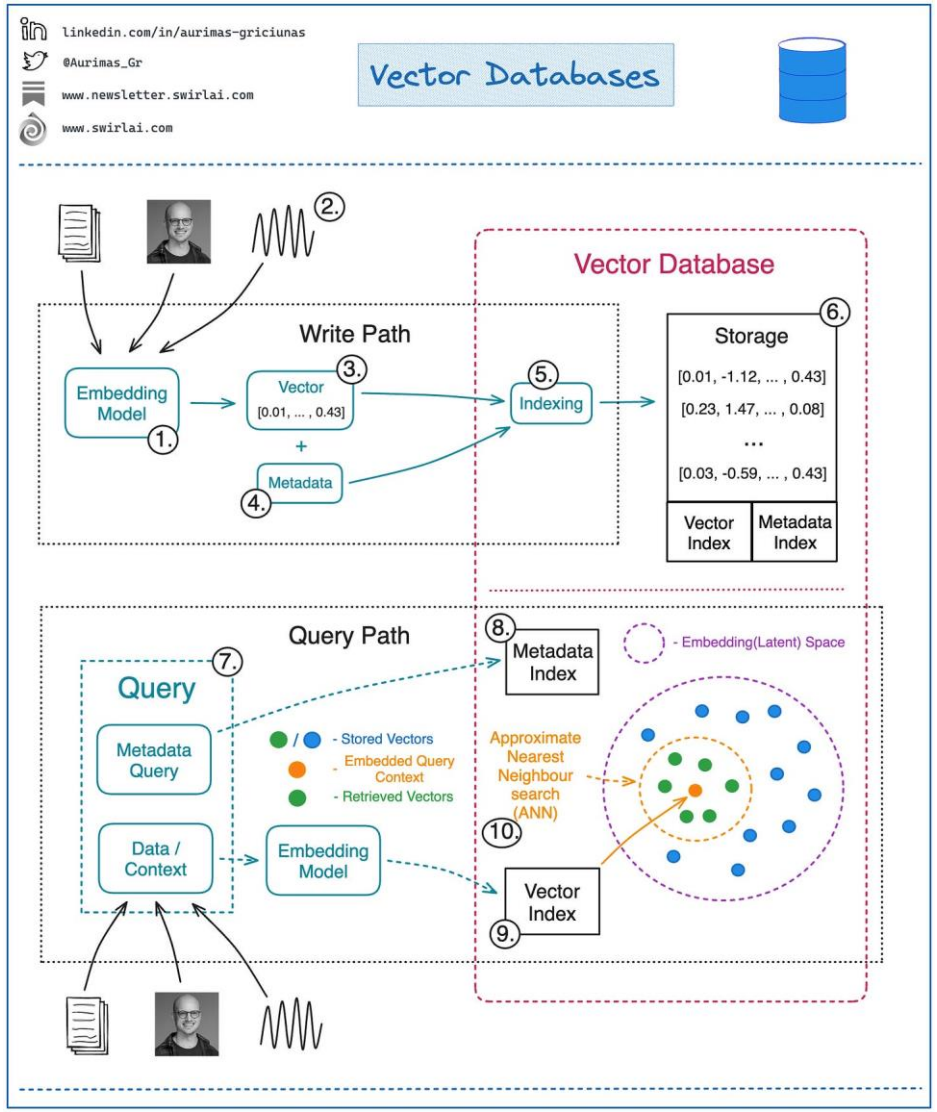


벡터 데이터베이스의 작동 방식

5. 벡터 데이터베이스 인덱스는 벡터 임베딩과 메타데이터를 별도로 수신

- 인덱스는 쿼리를 수행할 때 데이터를 더 빠르게 검색하기 위해 생성됨
- 벡터 인덱스를 생성하는 데 사용할 수 있는 방법
  - 무작위 투영(Random Projection)
  - 제품 정량화(Product Quantization)
  - 로컬리티에 민감한 해싱(Locality-sensitive Hashing)
  - 계층적 탐색 가능한 작은 세계(Hierarchical Navigable Small World) 등

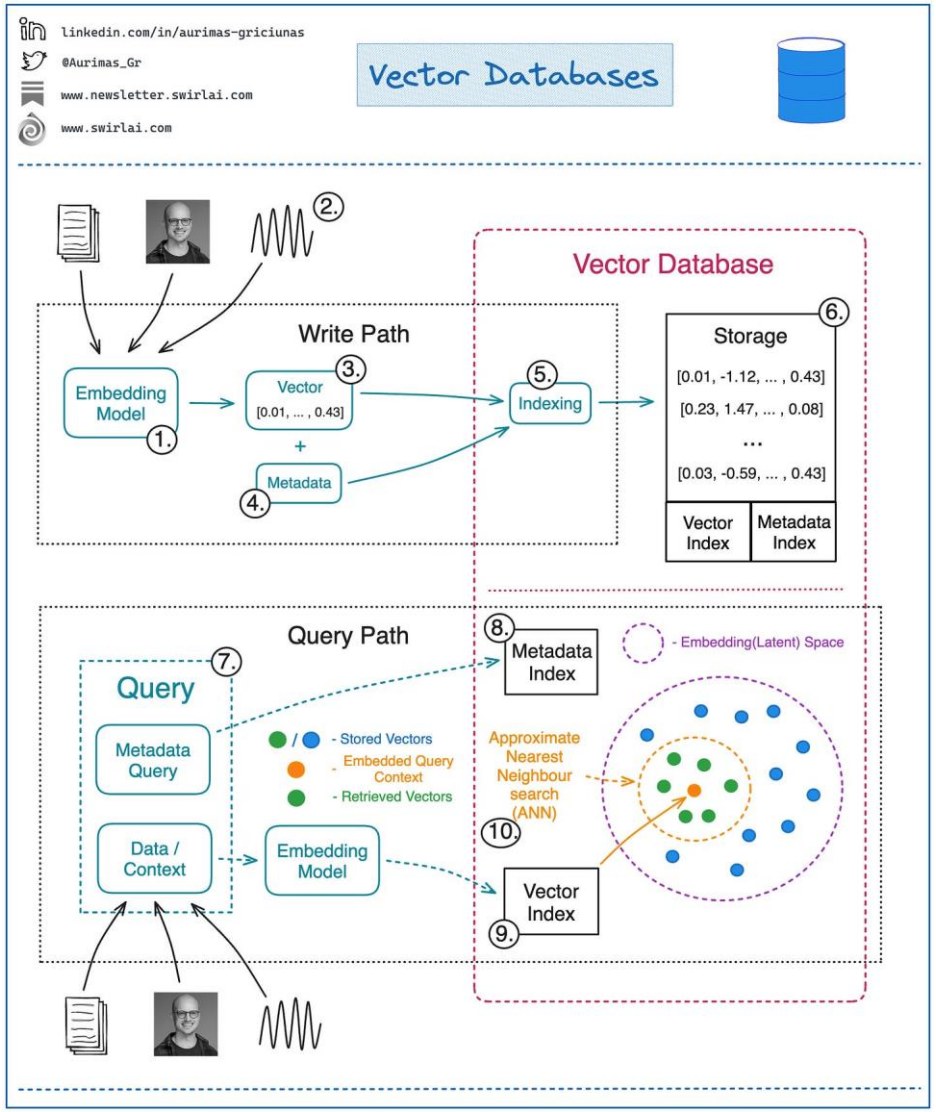
6. 벡터 데이터는 벡터 임베딩에 대한 인덱스 및 임베디드 오브젝트에 연결된 메타데이터와 함께 저장됨



벡터 데이터베이스의 작동 방식

- **데이터 읽기**

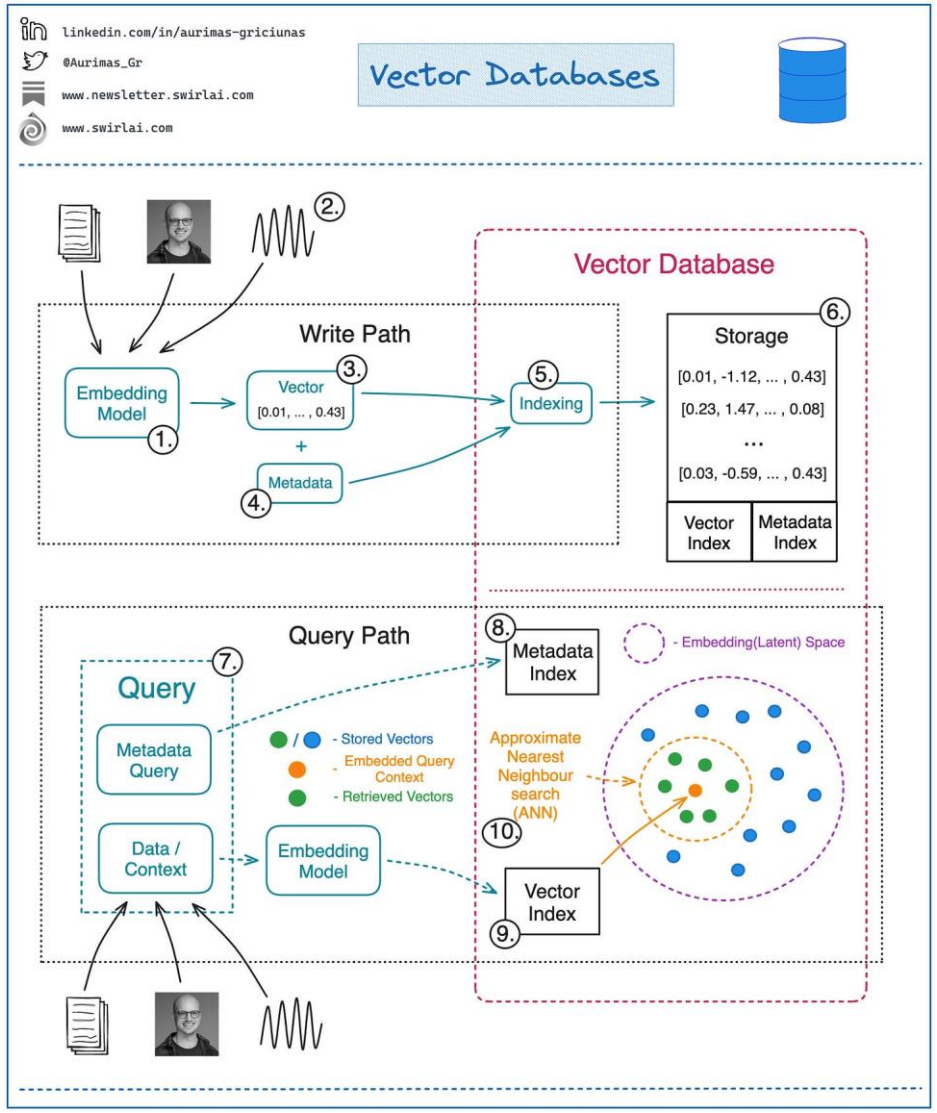
  7. 벡터 데이터베이스에 대해 실행할 쿼리 구성(2부분 구성)
     - ANN 검색에 사용될 데이터
       (예: 유사한 이미지를 찾고자 하는 이미지)
     - 미리 알려진 특정 특성을 가진 벡터를 제외하기 위한 메타데이터 쿼리
       (예: 비슷한 아파트 이미지를 찾고 있다면 이 정보가 메타데이터에 포함되어 있다면 특정 위치에 있는 아파트를 제외할 수 있음)



벡터 데이터베이스의 작동 방식

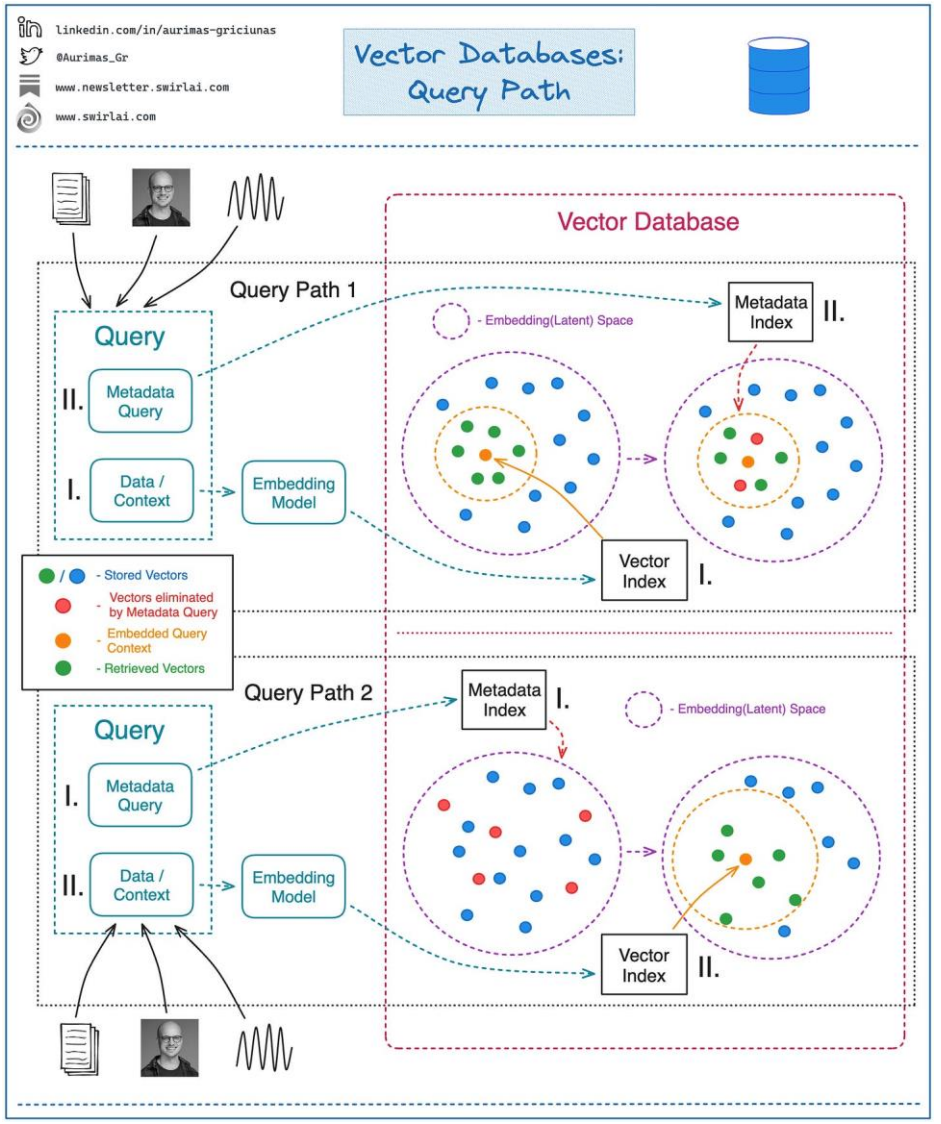
8. 메타데이터 인덱스에 대해 메타데이터 쿼리를 실행
   (이 작업은 ANN 검색 절차 전이나 후에 수행할 수 있음)
9. 데이터의 컨텍스트에 관해서는,
   벡터 데이터베이스에 데이터를 쓸 때 사용한 것과
   동일한 모델을 사용하여 데이터를 잠재 공간에 임베드 함
10. 사용된 인덱싱 방법에 따라,
    데이터베이스는 쿼리 벡터도 인덱싱해야 할 수도 있음.
    인덱싱이 완료되면 ANN 검색 절차가 적용되고
    벡터 임베딩 세트가 검색됨

ANN 검색에 널리 사용되는 유사도 측정값: 코사인 유사도(Cosine Similarity),
유클리드 거리(Euclidean Distance), 내적 곱(Dot Product) 등



벡터 데이터베이스의 작동 방식

- **가능한 쿼리 경로**
  - **메타데이터 쿼리가 포함된 경우,**
    두 가지의 가능한 쿼리 경로가 있음
    - **경로 1: 먼저 ANN 검색을 적용하고, 검색된 결과에 대해 메타데이터 쿼리를 실행함**
      - 메타데이터 쿼리를 먼저 적용할 때 보다 ANN 검색과 관련된 일부 컨텍스트가 손실될 수 있음
      - 그러나 검색 공간을 크게 줄일 수 있으며, 이를 통해 전반적인 쿼리 성능을 향상시킬 수 있음
    - **경로 2: 메타데이터 쿼리를 먼저 실행한 다음, 필터링된 결과에 대해 ANN 검색을 적용함**



벡터 데이터베이스의 활용 예시

- 자연어 처리
  - LLM에 컨텍스트를 제공하는 인기 있는 기법의 예
    - 서브스택에 있는 문서에 대한 질문에만 답변할 수 있는 챗봇을 만들고 싶다고 가정하면
      - 모든 문서를 벡터에 임베드하고 벡터 데이터베이스에 저장할 수 있음
      - 질문을 할 때 동일한 잠재 공간에 질문을 임베드하고
      - 임베드된 질문을 사용하여 쿼리를 실행하여
      - 벡터 데이터베이스에서 가장 관련성이 높은 텍스트를 검색
      - 그런 다음 검색된 텍스트 조각을 LLM으로 전송하여 답변을 구성

벡터 데이터베이스의 활용 예시

- **컴퓨터 비전**
  - **비슷하거나 다른 각도에서 찍은 사진에서 동일한 물체를 식별하기**
    - 에어비앤비에 등록된 방이 부킹닷컴에 등록된 방과 동일한지 확인하려고 한다고 가정하면
      - BookingCom 벡터 데이터베이스 인덱스에서 ANN 검색을 통해 Airbnb 사진을 실행하여
      - 가장 유사한 특징이 있는 객실을 식별할 수 있음

벡터 데이터베이스의 활용 예시

- 추천 시스템
  - 주로 2개의 단계로 구성됨
  - 후보 검색 단계
    - 쿼리가 주어지면
    - 순위를 매기기 위한 소수의 후보를 검색
      - 주로 계산 비용이 많이 드는 모델을 사용하게 됨
      - 이 단계에서 기존의 비용이 큰 모델 대신
        쿼리 벡터와 유사한 벡터를 대량으로 효율적으로 검색할 수 있는 벡터 데이터베이스를 사용할 수 있음
  - 순위 결정 단계
    - 이전 단계에서 (무거운 모델을 사용하여) 검색된 후보를 다시 점수를 매겨 순위를 결정함

벡터 데이터베이스의 활용 예시

- 인기 있는 벡터 데이터베이스
  - Qdrant, Pinecone, Weviate, Milvus, Faiss 등

- 벡터 데이터베이스는 아니지만 점점 더 많은 데이터베이스 제공업체가 ANN 검색기능을 지원하기 시작함
  - Redis, Cassandra 등

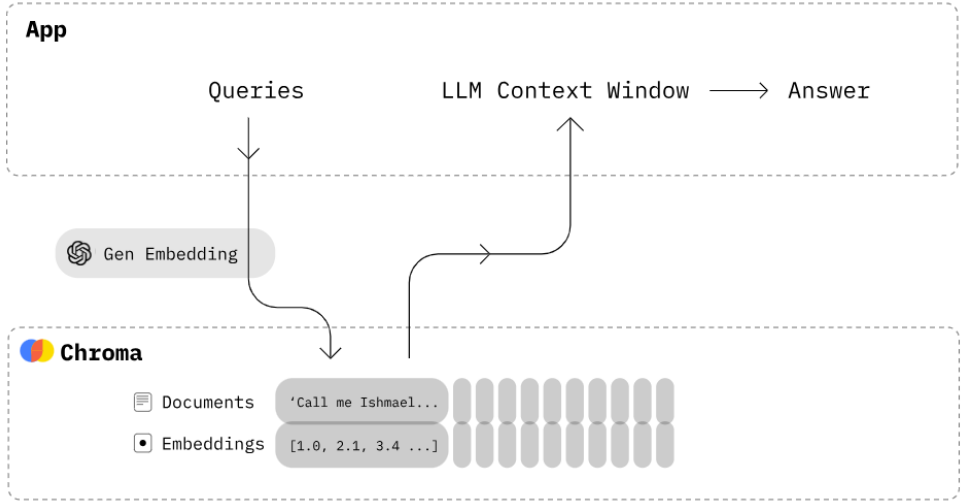
ANN (Approximate Nearest Neighbor, 근사 최근접 이웃) 검색
- 주어진 질의점에 가장 가까운(유사한) 점을 찾는 최적화 문제
- 공간 (M)의 점 집합 (S)와 질의 점 (q ₩in M)이 주어졌을 때 (q)에 가장 가까운 (S)의 점을 찾는 것
- 일반적으로 유사도 함수를 사용하여 가까움을 표현하며, 작은 함수 값은 객체 간 유사성이 높음을 나타냄

오픈소스 벡터 데이터베이스의 활용

Chroma 개요

- **Chroma**
  - **오픈 소스 벡터 데이터베이스(Vector Database)**
  - **벡터 형태의 데이터를 저장하고 유사성을 기반으로 검색하는 데에 특화됨**
    - **특히 텍스트 데이터에 최적화된 벡터 데이터베이스**
    - **텍스트 데이터를 벡터로 효율적으로 변환하고 저장하며, 벡터 공간에서의 유사성 검색 및 쿼리를 가능하게 함**

Chroma



Chroma 개요

- 대규모의 고차원 벡터 데이터를 효율적으로 저장하고 빠르게 검색할 수 있는 기능을 제공
  - 텍스트 데이터를 임베딩하여 고차원의 벡터 공간에 저장
  - 임베딩 과정은 텍스트의 의미와 구조를 보존하면서 벡터로 변환하는 작업을 포함
  - 저장된 벡터 데이터는 고속 및 고정된 시간에 쿼리할 수 있도록 색인화

Chroma 개요

- **Chroma의 주요 특징**
  - **효율적인 저장 및 검색**
    - 대용량의 텍스트 데이터를 효율적으로 저장
    - FAISS 라이브러리를 기반으로 빠른 벡터 검색 기능을 제공
    - 방대한 양의 데이터와 복잡한 쿼리에도 빠른 응답 속도 보장
    - 데이터 압축, 분산 저장 등 다양한 기능을 통해 데이터를 효율적으로 관리
  - **유사성 검색**
    - 임베딩된 텍스트의 유사성을 기반으로 검색 수행
    - 유사한 의미를 가진 문장이나 단어를 쉽게 찾을 수 있음
    - 벡터 거리, 코사인 유사도, Jaccard 유사도 등 다양한 검색 유형을 지원

Chroma 개요

- **다양한 응용**
  - 벡터 검색, 유사성 검색, 머신 러닝/인공지능 분야, 자연어 처리, 정보 검색, 추천 시스템 등 다양한 분야에서 텍스트 데이터를 다루는데 사용 가능

- **확장성**
  - 대규모 텍스트 데이터셋을 다루기 위해 설계됨
  - 필요에 따라 수평 및 수직으로 확장 가능하며 분산 시스템에서도 사용할 수 있음

- **사용자 친화적인 인터페이스**
  - Python, Java, C++ 등 다양한 프로그래밍 언어를 지원하며, 사용자 친화적인 인터페이스를 제공

Chroma 개요

- **Chroma의 장점**
  - **간편한 설치 및 사용:** Python과 JavaScript를 지원하며, 간단한 명령어로 설치할 수 있음
  - **빠르고 효율적인 검색:** 다른 벡터 데이터베이스에 비해 빠르고 효율적인 검색 성능을 제공함
  - **다양한 저장 방식 지원:** HTTP 방식, 디스크 저장 방식, 인메모리 방식을 선택할 수 있어 사용자의 요구에 맞게 선택할 수 있음
  - **다양한 기능:** 다양한 검색 유형, 데이터 관리 기능 등을 제공함
  - **확장성:** 분산 처리를 지원하여 대규모 데이터도 효율적으로 처리할 수 있음
  - **사용 편의성:** 사용자 친화적인 인터페이스를 제공
  - **개발 편의성:** 다양한 프로그래밍 언어 지원, LLM을 위한 LangChain 및 LlamaIndex 지원
  - **오픈 소스:** 오픈 소스 소프트웨어이며, 무료로 사용할 수 있음

Chroma 개요

- **Chroma의 단점**
  - **비교적 새로운 기술:** 다른 벡터 데이터베이스에 비해 비교적 새로운 기술
  - **커뮤니티 규모:** 아직 다른 벡터 데이터베이스에 비해 커뮤니티 규모가 작음

Chroma의 장점과 단점은 (다른 벡터 데이터베이스와 비교했을 때)
- 특히 데이터의 신속한 검색과 관리가 필요한 애플리케이션에 적합
- 사용의 편리함과 빠른 성능은 Chroma를 매력적인 선택으로 만드는 요소가 됨

Chroma 개요

- **타 벡터 데이터베이스와의 비교**

데이터 베이스	오픈소스 여부	자체 호스팅	클라우드 관리	벡터 전용	개발자 경험	커뮤니티	초당 쿼리 수	지연 시간 (ms)	인덱스 종류	하이브리드 검색	디스크 인덱스 지원	RBAC	데이터 샤딩	무료 호스팅 티어
Pinecone		✓	✓	✓		8k☆ github, 4k slack	150	1	Multiple	✓	✓	✓	Static	
Weaviate	✓	✓	✓	✓		23k☆ github, 4k slack	791	2	HNSW	✓	✓	✓	Dynamic	✓
Milvus	✓	✓	✓	✓		13k☆ github, 3k discord	2406	1	Multiple	✓	✓	✓	Dynamic	✓
Qdrant	✓	✓	✓	✓		9k☆ github, 6k discord	326	4	HNSW	✓	✓	✓	Static	✓
Chroma				✓		23k slack	?	?	HNSW					
Elasticsearch	✓	✓	✓			6k☆ github	700-100	8	HNSW/ IVFFlat	✓	✓	✓	Static	

Chroma 개요

- **Chroma 활용 사례**
  - **콘텐츠 검색:** 이미지, 음성, 텍스트 등 다양한 형태의 콘텐츠 검색에 활용
  - **개인 맞춤 추천:** 사용자의 과거 행동과 선호도를 기반으로 개인 맞춤 추천 시스템을 구축하는 데 활용
  - **챗봇:** 챗봇의 자연어 처리 및 대화 능력 향상에 활용
  - **사기 감지:** 금융 거래 데이터에서 사기성 거래를 감지하는 데 활용
  - **의료 영상 분석:** 의료 영상 데이터에서 질병을 진단하거나 치료 계획을 수립하는 데 활용

Chroma 사용하기

- **Chroma의 사용 단계**

1. **설치**
   - 명령어:

```
$ pip install chromadb
```

2. **Chroma 클라이언트의 생성 및 연결**
   - Chroma를 사용하기 위해 클라이언트를 생성
     - EphmeralClient 메모리에 저장하는 클라이언트
     - HttpClient 네트워크를 통해 접속하는 클라이언트

   - 소스 코드:

```
import chromadb

chroma_client = chromadb.Client()     # 데이터를 파일에 저장할 수 있음
```

3. 컬렉션 생성

- 컬렉션: 임베딩, 문서 및 추가 메타데이터를 저장하는 곳
- 소스 코드:

```
collection = chroma_client.create_collection(name="my_collection")
```

컬렉션 생성과 다음 과정의 사이에 필요에 따라 아래와 같은 과정이 올 수도 있음
- Pandas를 사용하여 데이터 불러오기
- SentenceTransformer와 같은 모델을 사용하여 벡터 생성하기

4. 데이터 추가하기

- Chroma는 텍스트를 저장하고 임베딩 및 인덱싱을 자동으로 처리함(벡터 데이터가 생성됨)
- 임베딩 모델을 사용자 지정할 수도 있음
- 생성한 벡터와 메타데이터를 ChromaDB에 추가
- 소스 코드:

```
collection.add(
    documents = [
        “This is a document about pineapple”,
        “This is a document about oranges”
    ],
    ids = [“id1”, “id2”]
)
```

5. 검색

- 원하는 쿼리 벡터를 생성하여 ChromaDB에서 유사한 벡터를 검색
- 쿼리 텍스트 목록을 사용하여 컬렉션을 쿼리할 수 있음
- Chroma는 가장 유사한 결과를 반환함
- 소스 코드:

```
results = collection.query(
    query_texts=["This is a query document about hawaii"],    # Chroma가 임베드 수행
    n_results=2      # 반환할 결과 수
)
print(results)
```

7. 직접 해 보기

```
import chromadb

chroma_client = chromadb.Client()

collection = chroma_client.get_or_create_collection(name="my_collection")


collection.upsert(
    documents=[
        "This is a document about pineapple",
        "This is a document about oranges"
    ],
    ids=["id1", "id2"]
)

results = collection.query(
    query_texts=["This is a query document about florida"],
    n_results=2
)
print(results)
```

매번 새 컬렉션을 생성하지 않으려면 `create_collection`을 `get_or_create_collection`으로 전환

매번 동일한 문서를 추가하지 않으려면 `add`를 `upsert`로 전환

Chroma가 이 내용을 삽입함

반환할 결과 수