

# LangChain RAG

검색 증강 생성(Retrieval-Augmented Generation) 아키텍처



# RAG 기술의 필요성과 핵심 개념

## RAG 시스템 개요

- ✓ RAG는 대규모 언어 모델(LLM)의 한계를 보완하는 혁신적 기술
- ✓ 외부 지식 소스(예: 데이터베이스, 문서 저장소)로 부터 관련된 정보를 검색하여 더 정확하고 신뢰할 수 있는 답변 생성
- ✓ 핵심은 LLM이 참조할 수 있는 고품질의 지식 기반 구축



## LLM의 한계점

### 지식 단절 (Knowledge Cut-off)

LLM은 학습 시점 이후의 최신 정보에 접근할 수 없어, 최신 사실과 데이터에 대한 답변 생성에 어려움을 겪습니다.

### 환각 (Hallucination)

학습된 지식을 기반으로 하지만, 정확하지 않은 정보를 생성하거나 특정 주제에 대한 깊이 있는 지식 없이 불완전한 답변을 제공할 수 있습니다.

# RAG 핵심 아키텍처 개요

LangChain RAG는 대규모 언어 모델(LLM)의 응답 품질을 향상시키기 위해 외부 지식을 통합하는 프레임워크로, 크게 두 가지 핵심 단계로 구성됩니다.



## 인덱싱 단계

### 1 로드 (Load)

다양한 소스에서 원본 데이터를 불러와 Document 객체로 변환

### 2 분할 (Split)

문서를 LLM이 처리할 수 있는 척크로 나눔

### 3 저장 (Store)

척크를 임베딩 모델로 변환하여 Vector Store에 저장

## 검색 및 생성 단계

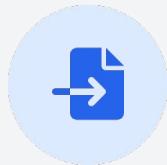
### 1 검색 (Retrieve)

쿼리 벡터를 Vector Store와 비교하여 관련 문서 척크 검색

### 2 생성 (Generate)

검색된 문서와 사용자 쿼리를 결합하여 LLM이 답변 생성

# 인덱싱 단계: 데이터 준비 프로세스



로드



분할



저장



## 1. 로드 (Load)

- ✓ 다양한 소스(웹페이지, 문서, 데이터베이스 등)에서 원본 데이터를 불러옴
- ✓ 데이터를 Document 객체로 변환
- ✓ LangChain은 Document Loaders를 통해 이 작업을 수행



## 2. 분할 (Split)

- ✓ 청크는 LLM이 처리할 수 있는 크기로 관리하기 쉬운 단위로 분할
- ✓ Text Splitters가 큰 문서를 작은 청크로 나눔



## 3. 저장 (Store)

- ✓ 분할된 텍스트 청크를 임베딩(embedding) 모델을 통해 벡터 표현으로 변환
- ✓ 벡터 임베딩은 Vector Store에 저장됨
- ✓ 나중에 사용자 쿼리와 유사한 문서를 효율적으로 검색하는 데 사용

ⓘ 인덱싱 단계는 LLM이 질문에 답변하는 데 필요한 관련 정보를 효율적으로 찾을 수 있도록 외부 데이터를 준비하는 과정입니다.

# LangChain의 Document Loaders와 Text Splitters



## 문서 로더 (Document Loaders)

다양한 외부 소스에서 데이터를 LangChain의 표준 Document 형식으로 불러오는 역할을 합니다.

### 주요 특징

- ✓ 모든 문서 로더는 BaseLoader 인터페이스를 구현
- ✓ load() 메서드로 모든 문서를 한 번에 로드
- ✓ lazy\_load() 메서드로 대규모 데이터셋을 효율적으로 스트리밍

### 지원되는 데이터 소스



웹페이지



문서



데이터베이스



API



## 텍스트 분할기 (Text Splitters)

대규모 문서를 LLM이 처리할 수 있는 smaller chunks로 분할 합니다.

### 분할 전략

- ✓ 문서의 구조적 특성을 이해하여 의미 있는 단위로 분할
- ✓ LLM의 컨텍스트 창 크기에 적합한 chunk 크기 조정
- ✓ 중복 제거 및 유지보수를 위한 메타 데이터 보존

### LangChain 지원 TextSplitter 종류

#### TextSplitter

CharacterTextSplitter

RecursiveCharacterTextSplitter

TokenTextSplitter

MarkdownHeaderTextSplitter

#### 설명

특정 문자(쉼표, 마침표 등)를 기준으로 텍스트 분할

문서를 재귀적으로 작은 청크로 분할하는 가장 일반적인 분할기

토큰 수를 기준으로 텍스트 분할

마크다운 헤더를 기준으로 분할

# 청킹(Chunking)

청킹은 RAG(검색 증강 생성) 시스템에서 긴 원본 문서를 의미 있는 작은 조각(Chunk)으로 나누는 과정입니다.

이는 마치 두꺼운 책을 한 번에 읽는 대신, 장이나 단락별로 나누어 읽는 것에 비유할 수 있습니다.

원본 문서: 긴 텍스트 (예: PDF, 웹페이지)



## ! 청킹의 중요성

- ✓ 검색 정확도와 효율성에 결정적인 영향을 미침
- ✓ LLM은 처리할 수 있는 입력 텍스트의 길이에 제한이 있음

# 청킹(Chunking) - 주요 기법

## ☒ 고정 크기 분할

- 개념: 텍스트를 미리 정해진 길이(예: 1000자 또는 512 토큰)로 자르고, 인접한 청크 사이에 일부 텍스트를 겹치게 하여 문맥 연결성을 유지합니다.

원본 텍스트:

"긴 문장은 의미를 담고 있습니다. 이 문장을 청크로 나눕니다. 중첩은 문맥을 유지합니다."

긴 문장은 의미를 담고 있습니다. 이 문장을  
문장을 청크로 나눕니다. 중첩은  
중첩은 문맥을 유지합니다.

- + 구현이 간단하고 처리 비용이 낮습니다
- 청크가 의미 단위로 잘리지 않을 수 있으며, 중요한 문맥이 여러 청크에 걸쳐 분리될 수 있습니다
- 체크한 문서 유형: 구조화되지 않은 산문 형태의 문서 (설문 조사, 포럼 게시물, 리뷰)

## ₩ 재귀적 문자 분할

- 개념: 여러 구분자(delimiter)를 우선순위에 따라 재귀적으로 적용하여 텍스트를 분할합니다. 가장 큰 구분자부터 시도하여 문맥을 최대한 보존하려 노력합니다.

원본 텍스트:

"첫 번째 문단입니다.\n\n두 번째 문단입니다. 이 문단은 여러 문장으로 구성됩니다.\n\n마지막 문장입니다.\n\n세 번째 문단입니다."

첫 번째 문단입니다.

두 번째 문단입니다. 이 문단은 여러 문장으로 구성됩니다.

마지막 문장입니다.

세 번째 문단입니다.

- + 의미적으로 가장 연관성이 강한 텍스트 조각을 가능한 한 길게 유지하여 문맥 보존에 유리합니다
- 체크한 문서 유형: 일반적인 텍스트 문서, 특히 문단 구조가 명확한 문서
- 구분자 선택과 순서에 따라 결과가 크게 달라질 수 있습니다

# 토큰화(Tokenization) - 개념

## A 토큰화란?

토큰화는 청킹된 텍스트 조각을 모델이 처리할 수 있는 최소 단위인 토큰(Token)으로 다시 분해하는 과정입니다.

AI 모델은 사람이 이해하는 단어나 문장이 아닌, 숫자 ID로 변환된 토큰을 입력으로 받아 처리합니다. 이 과정은 텍스트 데이터를 컴퓨터가 이해하고 분석할 수 있는 형태로 변환하는 데 필수적입니다.

### 토큰화의 특징

- ✓ 토큰은 일반적으로 단어, 문장 또는 구절일 수 있으며, 사용되는 토크나이저에 따라 단어의 일부 조각(서브워드)이 될 수도 있습니다.
- ✓ 토큰화 후 각 토큰은 고유한 숫자 ID로 맵핑되어 모델의 입력으로 사용됩니다.
- ✓ 서브워드 토큰화는 'Embedding'과 같은 긴 단어를 ['Embed', '#ding']과 같은 조각으로 분할하여 처리합니다.

### 토큰화 과정 시각화

원본 청크: "RAG 시스템에서 데이터 준비"



RAG 시스템에서 데이터 준비

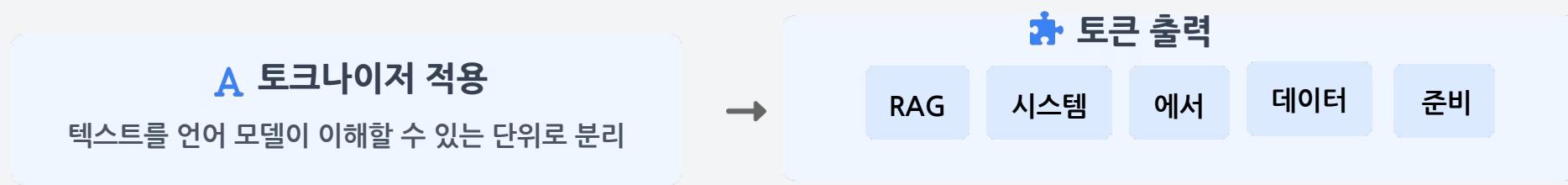


토큰 ID: [1234, 5678, 9012, 3456, 7890]

# 토큰화(Tokenization) - 과정 시각화

## “ 토큰화 예제

원본 청크: "RAG 시스템에서 데이터 준비"



## ▣ 토큰 → 숫자 ID 매핑



**i** 서브워드 토큰화: 'Embedding'과 같은 단어는 ['Embed', '##ding']과 같은 서브워드 토큰으로 분리될 수 있으며, 각 서브워드 토큰은 고유한 숫자 ID를 갖게 됩니다.

# 임베딩(Embedding) - 개념

## ▣ 임베딩이란?

임베딩(Embedding)은 토큰화된 텍스트 데이터를 컴퓨터가 이해하고 처리할 수 있는 **다차원 숫자 벡터(Vector)**로 변환하는 과정입니다.

### 💡 핵심 개념

- ✓ 토큰의 의미적 특성과 문맥적 관계를 수치적으로 표현
- ✓ 유사한 의미를 가진 단어나 문장은 벡터 공간에서 서로 가깝게 배치
- ✓ 의미가 다른 단어는 멀리 떨어져 배치되어 구별됨

### ▶ 임베딩 과정 시각화



A

토큰화된 텍스트: ['사과', '배', '자동차']



임베딩 모델 처리: 텍스트를 벡터로 변환

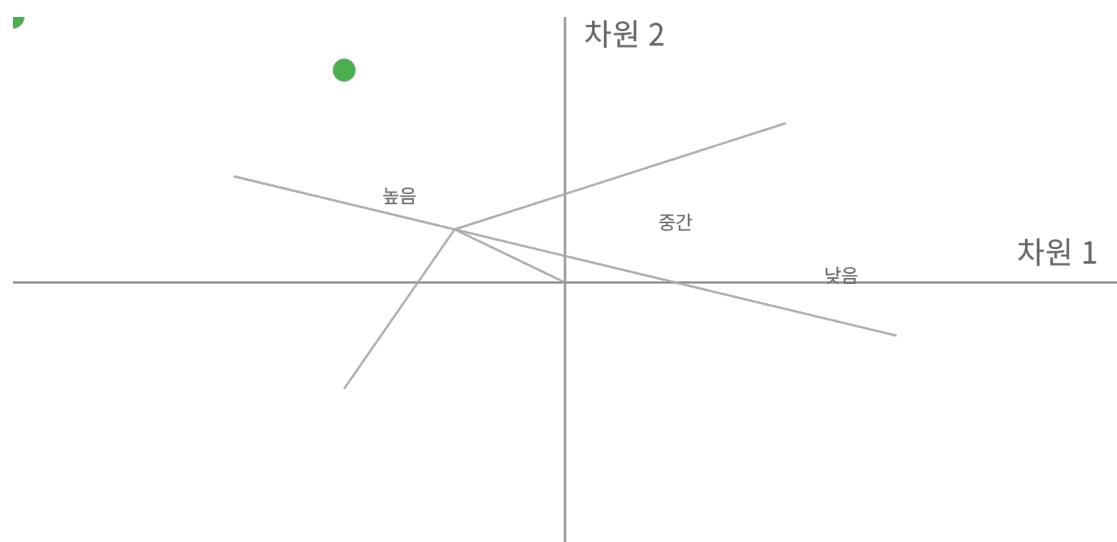


결과 벡터: 사과→[0.2, 0.5, 0.8, ...], 배→[0.3, 0.6, 0.7, ...], 자동차→[0.9, 0.1, 0.2, ...]

# 벡터 데이터베이스 저장 및 검색

## 벡터 데이터베이스 저장 및 검색 과정

- ✓ 임베딩된 벡터는 벡터 데이터베이스에 저장
- ✓ 벡터 데이터베이스는 고차원 벡터를 효율적으로 처리하도록 설계
- ✓ 쿼리 또한 동일한 임베딩 모델을 통해 벡터로 변환
- ✓ 유사도 측정(유클리드 거리, 코사인 유사도)으로 관련 청크 검색



## 검색 흐름

- 1 원본 문서 청크  
→ 청킹 → 토큰화 → 임베딩
- 2 벡터 데이터베이스에 저장  
→ 임베딩 벡터 + 메타데이터(원본 문서 ID, 청크 번호 등)
- 3 사용자 쿼리 처리  
→ 동일한 임베딩 모델 적용
- 4 유사도 계산 및 검색  
→ 쿼리 벡터와 문서 벡터의 유사도 비교
- 5 LLM으로 전달  
→ 메타데이터를 활용하여 생성된 정보 힌트가 LLM에

# RAG 시스템에서 Vector Store의 역할

## Vector Store란?

임베딩된 벡터를 저장하고, 사용자 쿼리에 가장 관련성 높은 정보를 신속하게 찾아내는 핵심 구성 요소

### 핵심 기능

- 텍스트 데이터를 **벡터 형태**로 변환하여 저장
- 사용자 쿼리 벡터와 문서 벡터 간의 **유사도 측정**
- 가장 관련성 높은 문서를 효율적으로 **검색**

### Q 벡터 임베딩의 중요성

의미적 유사성을 포착하여, 단순한 텍스트 매칭을 넘어 **문맥과 의미**를 이해하는 검색 가능함

## RAG 파이프라인에서 Vector Store의 위치



문서 저장

원본 텍스트 데이터를 Vector Store에 저장



벡터화

텍스트를 의미 벡터로 변환



검색

**Vector Store**에서 쿼리와 유사한 문서 검색



생성

검색된 문서를 기반으로 답변 생성

# 임베딩 및 벡터 저장소 구축

## 임베딩 모델 선택



### OpenAIEmbeddings

OpenAI API를 사용하는 클라우드 기반 임베딩 모델



### HuggingFaceEmbeddings

오픈소스 모델을 로컬에서 실행할 수 있는 옵션

## FAISS

- ✓ Facebook AI에서 개발한 벡터 저장소
- ✓ 빠른 유사도 검색 가능

## Chroma

- ✓ 클라우드 기반 벡터 저장소
- ✓ persist\_directory 매개변수로 로컬 저장 가능

## </> 벡터 저장소 구축 코드

### FAISS 예시

```
from langchain_community.vectorstores import FAISS
from langchain_community.embeddings import OpenAIEmbeddings

# 임베딩 모델 초기화
embeddings = OpenAIEmbeddings()

# FAISS 벡터 저장소 생성
db = FAISS.from_documents(chunks, embeddings)
```

### Chroma 예시

```
from langchain_community.vectorstores import Chroma

db1 = Chroma.from_documents (
    documents, # 청크
    embedding_model,
    collection_name='kakao_bank_2024',
    persist_directory='./chroma_db',
    collection_metadata={'hnsw:space': 'cosine'}
)
```

# HNSW (Hierarchical Navigable Small World graph)

효율적인 근접 이웃 탐색을 위한 고차원 데이터 구조

계층적 그래프 구조의 원리와 시각적 분석

# HNSW 등장 배경과 필요성

## 고차원 데이터 증가

이미지, 비디오, 텍스트 등 다양한 형태의 데이터는 벡터 공간에서 수백에서 수천 차원에 이르는 고차원 특징 벡터로 표현됩니다.

## 차원의 저주(Curse of Dimensionality)

고차원 공간에서는 모든 데이터 포인트를 일일이 비교하는 완전 탐색 방식은 계산 비용이 기하급수적으로 증가하여 현실적으로 불가능합니다.

## 근사 최근접 이웃 탐색의 중요성

차원의 저주를 극복하기 위해 등장한 **ANN(Accurate Nearest Neighbor)** 기법으로, HNSW는 고차원 데이터셋에서 빠르고 정확한 근사 최근접 이웃 탐색을 가능하게 합니다.

## 차원의 저주 시각화

### 차원의 저주 시각화



비교 대상 수:  
 $O(n)$

비교 대상 수: 복잡도 증가  
 $O(n^2)$

비교 대상 수:  
 $O(n^3)$

HNSW는 ANN 분야에서 뛰어난 성능을 보여주는 대표적인 알고리즘으로, 대규모 데이터 처리 환경에서 필수적인 역할을 수행하고 있습니다.

# HNSW 핵심 원리 개요

## 계층적 그래프 구조

**다층 그래프**: 여러 개의 "작은 세계" 그래프가 계층적으로 쌓인 구조

**최상위 계층**: 적은 노드, 긴 연결(롱 링크) - 넓은 영역 빠르게 탐색

**하위 계층**: 많은 노드, 짧은 연결(숏 링크) - 지역적 정밀 탐색

## 작은 세계 이론 적용

**작은 세계 네트워크**: 대부분의 노드가 적은 단계로 연결 가능한 구조

**탐색 효율성**: 상위 계층에서 전역 탐색 후 하위 계층에서 지역 탐색으로 정확도 향상

## 핵심 원리 요약

HNSW는 계층적 그래프와 작은 세계 속성을 활용하여 고차원 데이터에서 효율적인 근사 최근접 이웃(ANN) 탐색을 가능하게 합니다. 마치 고속도로(상위 계층)와 골목길(하위 계층)을 조합하여 목적지에 빠르게 도달하는 것과 유사합니다.

## HNSW 계층적 그래프 구조

### 최상위 계층 (Sparse Layer)

노드 수: 8

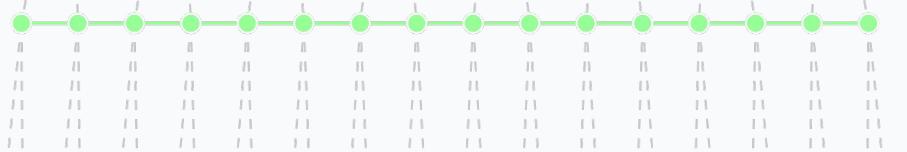


### 계층별 특성

- 최상위: 긴 연결
- 중간: 중간 연결
- 최하위: 짧은 연결

### 중간 계층 (Medium Layer)

노드 수: 16



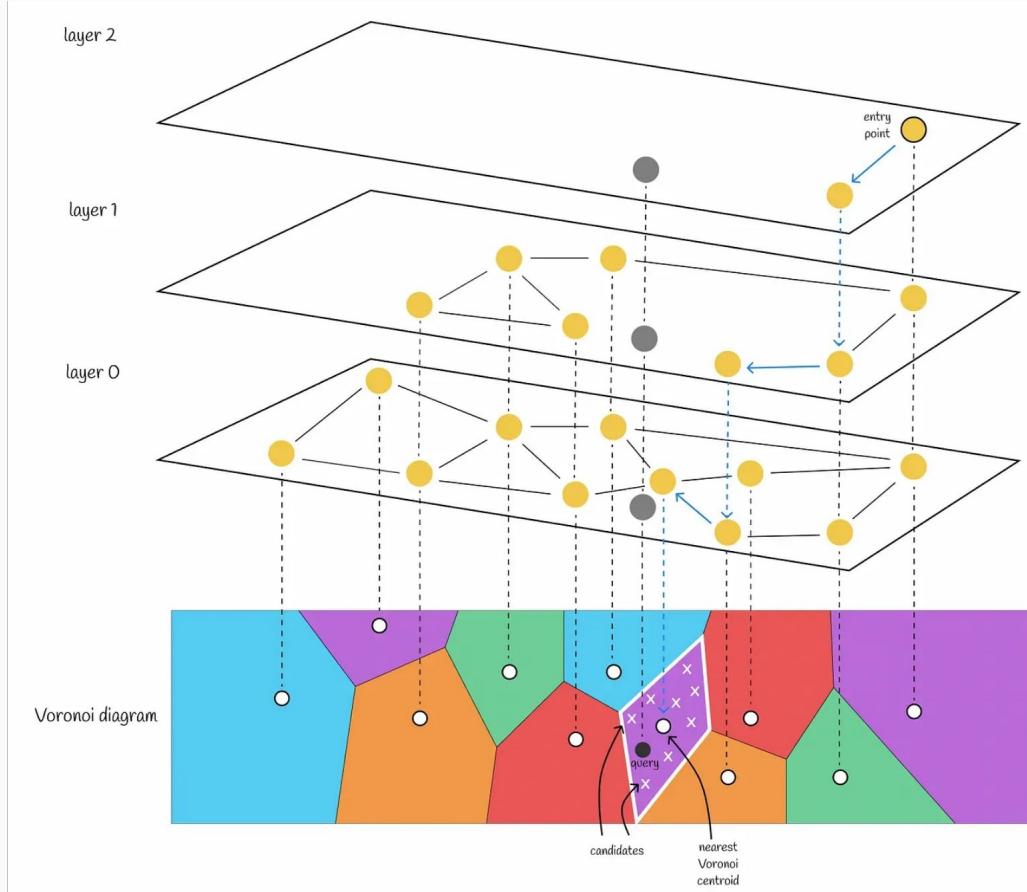
### 최하위 계층 (Dense Layer)

노드 수: 32



상위 계층: 적은 노드, 긴 연결 → 하위 계층: 많은 노드, 짧은 연결

# 계층적 그래프 구조 시각화



## 계층적 구조 특징

HNSW는 여러 개의 "작은 세계" 그래프가 계층적으로 쌓여 있는 형태로, 각 계층은 서로 다른 연결 밀도를 가집니다.

### 최상위 계층 (Sparse Layer)

가장 적은 수의 노드를 포함하며, 노드 간의 연결은 길고 드뭅니다. 이 계층은 넓은 영역을 빠르게 탐색하여 쿼리 포인트에 근접한 대략적인 위치를 찾습니다.

### 하위 계층 (Dense Layer)

계층이 아래로 내려갈수록 노드의 수가 많아지고, 노드 간의 연결은 짧고 조밀해집니다. 최하위 0계층은 모든 데이터 포인트를 포함합니다.

### 탐색 효율성

이러한 계층 구조는 탐색 시 상위 계층에서 전역적인 탐색을 통해 탐색 범위를 빠르게 좁힌 후, 하위 계층에서 지역적인 탐색을 통해 정밀도를 높입니다.

# 최근접 이웃 탐색 과정 시각화

## 탐색 과정 단계

### 1. 최상위 계층 진입점 시작

그래프의 최상위 계층(entry point)에서 탐색을 시작합니다.

### 2. 탐욕적 탐색 (Greedy Search)

현재 계층에서 쿼리와 가장 가까운 노드를 찾아 이동합니다. 더 이상 가까운 이웃을 찾을 수 없을 때까지 반복합니다.

### 3. 계층 하강

현재 계층에서 더 가까운 이웃이 없으면, 찾은 가장 가까운 노드를 다음 하위 계층의 시작점으로 사용합니다.

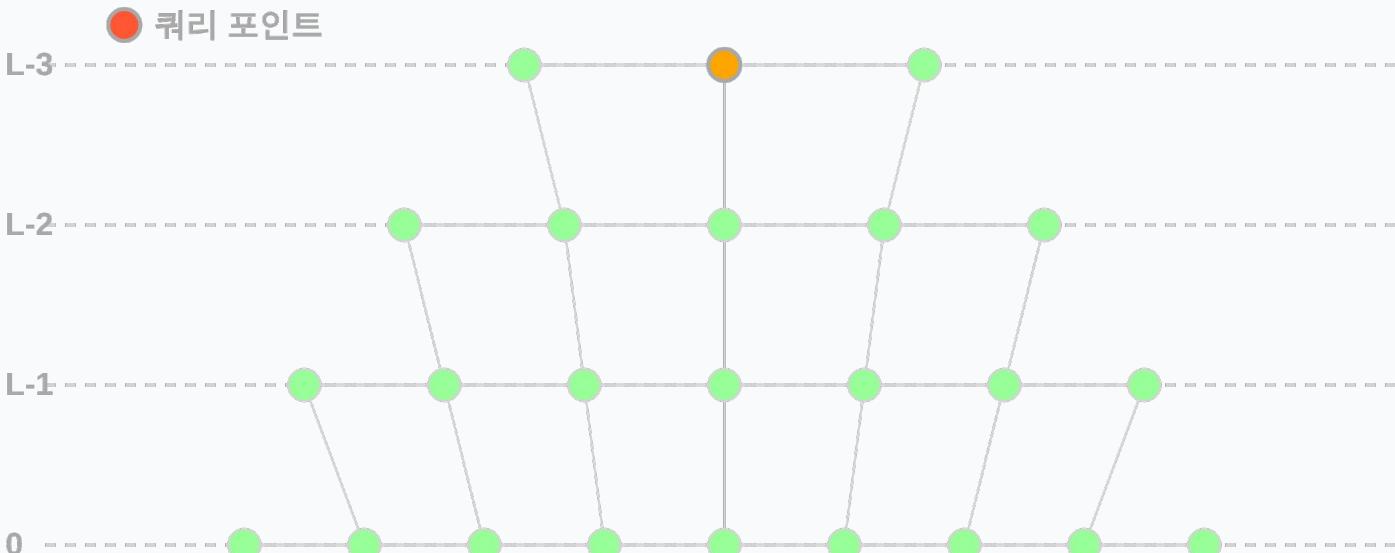
### 4. 최하위 계층 탐색 및 결과 반환

최하위 계층(0계층)에서 최종적인 k개의 최근접 이웃을 찾아 반환합니다.

💡 이 계층적 탐욕적 탐색 방식은 상위 계층에서 넓은 범위를 빠르게 탐색하여 전역 최적해에 근접한 영역을 찾고, 하위 계층에서 정밀하게 탐색하여 최종 결과를 효율적으로 찾아냅니다.

## HNSW 탐색 과정 시각화

● 쿼리 포인트 ● 현재 노드 ● 탐색 대상 노드

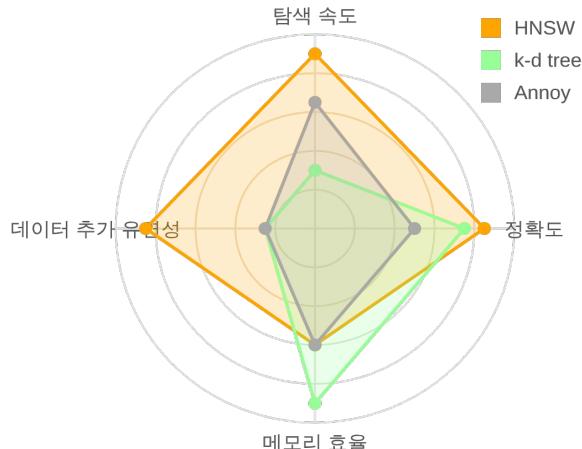


# 성능 비교 및 장점 분석

## HNSW vs ANN 알고리즘 성능 비교

비교 항목	HNSW	k-d tree	Annoy
탐색 속도	매우 빠름 (로그 스케일)	느림 (고차원 데이터에서 성능 저하)	빠름 (트리 깊이에 비례)
정확도	매우 높음 (파라미터 조정으로 최적화 가능)	높음 (정확한 탐색에 가까움)	중간 (트리 수에 따라 변동)
메모리 사용량	중간 (그래프 구조 저장)	낮음 (트리 구조 저장)	중간 (다수의 트리 저장)
데이터 추가 유연성	매우 높음 (동적 삽입 및 삭제 용이)	낮음 (트리 재구축 필요)	낮음 (트리 재구축 필요)

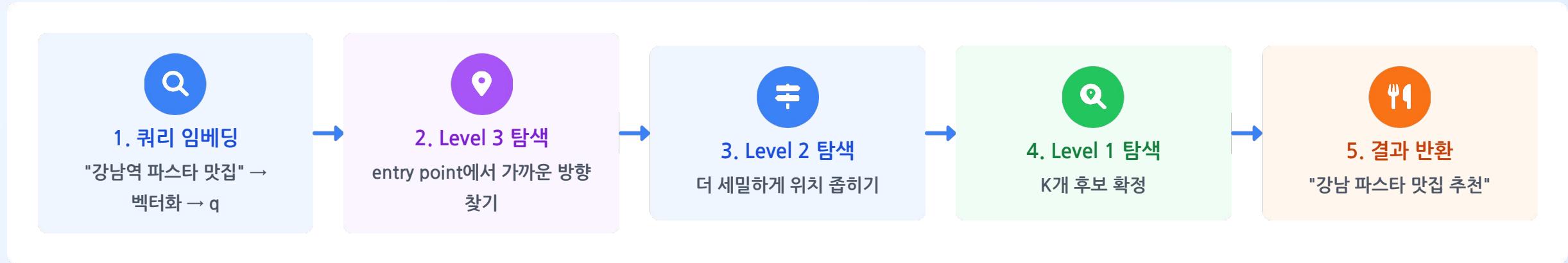
## 성능 지표 시각화



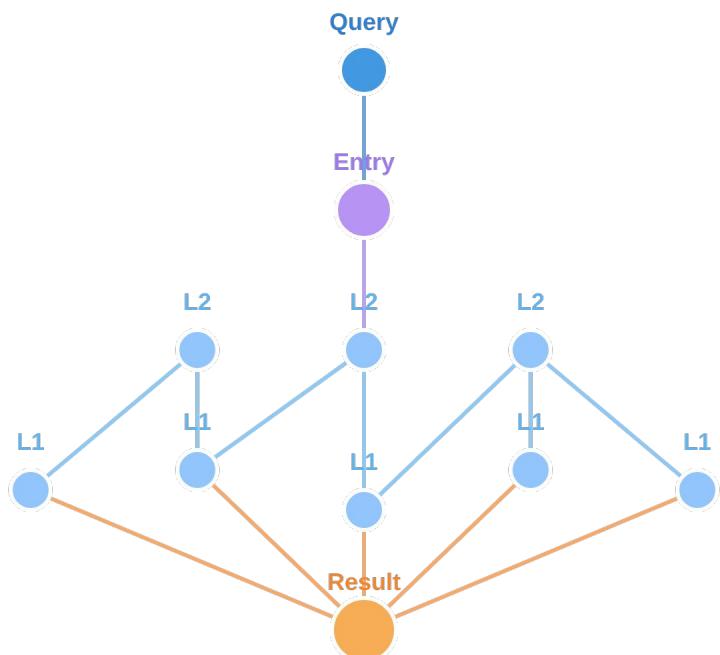
### HNSW의 주요 장점

- 속도와 정확도의 최적 균형**  
탐색 속도와 정확도를 동시에 높이면서도 메모리 사용량을 효율적으로 관리합니다.
- 동적 데이터 처리**  
데이터가 지속적으로 추가되거나 변경되는 환경에서도 효율적으로 작동합니다.
- 계층적 구조의 효율성**  
여러 계층의 그래프를 통해 넓은 탐색 범위와 정밀한 탐색을 동시에 달성합니다.

# HNSW 전체 탐색 흐름 정리



## HNSW 탐색 경로



## 탐색 과정 요약

**쿼리 처리:**  
사용자 쿼리 "강남역 파스타 맛집 추천해줘"를 임베딩 모델로 벡터화

**레벨 3 탐색:**  
최상위 entry point에서 q와 가장 가까운 방향만 따라 이동  
전국 지도에서 대충 "서울 쪽이네" 정도의 방향만 잡는 단계

**레벨 2 탐색:**  
Level 3에서 찾은 노드를 시작점으로 하여 더 가까운 노드로 이동  
서울 지도에서 "강남구 근처"까지 좁혀가는 과정

**레벨 1 탐색:**  
주변 노드들을 넓게 보며 K개 후보를 확정  
구체적인 식당 데이터들을 포함

**결과 반환:**  
K개 후보를 기반으로 결과를 추천합니다.

# 검색 및 생성 단계: Retriever의 핵심 개념

## Retriever의 역할

- 외부 지식 기반에서 관련 정보를 검색
- LLM에 컨텍스트로 제공하여 답변의 정확성과 신뢰성 향상

## 검색의 원리



1. 질문 벡터화



2. 문서 벡터 저장



3. 유사도 검색



4. 관련 문서 추출

### 벡터화 과정

- 임베딩 모델을 통해 질문과 문서를 벡터로 변환
- 벡터 저장소에 문서 벡터를 효율적으로 저장

### 유사도 검색 메커니즘

- 코사인 유사도(Cosine Similarity)를 사용하여 벡터 간 유사성 측정
- 상위 N개의 문서를 관련성 높은 결과로 추출

# Retriever 유형 및 특징



## Vector Store Retriever

가장 기본적인 Retriever 유형으로, 벡터 저장소와 질문 간의 유사도를 계산

### 작동 원리

질문 벡터와 문서 벡터 간의 코사인 유사도를 계산하여 상위 K개 문서 반환

### 활용 사례

간단한 질의응답 시스템, 구현이 용이한 경우에 적합



## Multi-Query Retriever

단일 질문을 LLM을 사용하여 여러 관점으로 변환 후 검색

### 작동 원리

LLM에 질문을 전달하여 여러 형태로 재구성한 후 각각에 대해 검색 수행

### 활용 사례

복잡하고 다각적인 정보가 필요한 질문에 효과적



## Contextual Compression Retriever

검색된 문서에서 질문과 관련된 부분만 추출하여 LLM에 전달

### 작동 원리

압축기(Compressor) 모델을 사용하여 문서 내 질문과 관련된 핵심 부분만 추출

### 활용 사례

기본 문서에서 핵심 정보를 차이야 할 때 유통하며 토크 스트리밍 방식으로 활용



## Self-Query Retriever

natural Language 질문을 분석하여 메타데이터 필터와 쿼리 생성

### 작동 원리

LLM을 사용하여 질문에서 메타데이터 필터와 핵심 쿼리 문자열 추출

### 활용 사례

"2022년 이후에 출시된 문서 중 PAG에 대한 내용"과 같은 자연어 질문 처리

# LangChain RAG Vector Store 검색 알고리즘

## Similarity Search와 MMR Search 비교 분석



### Similarity Search

벡터 간 유사도를 기준으로  
가장 관련성 높은 문서 검색



### MMR Search

관련성과 다양성을 동시에  
고려한 검색 알고리즘

# 유사도 검색(Similarity Search) 개념

## 🔍 유사도 검색이란?

사용자의 쿼리 벡터와 Vector Store에 저장된 문서 벡터들 간의 유사도를 측정하여 가장 유사한 문서를 찾아내는 기본적인 검색 방식

### ▣ 주요 유사도 지표

#### 코사인 유사도 (Cosine Similarity)

두 벡터가 가리키는 **방향**이 얼마나 유사한지를 측정. 문서 길이에 영향을 받지 않음. 값 범위: -1~1

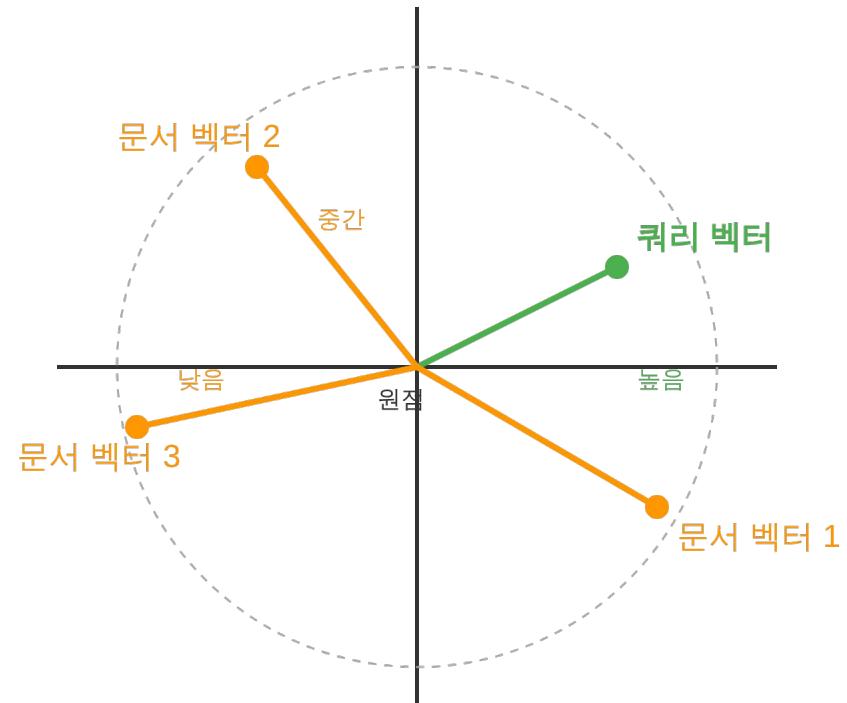
#### 유clidean 거리 (Euclidean Distance)

다차원 공간에서 두 벡터 간의 직선 **거리**를 측정. 거리가 짧을수록 유사도가 높음

#### 내적 (Dot Product)

벡터의 크기와 그 사이 각도의 코사인 값을 곱함. 벡터의 크기에 영향을 받음

### 벡터 공간에서의 유사도 측정



# 유사도 검색의 장점과 한계

## +

### 장점 (Pros)



#### 빠른 검색 속도

쿼리 벡터와 모든 문서 벡터 간의 유사도를 비교하는 비교적 간단한 연산으로, 계산 비용이 낮고 빠릅니다.



#### 높은 관련성

쿼리와 가장 의미적으로 유사한 문서를 높은 순위로 반환하여, 쿼리 관련성 측면에서 높은 정확성을 보입니다.



의미적 유사성을 포착하여 텍스트의 내용보다는 맥락과 의미를 기준으로 검색합니다.

## -

### 한계 (Cons)



#### 결과 다양성 부족

검색 결과가 쿼리와 유사한 내용으로 집중되어 다양성이 부족할 수 있습니다. 중복되거나 유사한 정보가 많이 포함될 가능성이 있습니다.



#### 중복 문제

유사한 내용의 문서들이 반복적으로 반환되는 문제가 발생하여, 사용자에게 불필요한 정보가 많게 될 수 있습니다.



특정 상황에서는 단순한 유사도 검색보다는 MMR 검색과 같은 다양성을 고려한 알고리즘이 더 적절할 수 있습니다.

# 유사도 검색의 장점과 한계

## +

### 장점 (Pros)



#### 빠른 검색 속도

쿼리 벡터와 모든 문서 벡터 간의 유사도를 비교하는 비교적 간단한 연산으로, 계산 비용이 낮고 빠릅니다.



#### 높은 관련성

쿼리와 가장 의미적으로 유사한 문서를 높은 순위로 반환하여, 쿼리 관련성 측면에서 높은 정확성을 보입니다.



의미적 유사성을 포착하여 텍스트의 내용보다는 맥락과 의미를 기준으로 검색합니다.

## -

### 한계 (Cons)



#### 결과 다양성 부족

검색 결과가 쿼리와 유사한 내용으로 집중되어 다양성이 부족할 수 있습니다. 중복되거나 유사한 정보가 많이 포함될 가능성이 있습니다.



#### 중복 문제

유사한 내용의 문서들이 반복적으로 반환되는 문제가 발생하여, 사용자에게 불필요한 정보가 많게 될 수 있습니다.



특정 상황에서는 단순한 유사도 검색보다는 MMR 검색과 같은 다양성을 고려한 알고리즘이 더 적절할 수 있습니다.

# MMR 검색(Maximal Marginal Relevance) 개념

## MMR이란?

검색 결과의 **관련성(relevance)**과 **다양성(diversity)**을 동시에 최적화하는 알고리즘

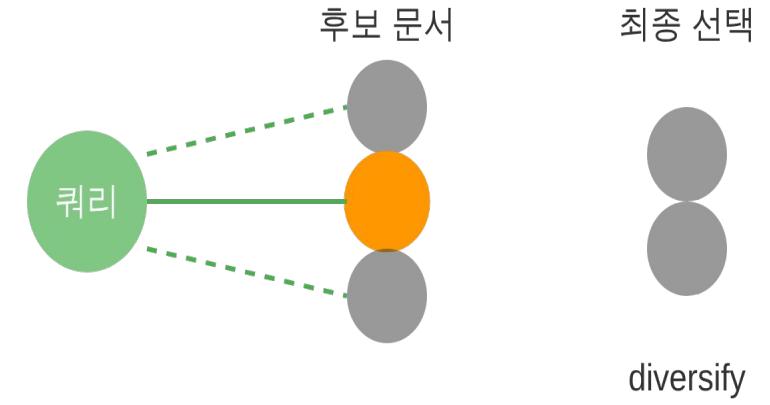
### ⚙️ 2단계 작동 방식

#### 1단계: 유사도 기반 후보군 선택

쿼리 벡터와 유사도 검색을 통해 가장 관련성이 높은 **fetch\_k**개의 문서 후보군을 선정

#### 2단계: 다양성을 고려한 최종 문서 재선택

선택된 후보군 내에서 쿼리와의 유사도뿐만 아니라 이미 선택된 문서들과의 유사도도 함께 고려하여 **diversity**를 극대화



MMR은 쿼리와 관련성이 높으면서도 이미 선택된 문서들과는 다른 정보를 포함하는 문서를 선정

## ⚠️ 관련성과 다양성의 균형

파라미터  **$\lambda$** 를 조절하여 쿼리 관련성과 문서 다양성의 중요도를 조절 가능

- $\lambda$ 가 1에 가까울수록 관련성에 초점
- $\lambda$ 가 0에 가까울수록 다양성에 초점

# MMR 검색의 장점과 활용 사례

## ★ 장점

### 정보의 중복 최소화

유사한 내용의 문서들이 반복적으로 반복되는 문제를 해결하여, 사용자에게 더 효율적이고 압축된 정보를 제공합니다.

### 주제에 대한 다각적 시각 제공

다양한 관점이나 세부 정보를 포함하는 문서를 함께 반환하여, 사용자가 특정 주제에 대해 보다 포괄적인 이해를 할 수 있도록 돕습니다.

### MMR vs 유사도 검색



유사도 검색  
중복 문서 반복



MMR 검색  
다양한 문서

## 활용 사례

### 복잡하거나 개방적인 질문에 대한 답변

"기후 변화가 경제에 미치는 영향은 무엇인가?"와 같은 질문에 대해, MMR은 탄소 배출량 관련 문서뿐만 아니라 재생 에너지 투자, 농업 생산성 변화, 보험 산업의 위험 증가 등 다양한 측면의 정보를 제공합니다.

### 특정 주제에 대한 요약 보고서 생성

특정 주제에 대한 포괄적인 요약 보고서를 작성할 때, MMR은 핵심적인 정보와 더불어 다양한 관점의 자료를 수집하여 보고서의 완성도를 높일 수 있습니다.

### 브레인스토밍 및 아이디어 탐색

새로운 아이디어를 얻거나 특정 문제에 대한 다양한 해결책을 탐색할 때, MMR은 예상치 못한 관련성 있는 정보를 제공하여 창의적인 사고를 촉진합니다.

# 상황별 검색 방식 선택 전략

RAG 시스템의 효율성과 성능을 극대화하기 위해서는, 사용자의 질문 유형과 시스템의 목표에 따라 적절한 검색 알고리즘을 선택하는 것이 중요합니다.

## ▣ 유사도 검색

적합한 상황

- 단순 사실 확인 질문
- 명확한 답변이 필요한 질문
- 쿼리와 직접적으로 관련된 정보를 빠르게 찾을 때
- 실시간 응답 속도가 중요한 시스템

✓ 높은 정확도

⌚ 빠른 응답

## ☒ MMR 검색

적합한 상황

- 개방형 질문 또는 탐색적 질문
- 특정 주제에 대한 포괄적인 정보 수집
- 브레인스토밍 및 다양한 관점 확보
- 중복 정보 최소화가 필요한 경우

✓ 높은 정보 풍부도

다양한 관점

## 💡 실용적 권장사항

### ⚙️ 검색 전략 하이브리드

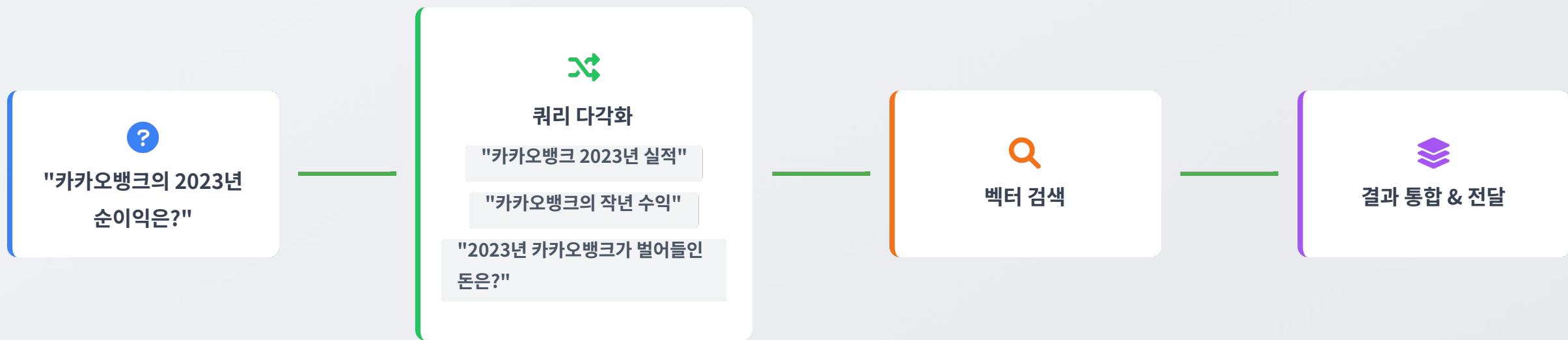
두 알고리즘을 조합하여 사용하는 전략도 고려해볼 만합니다. 예를 들어, MMR로 후보군을 선정하고, 그 중에서 유사도 검색으로 최종 정렬하는 방식 등.

### ⚠️ 실험과 최적화

데이터와 질문의 특성에 맞게 검색 전략을 실험적으로 선택하고, 실제 서비스 환경에서 최적의 파라미터를 찾는 과정이 필요합니다.

# Multi Query Retriever

## 작동 방식



사용자의 질문을 다양한 표현으로 재작성하여 검색 정확도와 포괄성을 높이는 기법

# Multi Query Retriever 개요

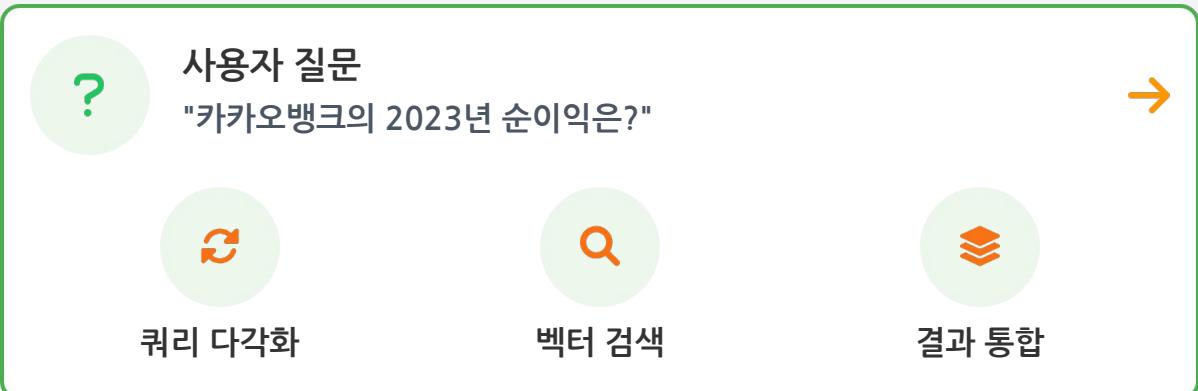
Multi Query Retriever는 사용자의 단일 질문을 여러 가지 다른 표현으로 재작성하여 검색 시스템의 정확도와 포괄성을 높이는 고급 검색 기법입니다.

## 핵심 개념

- 단일 쿼리의 한계를 극복하기 위해 다양한 표현으로 재작성
- 다양한 관점에서 정보 탐색을 통해 풍부한 결과 도출
- 벡터 검색 시스템에 개별적으로 전달하여 독립적 검색 수행
- 결과 통합 시 중복 제거 및 관련성 높은 정보 선별

## 작동 방식

LLM을 활용해 사용자 질문을 다양한 표현으로 다각화하고, 각 쿼리에 대해 독립적인 벡터 검색을 수행한 후, 중복을 제거하여 최종적으로 LLM에 전달합니다.



## 다각화된 쿼리 예시

"카카오뱅크 2023년 실적"

"카카오뱅크의 작년 수익"

"2023년 카카오뱅크가 벌어들인 돈은?"

# 1단계: 사용자 질문 입력

## 단계 설명

Multi Query Retriever의 작동은 **사용자의 질문 입력**으로 시작됩니다. 이

단계에서는 사용자가 특정 정보를 얻기 위해 시스템에 질의를 던집니다.

- 사용자는 자연어로 질문을 입력합니다
- 질문은 정보 탐색의 첫 단계이며, 시스템이 처리할 원본 요청을 설정합니다
- 이 질문은 후속 단계에서 다각화될 준비가 됩니다

### 중요한 포인트

이 단계에서 사용자의 질문은 검색 시스템의 정확도와 포괄성을 높이기 위한  
후속 과정의 입력값이 됩니다.



사용자

"카카오뱅크의 2023년 순이익은?"



Multi Query Retriever  
질문을 수신받습니다

• • •

# 2단계: 쿼리 다각화

쿼리 다각화는 사용자 질문을 다양한 표현으로 재작성하여 검색의 포괄성을 높이는 과정입니다.

## LLM의 역할

- 원본 질문의 핵심 의도를 유지하면서 다양한 표현 생성
- 키워드와 문맥을 조합하여 검색 범위 확대
- 다양한 관점에서 정보를 탐색할 수 있도록 도움

### LLM (대규모 언어 모델)

질문을 다양한 방식으로 해석하고 재작성할 수 있는 능력으로 인해 쿼리 다각화에 적합



### 원본 질문

"카카오뱅크의 2023년 순이익은?"

LLM

쿼리 다각화

#### 다양한 표현 1

"카카오뱅크 2023년 실적"

#### 다양한 표현 2

"카카오뱅크의 작년 수익"

#### 다양한 표현 3

"2023년 카카오뱅크가 벌어들인 돈은?"

### 쿼리 다각화 과정

LLM은 원본 질문의 핵심 의도를 분석하여 키워드와 문맥을 변환하거나 추가함으로써 다양한 표현으로 재작성합니다. 이 과정은 단일 쿼리의 한계를 극복하고 더 넓은 범위의 검색 결과를 포괄하는 데 기여합니다.

# 3단계: 개별 벡터 검색 수행

다각화된 각 쿼리에 대해 독립적으로 벡터 검색을 수행하여 다양한 관점에서 정보를 탐색합니다.

## 벡터 검색의 의미

- 쿼리가 벡터 공간에서 가장 유사한 문서를 찾아내기 위해 사용
- 다양한 관점에서 정보를 탐색하여 더 풍부한 결과 도출
- 각기 다른 관점에서 정보를 독립적으로 추출

## 검색 과정

- 다각화된 쿼리들을 각각 벡터 검색 시스템에 전달
- 각 쿼리에 대해 독립적으로 검색 수행
- 관련성이 높은 데이터 조각들을 추출
- 검색 결과는 후속 단계에서 통합될 예정



# 4단계: 결과 통합 및 전달

개별 벡터 검색을 통해 얻은 결과들은 하나의 통합된 풀로 모아지고, 중복되는 정보들을 제거한 후 LLM에 전달됩니다.

## 과정 설명

- 결과 통합: 각기 다른 관점에서 검색된 결과들을 하나의 풀로 모읍니다.
- 중복 제거: 동일하거나 유사한 내용의 결과들을 식별하여 제거합니다.
- 정제 및 선별: 가장 관련성이 높다고 판단되는 결과들을 선별하여 정제합니다.
- LLM 전달: 최종적으로 정제된 정보는 대규모 언어 모델에 전달되어 답변 생성에 활용됩니다.

## 효과

이 과정을 통해 Multi Query Retriever는 단일 쿼리로는 얻기 어려웠던 심층적이고 다각적인 정보를 제공할 수 있게 됩니다.



## 결과 통합 & 중복 제거 예시

### 통합된 결과 (중복 포함)

카카오뱅크 2023년 실적

카카오뱅크 2023년 실적 (중복)

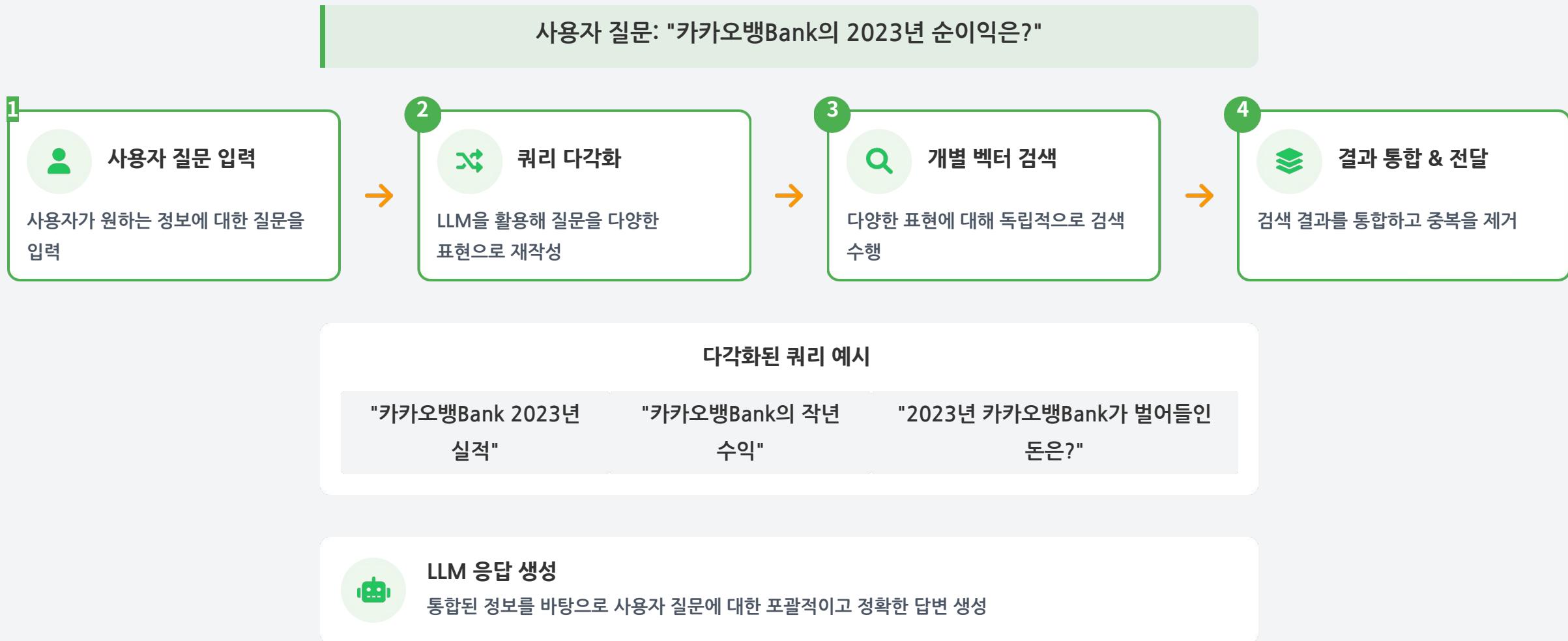
카카오뱅크 자녀 스타트업

### 최종 결과 (중복 제거)

카카오뱅크 2023년 실적

카카오뱅크 작년 수익

# Multi Query Retriever 전체 흐름도



# 기대 효과 및 장점

Multi Query Retriever는 단일 쿼리 검색의 한계를 극복하고, 사용자에게 보다 풍부하고 정확한 정보를 제공하기 위해 설계되었습니다.

## 검색 정확도 향상

다양한 표현으로 재작성된 쿼리는 원본 질문의 핵심 의도를 유지하면서도 더 정확한 검색 결과를 도출할 수 있게 합니다. 각 쿼리에 대해 독립적인 벡터 검색을 수행함으로써, 관련성이 높은 정보를 보다 쉽게 찾아낼 수 있습니다.

## 포괄적인 정보 제공

단일 쿼리로는 놓칠 수 있는 다양한 관점의 정보를 탐색함으로써, 보다 풍부하고 포괄적인 검색 결과를 도출합니다. 이는 다양한 키워드와 문맥을 포함하여 검색의 포괄성을 높임으로써 가능합니다.

## 복잡한 질문에 대한 답변 품질 향상

LLM을 통해 다각화된 쿼리는 복잡하거나 모호한 질문에 대한 답변 품질을 높이는 데 기여합니다. 다양한 관점에서 정보를 탐색하여 심층적이고 다각적인 정보를 제공할 수 있게 됩니다.

## 정보 검색 시스템의 효율성 극대화

벡터 검색 시스템에 개별적으로 전달되어 독립적인 검색을 수행함으로써, 각기 다른 관점에서 정보를 탐색하여 잠재적으로 더 풍부하고 다양한 검색 결과를 얻을 수 있습니다. 최종적으로 정제된 정보는 LLM에 전달되어 정보 검색 시스템의 효율성을 극대화합니다.