LangChain을 활용한 SW 개발

챗봇과 검색 증강 생성 기술

충북대학교 의과대학
박 승

충북대학교
CHUNGBUK NATIONAL UNIVERSITY

학습 목표

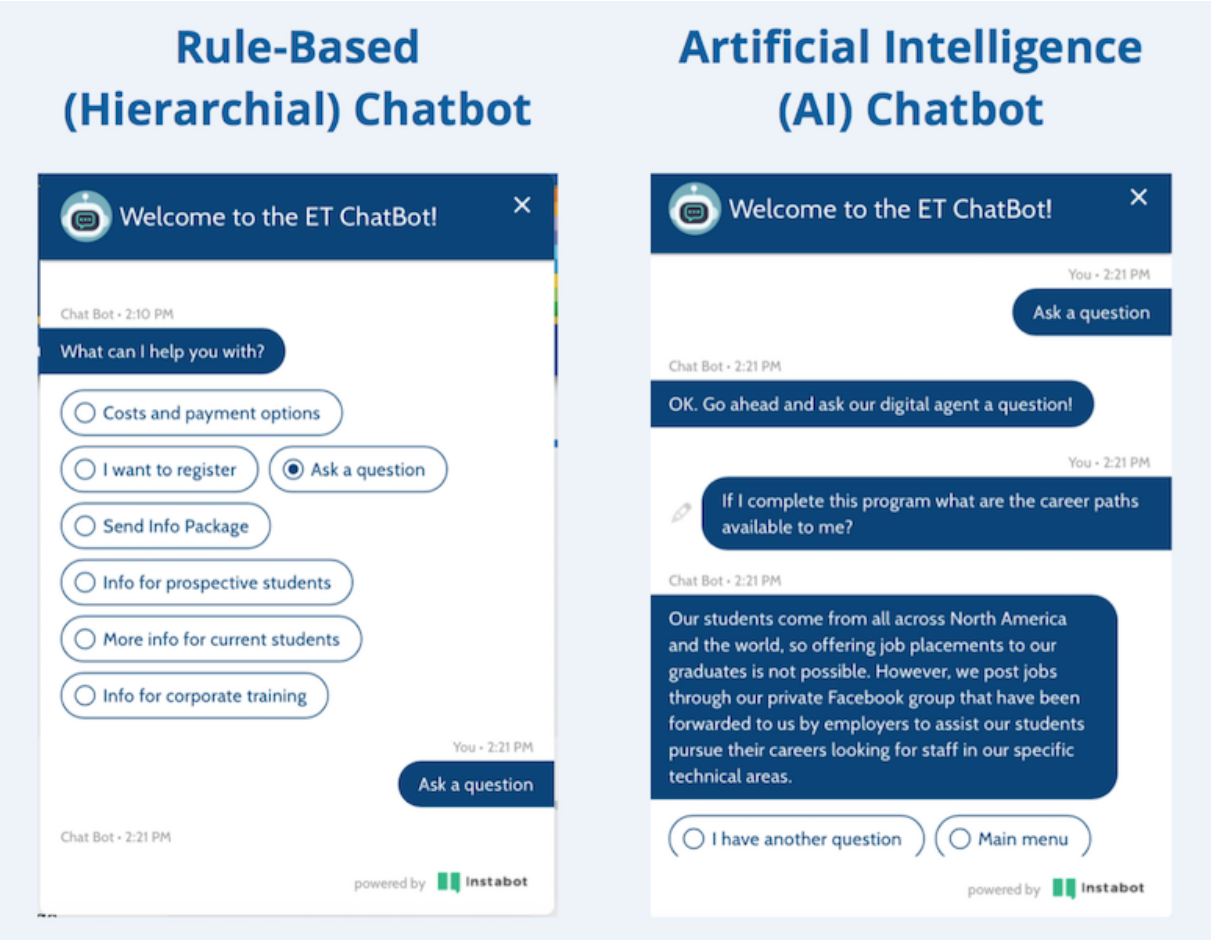
1. 규칙 기반 챗봇과 AI 기반 챗봇의 차이점과 활용 사례를 설명할 수 있다.
2. 검색 증강 생성의 개념과 작동 원리를 이해하고 설명할 수 있다.
3. 벡터 저장소와 벡터 인덱싱의 원리 및 주요 기법들을 이해하고 활용할 수 있다.
4. LangChain과 Chroma를 활용하여 벡터 저장소를 구현할 수 있다.

CHUNGBUK
NATIONAL UNIVERSITY

ChatBot

텍스트 또는 음성을 통해 사람과 상호작용을 하는 서비스

- 규칙 기반(rule-based) 챗봇
  - 미리 정의된 선택지를 제공하여 사용자가 그 중 하나를 선택
  - 제한된 답변을 사전 설정된 방식으로 제공
  - 대화의 흐름이 고정적이고 예측 가능
- AI 기반 챗봇
  - 사용자의 자유로운 질문을 받아 자연어 처리를 통해 응답
  - 입력된 질문을 분석해 보다 유연하고 상황에 맞는 답변 생성
  - 대화의 흐름이 사용자 입력에 따라 유동적으로 변화



ChatBot

챗봇 서비스 활용 사례

- 관리정보 검색: 날씨 업데이트, 뉴스 기사, 주식 가격 정보 제공
- 가상 비서: 알림 설정, 메시지 전달, 일정 관리
- 언어 학습: 상호작용 대화와 연습을 통한 학습 지원
- 정신 건강 지원: 정신적 지원 및 상담 제공
- 교육: 튜터 역할 수행, 학습 평가, 질문에 대한 답변 제공
- 인사 및 채용: 후보자 스크리닝, 면접 일정 예약, 채용 정보 제공
- 엔터테인먼트: 퀴즈, 스토리텔링 등 대화형 게임 제공
- 법률: 법률 질문 응답 및 관련 서류 작성 지원
- 의학: 건강 정보 제공, 진단 지원 및 전문가 추천

검색 증강 생성 (Retrieval Augmented Generation)

대규모 언어 모델의 한계점

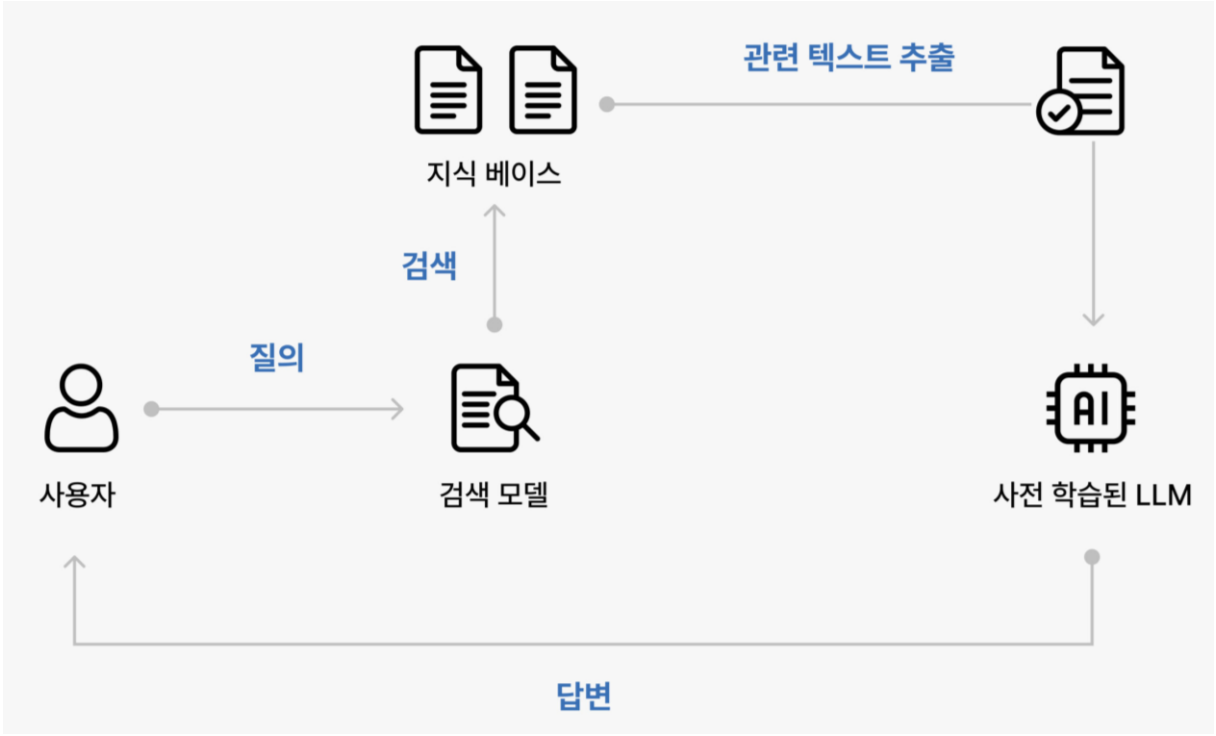
- 환각 현상(hallucination), 최신 지식 업데이트 한계, 특정 분야 전문 지식 부족

검색 증강 생성 (RAG)

- LLM과 외부 지식베이스의 결합을 통한 정확성 향상 기법
- 문서 검색을 통해 신뢰도 높은 정보 제공 방법론

RAG의 기본 원리

- 사용자 질문(query) 입력
- 지식베이스(knowledge base)에서 관련 문서 검색
- 검색된 문서에서 관련 텍스트 추출
- 추출된 정보를 LLM에 입력하여 답변 생성



검색 증강 생성 (Retrieval Augmented Generation)

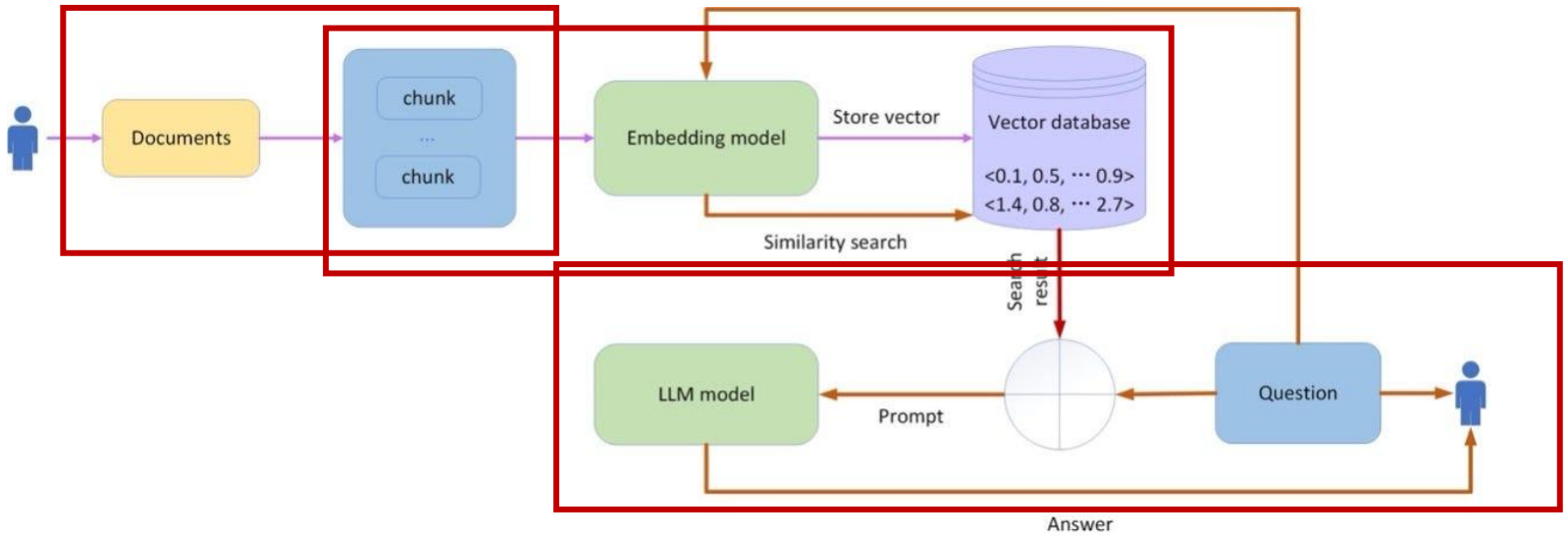
대규모 언어 모델과 검색 증강 생성 모델의 차이점

항목	Original LLM	LLM with fine-tuning	RAG
외부 지식	필요 없음	필요 없음	필요 (외부 지식 DB에서 정보 검색)
환각 최소화	어려움 (환각 발생 가능성이 높음)	어려움 (잘못된 정보를 생성할 수 있음)	가능 (정확한 정보 제공)
해석 가능성	낮음	낮음	높음 (검색된 정보를 근거로 할 수 있음)
기술 난이도	낮음	높음 (데이터 과학 및 최적화 기술 필요)	낮음
비용 및 리소스	비용 효율적 (추가적인 학습 과정 없음)	고비용 (데이터 수집 및 재훈련 필요)	비용 효율적 (재훈련 없이 정보 검색 활용)

검색 증강 생성 (Retrieval Augmented Generation)

검색 증강 생성의 기본 구조

- 1. 데이터 전처리 및 문서 분할 (document chunking)
  - 사용자가 제공한 원본 문서를 일정 크기의 작은 단위(chunk)로 분할
- 2. 벡터화 및 임베딩 생성 (embedding generation)
  - 분할된 각 chunk는 임베딩 모델을 통해 의미적 표현인 벡터(vector)로 변환되어 벡터 데이터베이스에 저장됨
- 3. 질문 벡터화 및 유사성 검색 (query vectorization and similarity search)
  - 사용자의 질문은 동일한 임베딩 모델을 사용하여 벡터화된 후, 벡터 데이터베이스 내의 문서 벡터들과 유사성 비교 수행
- 4. 프롬프트 기반 답변 생성 (prompt-based answer generation)
  - 유사성이 높다고 판단된 정보들은 프롬프트로 구성되어 LLM에 전달된 후, 맥락에 적합한 최종적인 답변이 생성됨



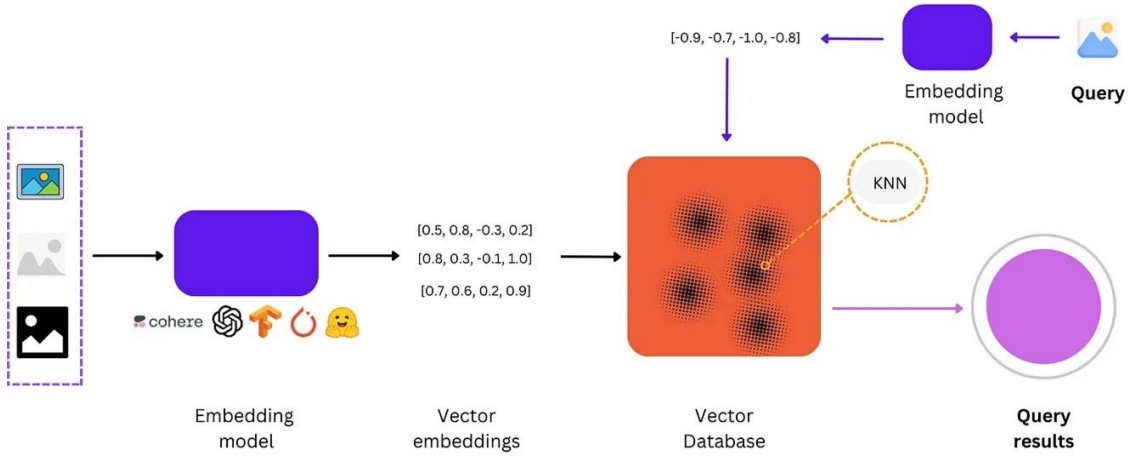
검색 증강 생성 (Retrieval Augmented Generation)

벡터 저장소 (vector storage)

- 벡터화된 데이터를 저장하고 검색하는 데 사용되는 구조
- RAG 모델이 생성한 벡터를 저장하고, 유사도를 계산해 관련 정보를 효율적으로 검색

벡터 저장소의 동작 원리

- 데이터 임베딩
  - 입력된 텍스트를 벡터로 변환하여 고차원 공간에서 텍스트 간의 유사성 표현
- 데이터 저장 및 관리
  - 생성된 벡터는 벡터 저장소에 저장하여 데이터베이스 생성
  - 효율적인 검색을 위해 인덱싱과 클러스터링 같은 기법을 적용하여 벡터 관리
- 검색 및 유사성 비교
  - 입력된 쿼리 문장을 벡터로 변환하고 저장소에서 유사한 벡터들을 검색
  - 가장 유사한 벡터들에 해당하는 데이터를 반환



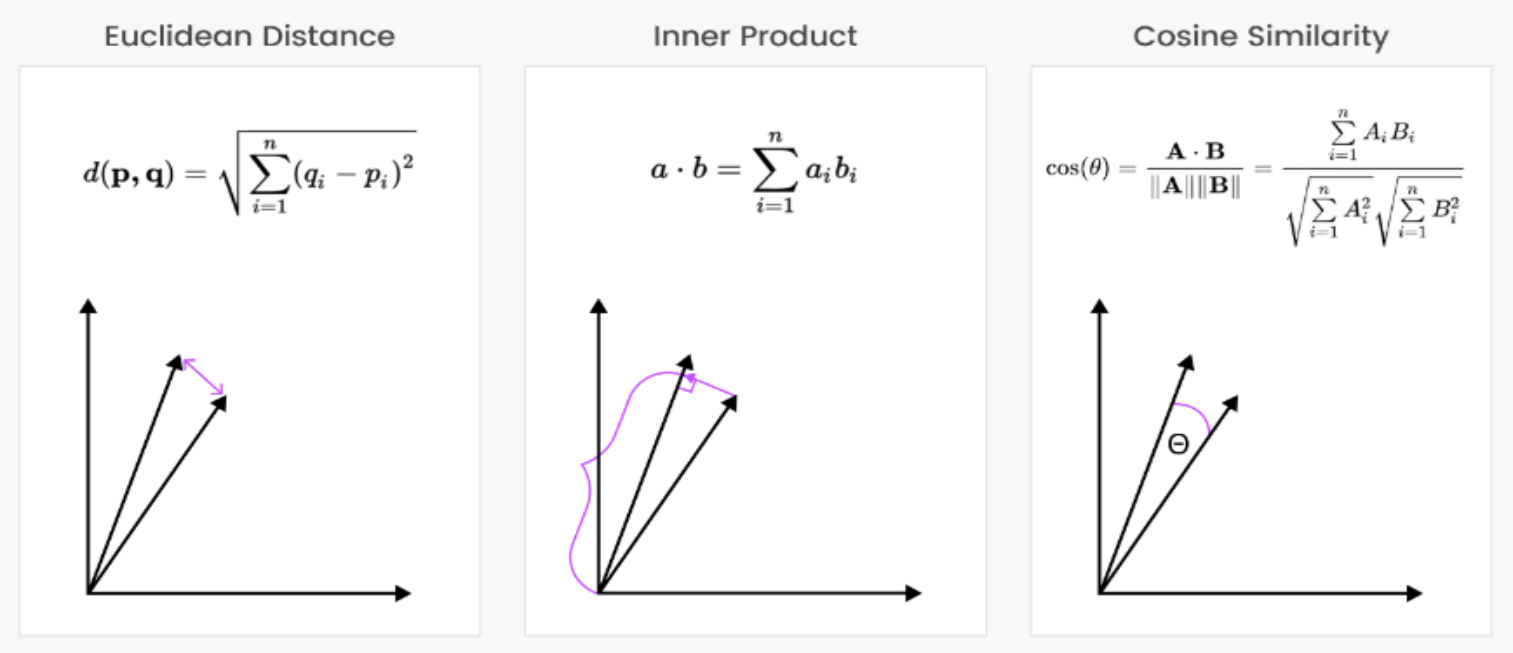
검색 증강 생성 (Retrieval Augmented Generation)

벡터 인덱싱 (vector indexing)

- 벡터 저장소에서 벡터를 효율적으로 검색하고 유사성을 비교하기 위해 사용하는 기법
  - 대규모 벡터 데이터를 검색할 때 모든 벡터를 하나씩 비교하는 것은 매우 비효율적임
  - 벡터 인덱싱은 벡터를 적절히 정렬하고 구조화하여 빠른 검색을 가능하게 함

벡터 인덱싱의 유사성 측정 방법

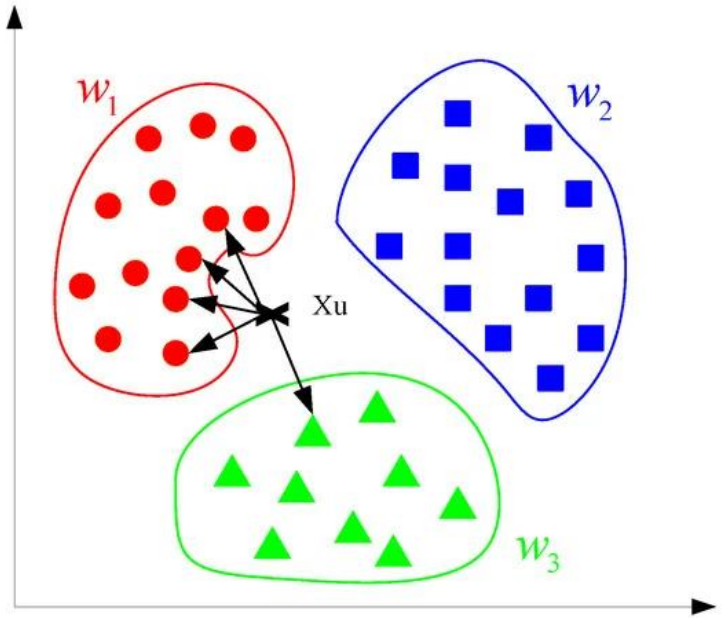
- 1. 유클리드 거리 (Euclidean distance)
  - 벡터 간의 직선 거리를 계산하여 유사성 측정
- 2. 내적 (dot product)
  - 두 벡터의 내적을 통해 유사성 측정
- 3. 코사인 유사도 (cosine similarity)
  - 두 벡터 간의 각도를 기반으로 유사성 측정



검색 증강 생성 (Retrieval Augmented Generation)

주요 벡터 인덱싱 기법

- 1. k-NN (k-Nearest Neighbors)
  - 모든 벡터와의 거리를 계산하여 가장 유사한 벡터들을 반환
- 2. FAISS (Facebook AI Similarity Search)
  - Facebook에서 개발한 벡터 검색 및 인덱싱 라이브러리
  - Flat Index: 모든 벡터를 단순한 배열로 저장하고, 벡터 간 거리를 모두 계산하는 방식
  - Inverted File Index: 벡터를 여러 개의 클러스터로 나누고, 각 클러스터 내에서만 유사성 비교
- 3. LSH (Locality-Sensitive Hashing)
  - 유사한 벡터를 동일한 해시 값으로 변환하여, 고차원 데이터에서 유사한 벡터 검색
- 4. HNSW (Hierarchical Navigable Small World)
  - 벡터를 노드로 하는 그래프를 만들고, 노드 간의 거리를 기반으로 탐색
- 5. Product Quantization (PQ)
  - 벡터를 더 작은 서브벡터로 나누고, 각 서브벡터를 개별적으로 양자화(quantization)하여 저장
  - 벡터를 저장하는 데 필요한 공간을 크게 줄일 수 있고, 검색 시에도 서브벡터 간의 유사성을 빠르게 계산할 수 있음



k-Nearest Neighbors

검색 증강 생성 (Retrieval Augmented Generation)

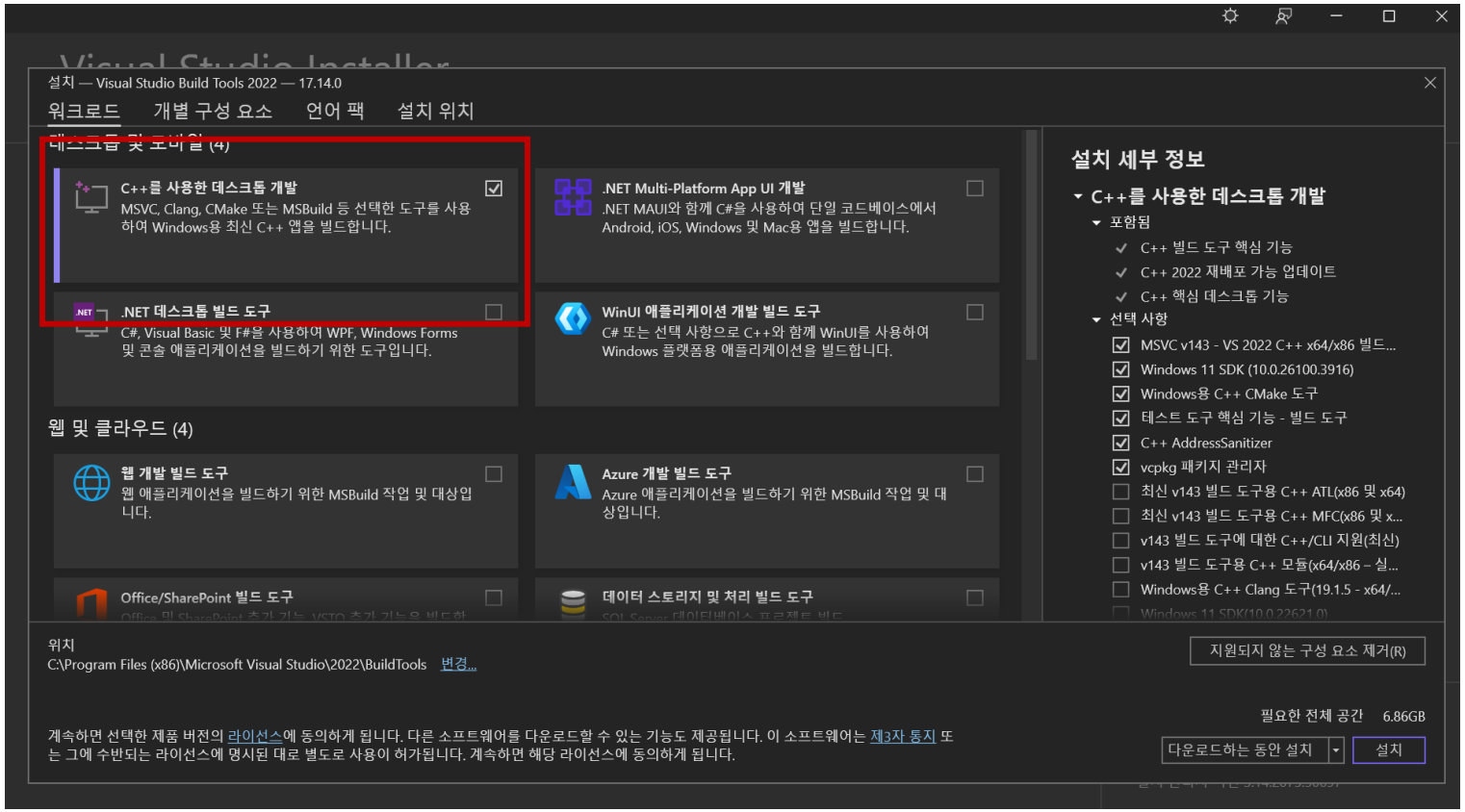
상용 벡터 저장소

이름	설명	비즈니스 모델	인덱싱
Chroma	상업용 오픈 소스형 임베딩 스토어	부분 공개 SaaS	HNSW
Qdrant	관리형/셀프 호스팅 백터 검색 엔진 및 필터링 지원	부분 공개 SaaS	HNSW
Milvus	확장 가능한 유사성 검색을 위해 구축된 벡터 데이터베이스	부분 공개 SaaS	IVF, HNSW, PQ 등
Weaviate	객체와 백터를 모두 지원하는 클라우드 네이티브 벡터 데이터베이스	공개 SaaS	CRUD를 지원하는 맞춤형 HNSW
Pinecone	AI 모델의 임베딩을 사용한 빠르고 확장 가능한 어플리케이션	공개 SaaS	Faiss 기반
Vespa	벡터 검색, 어휘 검색 등을 지원하는 상업용 오픈 소스 벡터 데이터베이스	공개 SaaS	HNSW, BM25
Marqo	클라우드 네이티브 상업용 오픈 소스 검색 및 분석 엔진	공개 SaaS	HNSW

검색 증강 생성 (Retrieval Augmented Generation)

실습 1) Chroma를 활용한 벡터 저장소 만들기

- 1. Microsoft C++ Build Tools
  - 다운로드 링크: https://drive.google.com/drive/folders/10hspG3q7X5H3JMSlQAfDq0kEYyY6pw4b?usp=sharing



검색 증강 생성 (Retrieval Augmented Generation)

실습 1) Chroma를 활용한 벡터 저장소 만들기

- 2. 라이브러리 업데이트 및 로드

```
# 라이브러리 충돌을 방지하기 위한 업데이트 수행
!pip uninstall -y langchain langchain-core chromadb langsmith pydantic pydantic-settings
!pip install langchain==0.3.25 langchain-core==0.3.60 chromadb==1.0.9
!pip install langsmith==0.3.42 pydantic==2.11.4 pydantic-settings==2.9.1
!pip install arxiv
```

```
# 라이브러리 연동
from langchain_openai import OpenAIEmbeddings
from langchain.vectorstores import Chroma
from langchain.document_loaders import ArxivLoader
from langchain.text_splitter import CharacterTextSplitter


# OpenAI API 키 설정
api_key = "sk-proj-zcYjewY8DYjQ4gjIKzBeASFA1ru_McHtN4kA50N-QjVLGEXQx9pivZHXI7ste6bOI7yOl
```

검색 증강 생성 (Retrieval Augmented Generation)

실습 1) Chroma를 활용한 벡터 저장소 만들기

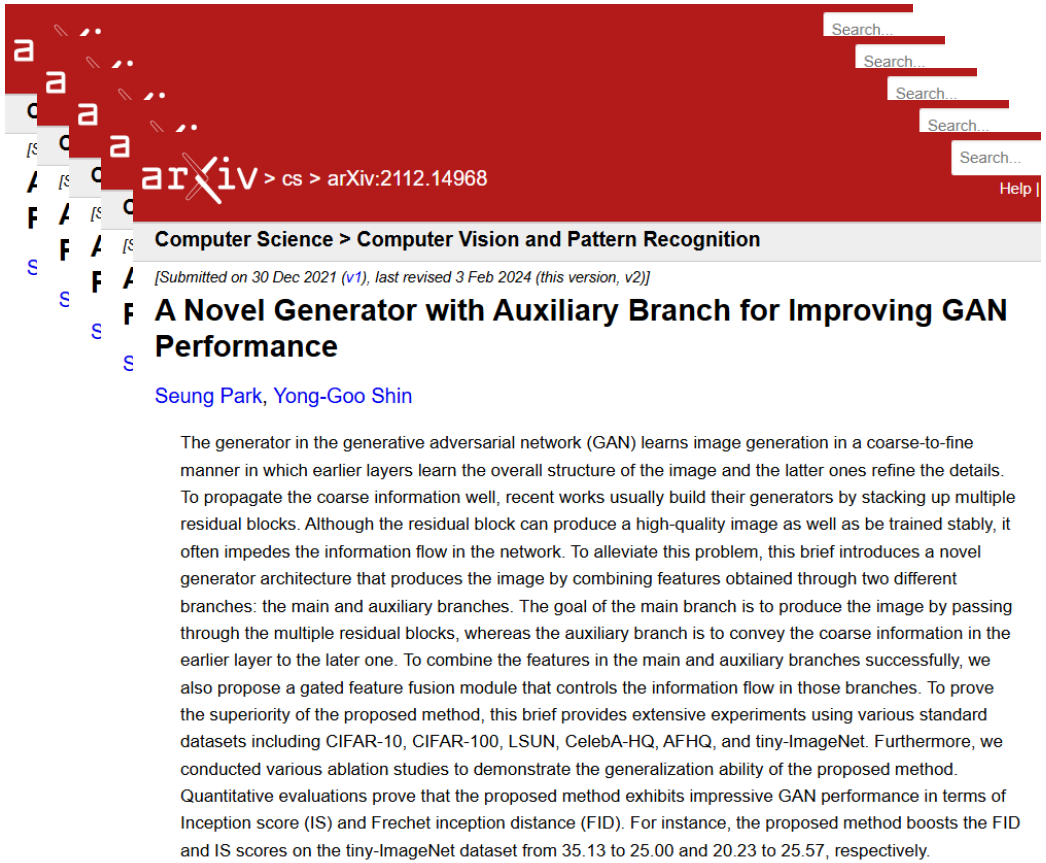
- 3. arXiv 논문 다운로드를 통한 벡터 저장소 구축

```
# 여러 논문 ID를 사용하여 논문을 불러오기 위한 ArxivLoader 설정
queries = ["2407.05527", "2112.14968", "2210.09638", "2201.11351", "2111.15171"]
documents = []

for query in queries:
    loader = ArxivLoader(query=query)
    documents.extend(loader.load())

text_splitter = CharacterTextSplitter(chunk_size=1000, chunk_overlap=0)
docs = text_splitter.split_documents(documents)

# 벡터 저장소에 문서 저장
vector_store = Chroma.from_documents(
    documents=docs,
    embedding=OpenAIEmbeddings(openai_api_key=api_key)
)
```



검색 증강 생성 (Retrieval Augmented Generation)

실습 1) Chroma를 활용한 벡터 저장소 만들기

- 4. 입력과 가장 유사한 문서 탐색 결과

```
# 새로운 쿼리 입력
new_query = "Generator in GAN"

# 유사한 벡터 데이터 3개 검색
similar_docs = vector_store.similarity_search(new_query, k=3)

# 결과 출력
for idx, doc in enumerate(similar_docs):
    print(f"유사한 문서 {idx+1}:\n{doc.page_content}\n")
```

```
유사한 문서 1:
IEEE TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS (ACCEPT, TO BE PUBLISHED)
1
A Novel Generator with Auxiliary Branch
for Improving GAN Performance
Seung Park and Yong-Goo Shin, Member, IEEE
Abstract—The generator in the generative adversarial network
(GAN) learns image generation in a coarse-to-fine manner in
which earlier layers learn the overall structure of the image
and the latter ones refine the details. To propagate the coarse
information well, recent works usually build their generators by
stacking up multiple residual blocks. Although the residual block
can produce a high-quality image as well as be trained stably, it
often impedes the information flow in the network. To alleviate
```

감사합니다

Q&A

충북대학교
CHUNGBUK NATIONAL UNIVERSITY