9

9장 LLM 애플리케이션 개발하기

💡 **요약**

LLM 외에 다양한 요소 필요

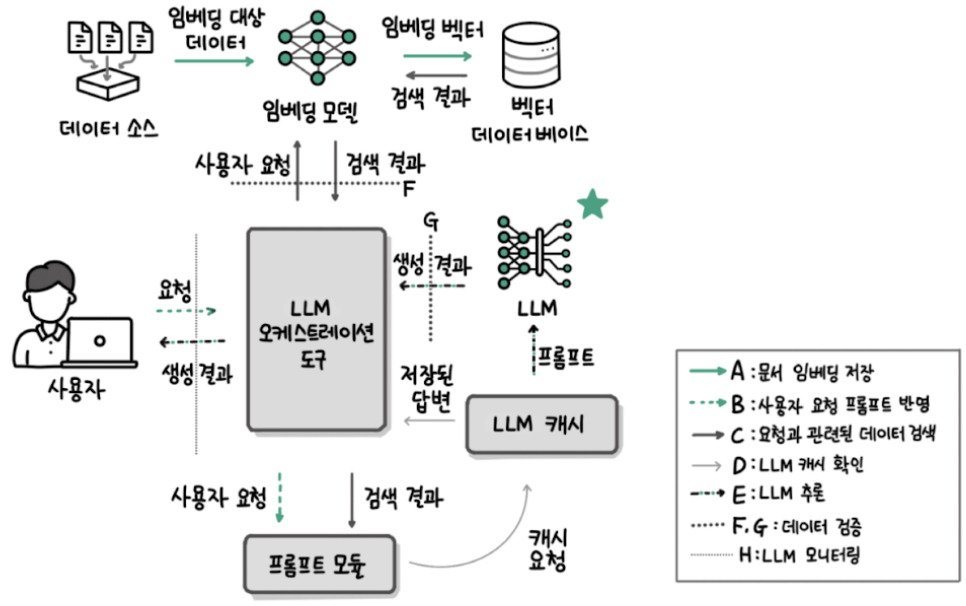


그림 9.1 LLM 애플리케이션 아키텍처

RAG Retrieval-Augmented Generation

- 검색과 생성 모델을 결합하여, 모델이 필요한 정보를 외부에서 검색한 후 그 정보를 기반으로 응답을 생성하는 방식

LLM 캐시

- 모델 응답을 빠르게 제공하기 위해, 이전의 요청에 대한 응답을 캐시하여 재사용하는 단계

데이터 검증

- 입력 데이터와 모델의 출력 결과가 정확하고 일관되게 처리되도록 검증하는 단계

데이터 로깅

- 모델의 입력과 출력 데이터를 로깅하여, 향후 분석과 성능 개선을 위해 저장하는 단계

9.1 RAG

⚒️ **RAG** *Retrieval-Augmented Generation*

검색 증강 생성은 필요한 정보를 검색하고 프롬프트를 증강해 추론(생성)하는 기술

📍 **기존 언어 모델의 한계**

기존의 사전 학습된 언어 모델은 데이터에서 많은 지식을 학습하고 이를 바탕으로 언어 생성이나 질문 응답 등의 Task에서 뛰어난 성능을 보임

1. 확장성
2. 설명 불가능성
3. 할루시네이션 hallucination

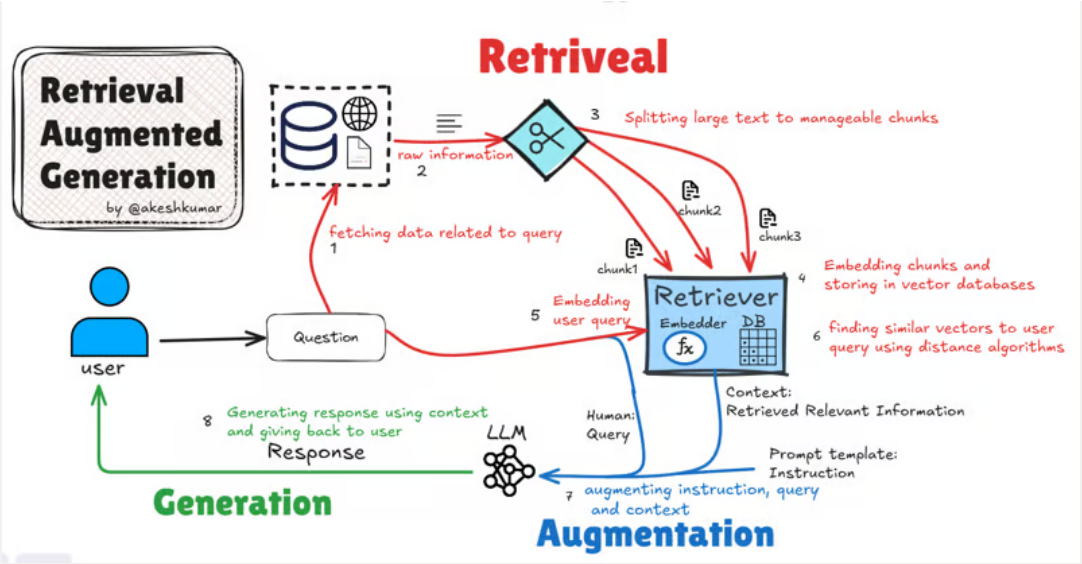
   LLM이 인간 관찰자에게는 존재하지 않거나 인식할 수 없는 패턴이나 물체를 인식하여 무의미하거나 전적으로 부정확한 아웃풋을 생성하는 현상

▼ 유사도

📌 **method**

- 내적을 통해 질문과 문서 간의 유사도를 계산
  - **MIPS** Maximum Inner Product Search
    - 상위 K개의 문서 선택 ← 질문에 대해 가장 높은 유사도를 가지는 상위 k개의 문서를 찾는 과정
    - 많은 데이터 중 가장 큰 내적을 갖는 항목을 찾는 문제로 빠르게 수행하기 위해 효율적인 검색 알고리즘 사용
  - 내적 결과가 클수록 질문과 문서가 더 관련이 있다고 판단
- 주어진 질문에 가장 관련 있는 문서들을 찾음

🔽 Full Flow



https://dev.to/akeshlovescience/rag-architecture-explained-beginner-5hn4

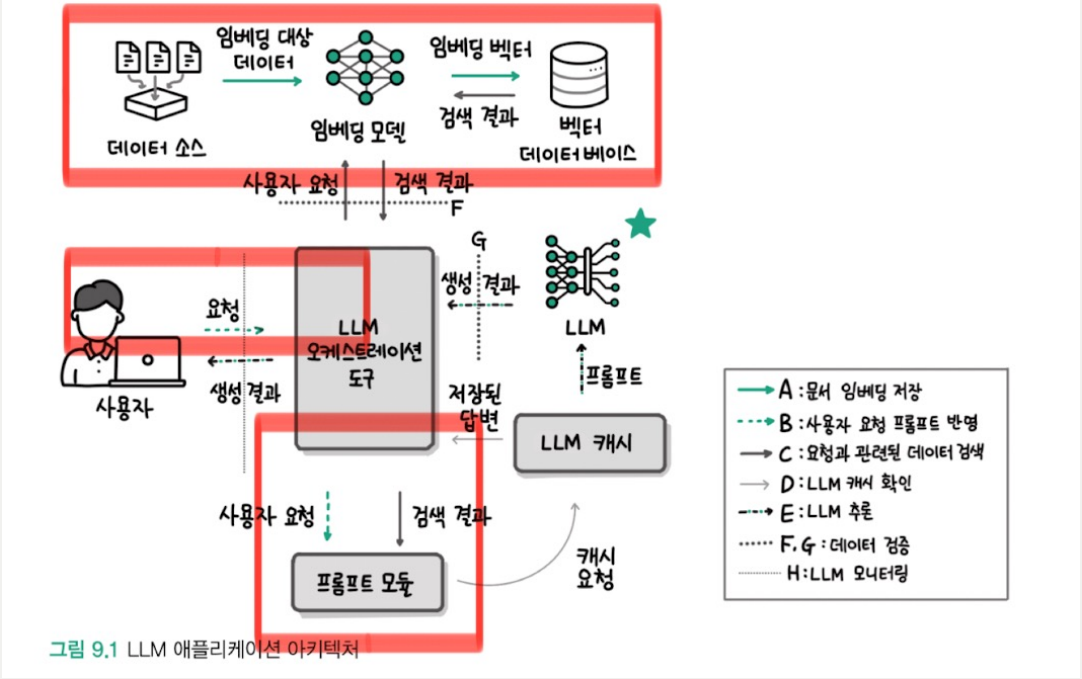
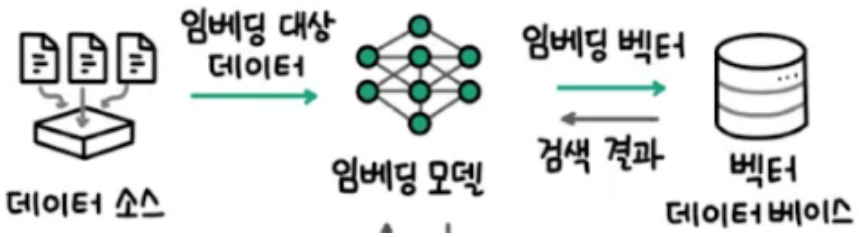


그림 9.1 LLM 애플리케이션 아키텍처

데이터 저장



- 데이터 소스 : 비정형 데이터가 저장된 데이터 저장소
- 임베딩 모델
  - 상업용 모델 OpenAI text-embedding-ada-002
  - 오픈소스 Sentence Transformers 라이브러리
- 벡터 데이터 베이스 : 임베딩 벡터의 저장소로 입력한 벡터와 유사한 벡터 찾음

검색 결과 통합 → 프롬프트

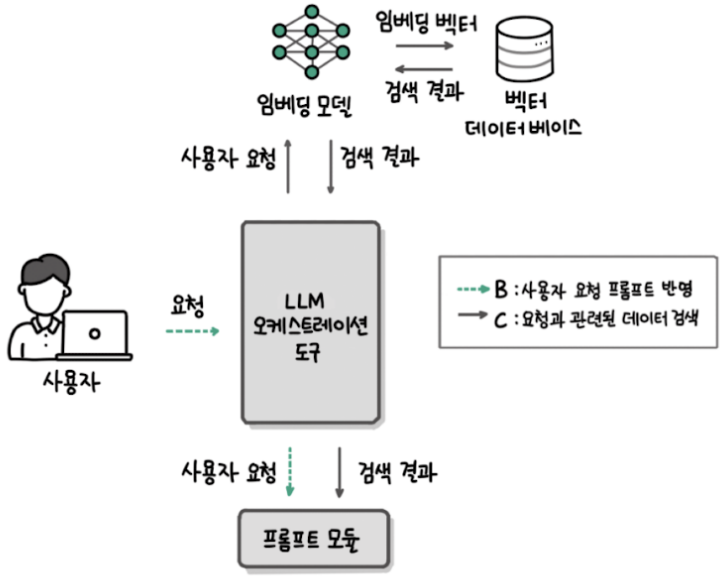


그림 9.6 검색 결과를 프롬프트에 통합

1. 사용자의 요청과 관련이 큰 문서를 벡터 데이터베이스에서 탐색 ⇒ **C**
2. 검색 결과를 프롬프트에 통합 ⇒ **B**

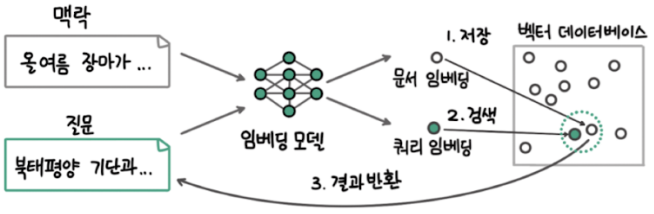


그림 9.9 예시 데이터로 검색 증강 생성을 수행하는 과정

- "북태평양 기단"과 같은 질문이 있을 때, 임베딩 모델을 통해 벡터로 변환한 후, 데이터베이스에서 관련된 정보를 찾아옴
- 사용자가 이해할 수 있는 형태로 출력

9.2 LLM 캐시

⚒️ LLM Cache

특정 입력에 대한 모델의 응답을 저장해두었다가 동일한 입력에 대한 요청이 있을 때 저장된 응답을 반환하는 기능

▼ **캐시**란?

> 자주 사용되는 데이터나 연산 결과를 임시로 저장해두어 동일한 요청이 있을 때 다시 계산하지 않고 저장된 데이터를 반환하는 기술

📍 장점

1. 동일한 완료를 여러번 요청하는 경우 LLM 공급자에 대한 API 호출 횟수를 줄여 `비용 절감`
2. API 호출 횟수를 줄여 애플리케이션 `속도 향상`

📍 캐시를 적용하는 방법 📎

1. 메모리 캐시
2. SQLite 캐시
3. Redis 캐시
4. 시맨틱 캐시

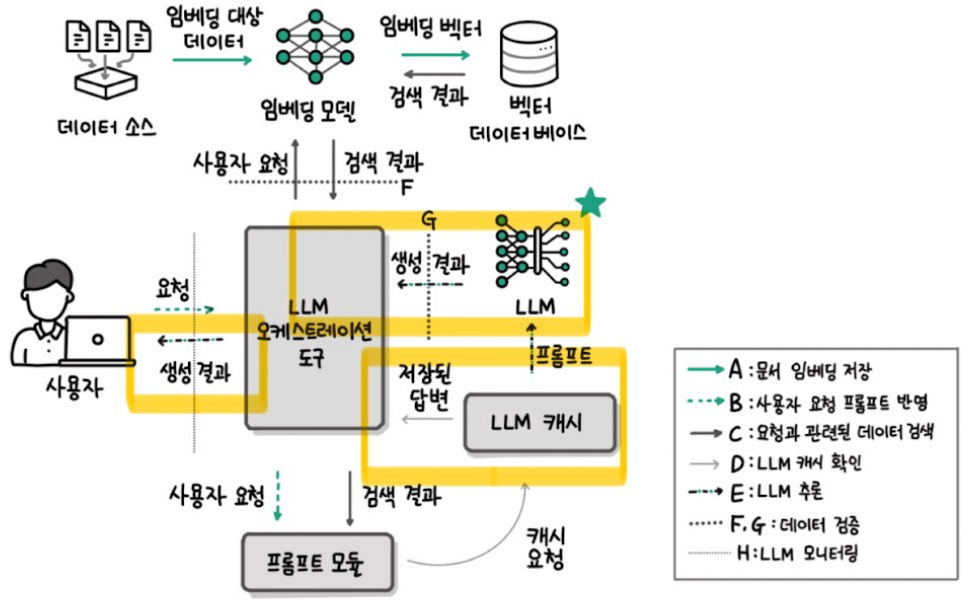


그림 9.1 LLM 애플리케이션 아키텍처

확인 과정

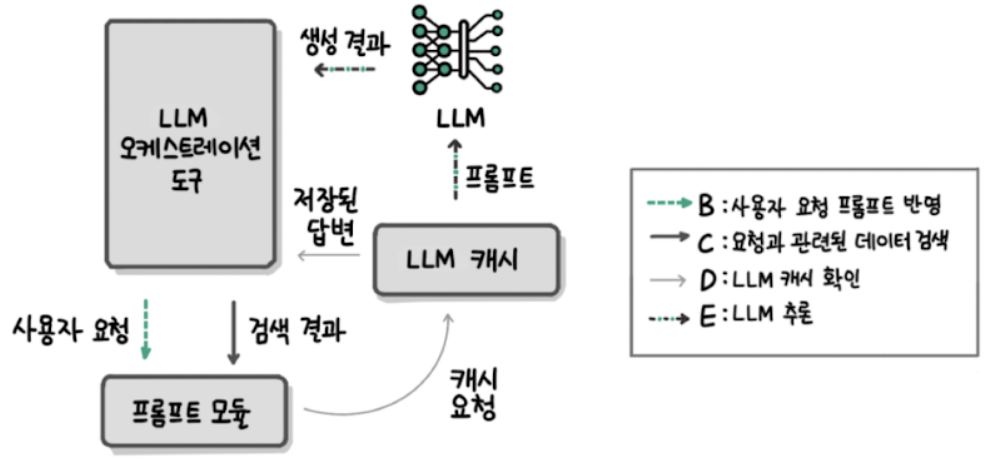


그림 9.12 프롬프트 통합과 LLM 사이에 LLM 캐시 확인 과정 추가

- 사용자가 질문을 입력하면, 임베딩 모델을 통해 이 질문은 벡터로 변환
- 변환된 벡터는 벡터 데이터베이스에서 검색되어, 유사한 벡터를 찾음
- 만약 유사한 벡터가 캐시에서 발견된다면, 저장된 텍스트를 반환하고, LLM은 새로 텍스트를 생성하지 않고 캐시된 결과를 바로 제공
- 만약 유사한 벡터가 없다면, 새로운 텍스트를 생성하여 벡터 데이터베이스에 저장하고, 그 결과를 반환

캐시 방식

Exact Match Cache

사용자의 요청이 이전에 저장된 캐시와 정확히 일치하면 그 결과 바로 반환

Similar Search Cache

정확히 일치하지 않지만 유사 요청이 이전에 있다면 그와 비슷한 결과 반환

9.3 데이터 검증

⚒️ 데이터 검증

입력 데이터와 모델의 출력 결과가 정확하고 일관되게 처리되도록 검증

📍 문제

생성형 AI 서비스 경우 사용자의 요청이 다양하며 그만큼 LLM의 생성 결과도 예측하기 어려움

▶ 해결책

1. 사용자의 요청 중 적절하지 않은 요청에 응답하지 않기
2. 검색 결과나 생성 결과가 적절치 않은 정보가 포함됐는지 확인하는 절차 필요

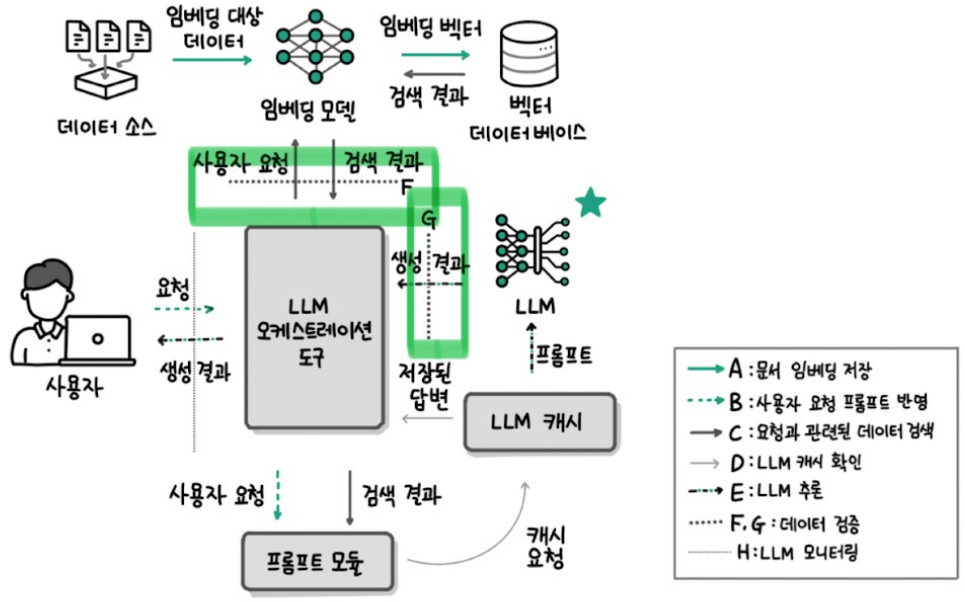


그림 9.1 LLM 애플리케이션 아키텍처

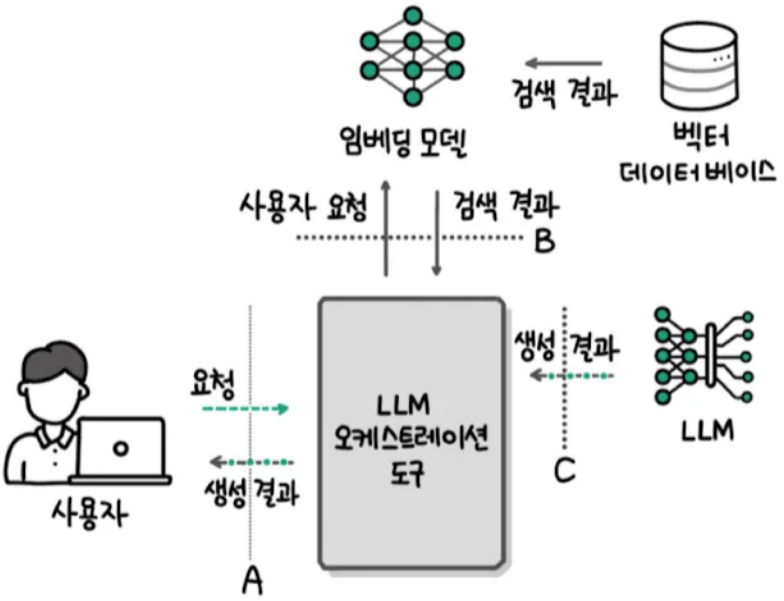


그림 9.14 LLM 애플리케이션에서 데이터 검증이 필요한 단계

⇒ 사용자 요청, 검색 결과, 생성 결과 모두 검증

- 벡터 검색과 LLM 생성 결과를 활용하여, 데이터를 검토하고 불필요한 정보나 부정확한 결과를 걸러냄
- 검증을 통해 사용자가 원하는 정확한 정보만을 제공하기 위해 추가적인 확인 작업

검증 방식

규칙 기반 검증

텍스트에서 정규 표현식을 사용해 데이터 확인

임베딩 유사도 검증

임베딩 벡터 유사도를 이용해 사용자 요청과 생성된 응답이 얼마나 일치하는 지 평가

9.4 데이터 로깅

⚒️ 데이터 로깅

모델의 입력과 출력 데이터를 로깅하여, 향후 분석과 성능 개선을 위해 저장하는 단계

📍 문제

LLM 애플리케이션의 경우 입력이 동일해도 출력이 달라질 수 있기 때문에 어떤 입력에서 어떤 출력을 반환했는지 반드시 기록해야 함

📍 장점

로그를 기록함으로써, 모델의 성능을 추적하고, 문제가 발생할 때 원인을 분석

📍 주요 로깅 도구

W&B (Weights & Biases), **MLflow**, **PromptLayer**

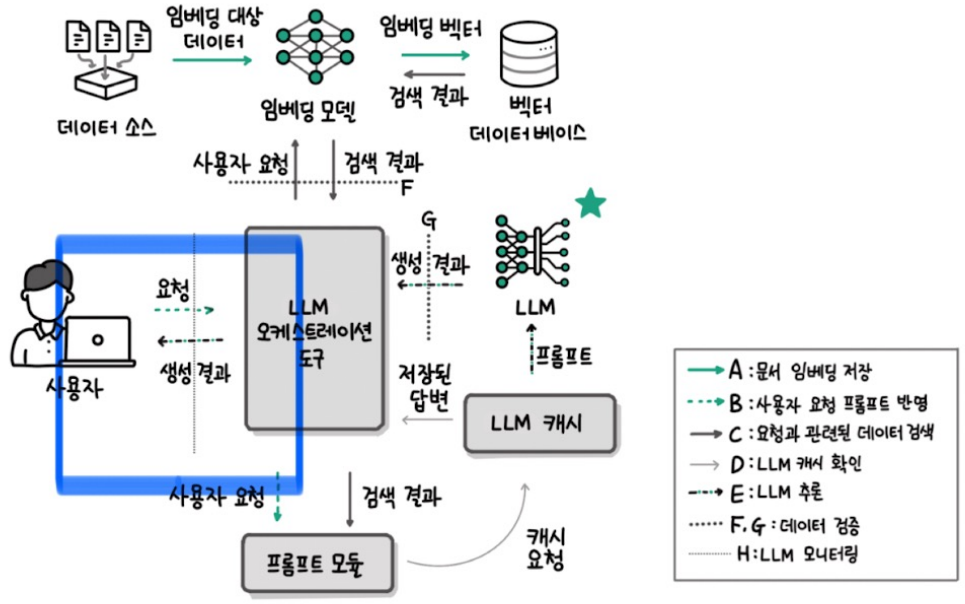


그림 9.1 LLM 애플리케이션 아키텍처

W&B

해당 도구를 사용하여 **로그 기록**을 진행하는 과정이 설명

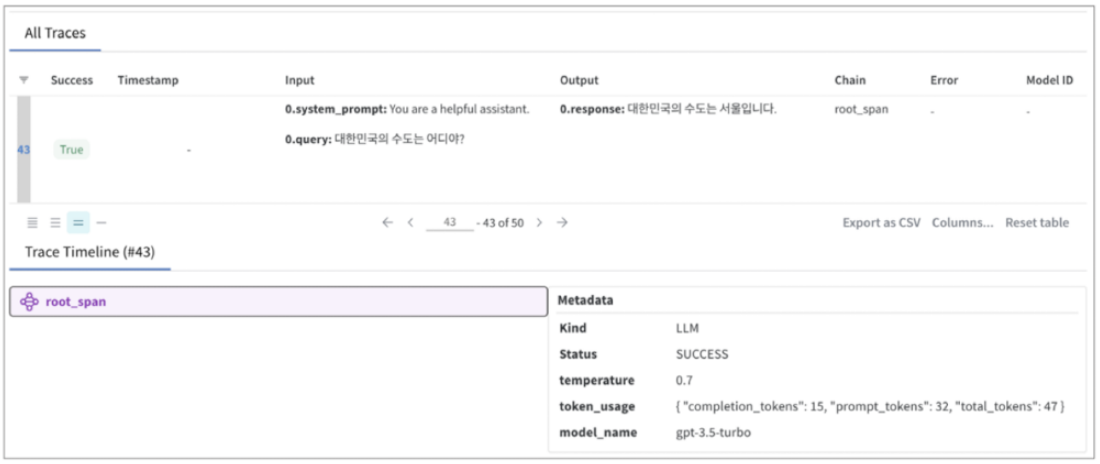


그림 9.16 W&B의 Trace 기능 스크린샷 화면

- Trace 기능을 제공하며, 사용자의 요청과 응답을 기록하고 확인
- 로그는 성능 분석 및 문제 해결 가능