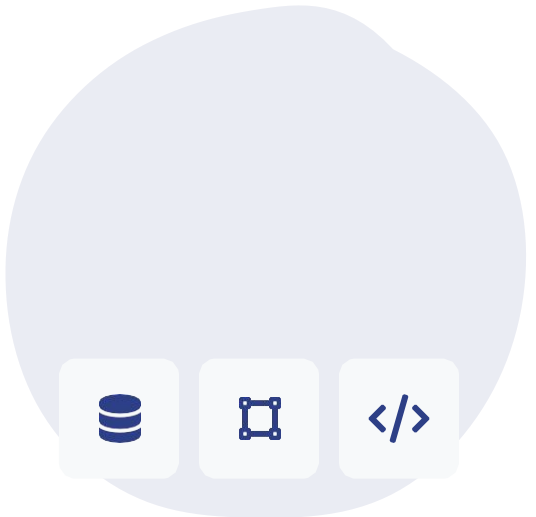
07. 실시간 반응형 추천 개발 일지 2부

벡터 검색, 그리고 숨겨진 요구 사항
기술 도입 의사 결정을 다루는 방법



아젠다

1. 프로젝트 개요 및 핵심 요구사항
2. 핵심 컴포넌트
3. 기술적 도전 과제
4. 벡터 유사도 검색: 엔지니어링 현실
5. 기술 선택 및 실험
6. 실제 적용: 현장에서의 교훈
7. 핵심 교훈

프로젝트 요구사항

실시간 관심사 파악

사용자의 최근 행동(검색, 클릭)을 통해 현재 관심사를 실시간으로 파악하고

관심사 기반 가게 매칭

사용자의 현재 관심사와 배달 반경을 고려하여 적절한 가게 추천

즉각적인 응답 시간

사용자가 추천을 요청하는 시점에 '최대한 빠르게' 추천 결과 제공

프로젝트 설명문

우리는 사용자의 관심사를 실시간으로 파악하고, 이 관심사에 맞춰 추천을 제공하는 시스템을 만들어보기로 했습니다. 그리고 이를 위해 세 가지 새로운 컴포넌트를 개발했습니다.

프로젝트 요구사항

- 다음의 모든 것은 사용자가 추천을 조회하는 시점에 수행한다.
- 사용자의 관심사를 최대한 빨리 얻어낸다.
- 사용자의 현재 위치 기반으로 배달 가능한 가게 목록을 최대한 빨리 얻어낸다.
- 사용자의 관심사와 가게의 유사도를 최대한 빨리 비교할 수 있도록 준비해야 한다.
- 사용자의 관심사에 가장 잘 맞는 순서로 최대한 빨리 정렬한다.

숨겨진 요구 사항 : 최대한 빨리!!!!

숨겨진 요구사항

- 사용자의 위치는 추천 조회 시점에만 확인 가능
- 배달 가능 영역으로 필터링 필요
- 실시간 관심사와 가게 간 유사도 빠르게 계산
- 관심도 순으로 신속하게 정렬하여 제공

핵심 컴포넌트

실시간 행동 이력 스트리밍

사용자의 검색 키워드와 클릭한 가게 등 행동 이력을 실시간으로 수집하고 처리하는 파이프라인 . 사용자 관심사를 빠르게 포착 합니다.

속도 & 확장성

인코더 모델 & 임베딩 추출

사용자 관심사와 가게 정보를 벡터 공간에 매핑하여 **의미적 유사성을 계산할 수 있는 형태로 변환하는 컴포넌트** 입니다.

관련성 & 정확도

벡터 유사도 검색

사용자 관심사 **임베딩과 가게 임베딩 간의 유사도를 효율적으로 계산하**여 사용자 관 심사에 **가장 잘 맞는 가게를 찾아내는 컴포넌트** 입니다.

응답 시간 최적화

좌표 반경 검색

사용자의 **현재 위치를 기준으로 배달 가능한 가게만 필터링하여 실질적으로 주문 가능한 추천을 제공** 합니다.

런타임 필터링

실시간 추천을 위한 임베딩 기반 처리와 제약 조건

사용자 암묵적 피드백을 임베딩으로 변환

검색 키워드와 클릭한 가게 등의 행동 데이터를 의미있는 벡터 공간으로 표현

사용자 의도와 가게 간 빠른 유사도 계산

임베딩 벡터 간 유사도 측정을 통해 사용자 관심사와 가게 간 관련성 분석

런타임에서의 지리공간 및 관심사 필터링

사용자 현재 위치 기반 배달 가능 여부와 관심사 관련성을 실시간으로 적용

핵심 제약 조건

필터링과 랭킹은 **추천 조회 시점**에만 이루어져야 함

- 미리 계산된 글로벌 인덱스 활용 불가 사용자 위치 기반 검색은 조회 시점에 발생하므로 미리 계산된 인덱스로 처리할 수 없음

벡터 검색 알고리즘의 선택과 실시간 제약의 딜레마

Exact-KNN vs ANN 알고리즘

Exact-KNN: 정확한 거리 계산, 느린 검색 속도 모든 벡터 쌍 비교 필요, 확장성 제한

ANN (HNSW, IVFFlat): 근사치 계산으로 속도 향상 재현율 손실, 사전 인덱스 구축 필요

포스트 필터 (Post-filtering) 방식의 한계

ANN으로 유사한 가게 검색 후 배달 가능한 것만 필터링 → 결과 보장성 부족 예: 상위 3,000개 검색 결과가 모두 배달 불가 지역이면 제공할 결과 없음

ex: 서울에서 주문 –> 다수의 데이터가 제주도 맛집에서 검색되는 경우 발생

프리 필터 (Pre-filtering) 방식의 도전

지역 필터링 후 유사도 검색 → 사용자 위치는 런타임에만 얻을 수 있음 사전 구축된 ANN 인덱스를 활용할 수 없는 구조적 한계 발생

핵심 해결 과제

- 시스템 제약 조건 하에서 **정확도**, **속도**, **실현 가능성** 간의 균형 필요
- 런타임 필터링과 벡터 검색을 효율적으로 결합하는 방법 모색
- 프리 필터 상황에서도 최적의 성능을 낼 수 있는 구현 방식 탐색

기술 후보군 평가 및 실험

1차 실험 OSS/사내 운영 중점

- Milvus
- Redis Stack JSON 지원
- MongoDB Atlas Search
- OpenSearch

개발 가능성 및 기능 검증 중심

– 인덱스 생성, 데이터 입력, 검색 파트에 대한 기능 구현 및 운영 가능성 –
각 시스템의 **API** 구현 방식 및 운영성 탐색

2차 실험 성능 중심 비교

1차 실험 후보군 + **Amazon RDS (pgvector)**

- Pgvector 0.5.0 (HNSW 지원)

실제 워크로드 기반 벤치마크 프리필터
성능 중점 검증(부하 테스트 후 성능 최적화 진행)

각각에 대한 총평은 아래와 같다.

	사용 가능 한가?	판단 이유	특이점

Milvus	운영 불가	• 개발팀에서 운영할 수 없음
		• 전담 운영팀이 필요할 것으로 보임
			• K8S 기반 클러스터 방식으로 확장성, 설치 등이 용이하다고 생각했음.
			• 실제로는 기반 플랫폼 의존도가 높음
			  • 테스트로 S3, Kafka로 기반 플랫폼을 분리해서 분산 노드 클러스터로 구성했음.
			• 특히 기반 플랫폼 중 Kafka 같은 경우에는 전사 카프카 팀의 도움이 필요함.
			  • 권한 이슈 등 운영 이슈가 있을 것으로 보임

redis stack	운영 불가	• ~~AWS elastic cache에서 지원되지 않음. 앞으로 지원 가능. 이부 블루엠~~ • (2023.12 추가) 서울 리전에는 아직 서비스하지 않지만 지원 예정이다.	• redis enterprise에서 사용 가능한 것으로 보임. • RedisSearch가 클러스터를 지원한다고는 되어있으나 현재까지는 Enterprise 전용 기능으로 보인다. (RSCoodinator가 OSS로 나와있다고는 하나, 문서가 부족하고, 관련 문의에도 답변이 없음.) • 클러스터링을 사용하지 않는다면, 단일 머신을 사용해야하기 때문에 스케일링에 제약이 있어보인다.
Atlas mongodb	운영 가능	• Preview 스테이지이나 기능은 동작함.	• 현재 추천팀에서 가장 활발하게 사용하고 있는 저장소로 접근성이 높다. • aggregate 쿼리 자체가 몽고디비의 CPU 연산량이 높다. 기존 처럼 K-V 저장소가 아니라 쿼리용 엔진으로 보고 운영 패턴을 바꿔야할 수 있다.
OpenSearch(OSS)	운영 가능	• 전통적 강자 • 가장 많은 옵션 (엔진, 메트릭, 각 세부 옵션등) • 시맨틱 + 렉시컬 검색 등 하이브리드 케이스에서도 가장 유리 • 태생적으로 분산처리 • OSS, Managed 구성 가능	• 예를 들면, faiss 의 pq 등이 지원된다. hnsw에서도 ef_*류의 파라미터를 제어할 수 있다. • 일부 지원되는 스페이스 검색을 위해서는 사전 모델 트레이닝이 필요 할 수 있다.

출처 : 실험 당시 작성한 결과 문서

실험 초점

실시간 필터링이 적용된 벡터 검색을 얼마나 효율적으로 처리하는가?

주요 발견점

- 대부분의 시스템은 **프리필터 제약 조건에서 Exact-KNN으로 동작 실시간 필터링을 위해 ANN 인덱스 사용이 어려움**
- 벤치마크 결과 기반으로 실제 프로덕션 워크로드에 가장 **적합한 솔루션 선정 응답 속도와 정확도 간의 균형 필요**

교훈 및 결론

숨겨진 제약 사항이 기술 선택을 좌우 한다

비즈니스 요구사항을 기술적인 문제로 환원하는 과정에서 **숨겨진 제약이 기술 선택의 핵심** 이 됨

ANN이 항상 최선의 답은 아니다

실시간 필터링이 필요한 워크로드에서는 **ANN보다 Exact-KNN이 더 실용적인 선택** 일 수 있음

실제 데이터로 검증하라

실제 데이터와 워크로드 기반의 실험이 의사 결정에 결정적인 역할 을 함

성능, 운영성, 확장성의 균형

기술 도입에서 성능만큼 중요한 것은 **플랫폼 적합성, 유지보수성, 운영 용이성**

→ 다음 단계

사용 사례가 발전하고 인프라가 성숙함에 따라 시스템을 계속 발전시키며 새로운 기술과 알고리즘을 검토하고 적용

참고. Database 관리형 vs. Managed vs. fully-managed

- 비관리형(Non-Managed)
  - 사용자가 직접 관리
  - 사용자가 데이터센터를 운영하는 경우
  - 장비 운영, OS 설치 및 운영, 데이터베이스 솔루션 설치 및 운영까지 모두 담당
- 관리형(Managed)
  - 사용자와 AWS가 함께 관리
  - AWS EC2 서버에 데이터베이스 솔루션을 설치 하고 운영하는 경우
  - 장비 운영, OS 설치 및 운영은 AWS가 담당하고, 데이터베이스 솔루션의 설치 및 운영은 사용자가 담당
- 완전관리형(Fully-Managed)
  - AWS가 모두 관리
  - AWS에서 제공하는 RDS 솔루션을 이용하는 경우
  - 장비 운영, OS 설치 및 운영, 데이터베이스 솔루션 설치 및 운영까지 AWS에서 모두 담당