# 차세대 RAG 벤치마크 러너 구축을 위한 OpenTelemetry 아키텍처 및 프레임워크 상호 운용성 심층 분석

## 서문: 생성형 AI 엔지니어링의 전환점과 관측 가능성의 필요성

2024년을 기점으로 대규모 언어 모델(LLM)을 활용한 애플리케이션 개발 패러다임은 단순한 프로토타이핑(Prototyping) 단계를 넘어, 엔터프라이즈급 프로덕션 환경에서의 안정성과 성능을 보장해야 하는 엔지니어링 단계로 진입하고 있습니다. 특히 외부 지식을 동적으로 검색하여 모델의 응답 생성에 활용하는 검색 증강 생성(RAG, Retrieval-Augmented Generation) 패턴은 할루시네이션(Hallucination)을 억제하고 최신 정보를 반영할 수 있다는 점에서 사실상의 표준 아키텍처로 자리 잡았습니다. 그러나 이러한 RAG 시스템은 확률적(Stochastic) 특성을 내재한 LLM, 비결정적(Non-deterministic) 검색 알고리즘, 그리고 복잡한 벡터 데이터베이스와 오케스트레이션 로직이 결합된 고도로 복잡한 분산 시스템입니다. 따라서, 이러한 시스템의 성능을 정량적으로 측정하고 지속적으로 개선하기 위한 '벤치마크 러너(Benchmark Runner)'의 구축은 선택이 아닌 필수가 되었습니다.

전통적인 소프트웨어 엔지니어링에서의 벤치마크가 주로 응답 속도(Latency)나 처리량(Throughput)과 같은 시스템 리소스 관점의 지표에 집중했다면, RAG 벤치마크는 답변의 정확성(Faithfulness), 관련성(Relevancy), 그리고 검색된 문맥의 정밀도(Context Precision)와 같은 의미론적(Semantic) 품질 평가를 포함해야 한다는 점에서 근본적인 차이가 있습니다. 이러한 다차원적인 평가를 수행하기 위해서는 애플리케이션의 실행 경로를 투명하게 추적하고, 각 단계에서 발생하는 데이터(입력 프롬프트, 검색된 청크, 생성된 답변, 사용된 토큰 수 등)를 손실 없이 포착할 수 있는 강력한 관측 가능성(Observability) 체계가 선행되어야 합니다.

OpenTelemetry(OTel)는 클라우드 네이티브 환경에서 분산 트레이싱, 메트릭, 로그를 수집하기 위한 벤더 중립적인 산업 표준으로, GenAI 시스템을 위한 시맨틱 규칙(Semantic Conventions)을 빠르게 확장하며 RAG 벤치마크를 위한 기술적 토대를 제공하고 있습니다. 그러나 LangChain이나 LlamaIndex와 같은 주요 LLM 오케스트레이션 프레임워크들은 각기 다른 철학으로 관측 가능성을 구현해 왔으며, 이로 인해 OTel 표준과의 통합 과정에서 심각한 호환성 문제와 기술적 부채가 발생하고 있습니다. 특히 Python의 비동기(Async) 런타임 환경에서의 컨텍스트 전파(Context Propagation) 실패, 과도기적인 표준(OpenInference 대 OTel GenAI) 간의 충돌, 그리고 수집된 데이터를 Ragas나 DeepEval과 같은 평가 프레임워크와 연동하는 과정에서의 데이터 정합성 문제는 벤치마크 러너 구축을 저해하는 핵심 난제들입니다.

본 보고서는 OpenTelemetry 스타일로 RAG 벤치마크 러너를 구축하고자 하는 아키텍트와 엔지니어를 위해, 현재 직면하고 있는 기술적 장벽을 심층적으로 분석하고, 이를 극복하기 위한 아키텍처 및 구현 전략을 포괄적으로 제시합니다. 2024년 말부터 2025년 초까지 급격하게 변화하고 있는 OTel GenAI 표준과 주요 라이브러리들의 최신 업데이트 현황을 바탕으로, 실무적으로 적용 가능하고 확장 가능한 '벤치마크 파이프라인' 설계 방안을 도출하는 데 그 목적이 있습니다.

## 1. 생성형 AI 관측 가능성 표준의 지형도: 혼돈과 수렴의 과도기

RAG 벤치마크 러너를 구축하기 위한 첫 번째 단계는 수집할 데이터의 '형식'과 '의미'를 정의하는 것입니다. 2025년 현재, LLM 관측 가능성 시장은 CNCF(Cloud Native Computing Foundation)가 주도하는 공식 **OpenTelemetry GenAI Semantic Conventions**와, AI 엔지니어링 커뮤니티(주로 Arize AI)를 중심으로 실용성을 강조하며 등장한 **OpenInference** 표준이 공존하며 경쟁 및 보완하는 관계에 있습니다. 이 두 표준 간의 차이를 이해하고 적절한 전략을 수립하는 것은 벤치마크 데이터의 일관성을 확보하는 데 결정적인 역할을 합니다.

### 1.1 OpenTelemetry GenAI Semantic Conventions (v1.38.0+)의 현황과 한계

OpenTelemetry 프로젝트는 2023년부터 GenAI 시스템을 위한 시맨틱 규칙을 정의하기 위해 노력해 왔으나, 해당 분야의 기술 발전 속도가 워낙 빠르다 보니 표준화 작업은 여전히 유동적입니다. 2025년 1월 기준으로 OTel GenAI Semantic Conventions는 '개발 중(Development)' 상태이며, 이는 언제든지 하위 호환성을 깨뜨리는 변경(Breaking Change)이 발생할 수 있음을 의미합니다.

가장 대표적인 변화는 LLM 호출을 추상화하는 방식입니다. 초기에는 llm.provider나 llm.request.type과 같은 속성을 사용했으나, v1.38.0 이후로는 gen\_ai.system (모델 제공자), gen\_ai.operation.name (작업 유형), gen\_ai.request.model (요청 모델) 등의 gen\_ai.\* 네임스페이스로 개편되고 있습니다.1 예를 들어, 텍스트 생성을 나타내는 작업 이름은 text\_completion에서 generate\_content 등으로 변경되거나, 채팅(Chat)과 완성(Completion)을 통합하려는 시도가 계속되고 있습니다.

#### 안정성(Stability) 이슈와 Opt-in 메커니즘

이러한 잦은 변경은 벤치마크 러너의 안정성을 해칠 수 있습니다. 이를 방지하기 위해 OTel 커뮤니티는 기존 계측(Instrumentation) 라이브러리들이 구버전(v1.36.0 이전)의 규약을 유지하되, 사용자가 명시적으로 선택할 때만 최신 실험적 규약을 방출하도록 권고하고 있습니다. 이는 OTEL\_SEMCONV\_STABILITY\_OPT\_IN 환경 변수를 통해 제어되며, 값으로 gen\_ai\_latest\_experimental을 설정해야만 최신 속성을 사용할 수 있게 됩니다.1

벤치마크 러너 설계 시 이는 중요한 고려 사항입니다. 타겟 애플리케이션(LangChain 또는 LlamaIndex)이 사용하는 OTel 라이브러리의 버전에 따라, 그리고 해당 라이브러리가 참조하는 opentelemetry-semantic-conventions 패키지의 버전에 따라 수집되는 트레이스 데이터의 키(Key) 값이 달라질 수 있기 때문입니다. 따라서 벤치마크 러너는 단일 스키마를 가정하기보다는, 다양한 버전의 OTel 속성을 내부적으로 정규화(Normalization)할 수 있는 유연한 어댑터 계층을 반드시 포함해야 합니다.

### 1.2 OpenInference: RAG 벤치마킹을 위한 실용주의적 대안

OpenTelemetry 공식 표준이 범용성과 합의(Consensus)에 초점을 맞추느라 진행이 더딘 반면, **OpenInference**는 RAG 시스템의 복잡한 내부 동작을 표현하기 위해 탄생한 실용주의적 표준입니다.4 Arize AI가 주도하고 있으며, LlamaIndex와 같은 프레임워크가 이를 기본 관측 가능성 포맷으로 채택하면서 사실상의 산업 표준(De facto standard) 중 하나로 자리 잡았습니다.

OpenInference가 RAG 벤치마크 러너에 적합한 이유는 **RAG 파이프라인의 각 구성 요소를 명시적으로 구분할 수 있는 풍부한 Span Kind를 제공**하기 때문입니다. OTel 표준은 기본적으로 CLIENT, SERVER, INTERNAL 등의 범용적인 Span Kind만 제공하며, GenAI 특화 속성은 Attribute로 처리합니다. 반면, OpenInference는 다음과 같이 AI 엔지니어링에 특화된 Span Kind를 정의합니다 5:

* **RETRIEVER:** 벡터 데이터베이스나 검색 엔진에서 문서를 조회하는 작업을 명시합니다. 벤치마크 시 Context Recall이나 Context Precision을 측정하기 위해 가장 중요한 Span입니다.
* **EMBEDDING:** 사용자 쿼리나 문서를 벡터로 변환하는 작업을 추적합니다. 임베딩 모델의 지연 시간과 비용을 분석하는 데 사용됩니다.
* **RERANKER:** 검색된 문서들을 재순위화(Re-ranking)하는 과정을 추적합니다. Cross-Encoder 모델 등의 성능을 평가할 때 필수적입니다.
* **CHAIN / AGENT:** 여러 단계의 LLM 호출이나 툴 사용을 포함하는 상위 논리적 단위를 나타냅니다.
* **TOOL:** 검색 도구, 계산기, API 호출 등 에이전트가 사용하는 외부 도구의 실행을 추적합니다.

또한, OpenInference는 입출력 데이터를 input.value, output.value와 같이 직관적인 속성에 JSON 문자열 형태로 저장하도록 규정합니다. 이는 복잡한 중첩 구조를 가진 GenAI 속성(gen\_ai.usage.input\_tokens 등)보다 파싱이 용이하여, 벤치마크 데이터셋 구축 시 유리한 점이 있습니다.6

### 1.3 표준 간의 충돌과 상호 운용성 전략

2025년 현재, 벤치마크 러너 구축의 가장 큰 딜레마는 이 두 표준 간의 간극입니다. LangChain은 자체적인 langsmith 스키마를 OTel로 래핑하여 내보내는 경향이 있고, LlamaIndex는 OpenInference를 따릅니다. 순수 OTel 라이브러리(OpenLLMetry 등)는 OTel GenAI 표준을 준수하려 노력합니다. 이로 인해 동일한 "LLM 호출"이라도 프레임워크에 따라 전혀 다른 속성 집합을 가질 수 있습니다.

| **속성 구분** | **OpenInference** | **OTel GenAI (Experimental)** | **LangSmith (via OTel)** |
| --- | --- | --- | --- |
| **LLM 호출 유형** | openinference.span.kind="LLM" | gen\_ai.operation.name="chat" | langsmith.run\_type="llm" |
| **입력 프롬프트** | input.value (JSON/String) | gen\_ai.prompt (Event Payload) | inputs (JSON String) |
| **출력 응답** | output.value (JSON/String) | gen\_ai.completion | outputs (JSON String) |
| **검색 문서** | document.content | gen\_ai.retrieval.documents (예정) | langsmith.run\_type="retriever"의 output |
| **모델 이름** | llm.model\_name | gen\_ai.request.model | llm\_model |

이러한 파편화를 해결하기 위해 최근 arizeai/openinference-genai와 같은 유틸리티 패키지가 등장하여 OTel GenAI 속성을 OpenInference 속성으로 변환(Polyfill)해주는 기능을 제공하고 있습니다.7 벤치마크 러너는 이러한 변환 로직을 내재화하여, 어떤 프레임워크에서 데이터가 수집되든 최종적으로는 \*\*단일화된 분석용 스키마(Canonical Schema)\*\*로 데이터를 적재해야 합니다. 본 보고서는 RAG 벤치마크의 특성상 Retrieval 단계의 상세 분석이 중요하므로, 내부적으로는 **OpenInference 스키마를 기준으로 데이터를 정규화**하는 전략을 제안합니다.

## 2. LangChain과의 호환성 심층 분석: 아키텍처의 진화와 비동기 전파의 난제

LangChain은 LLM 애플리케이션 개발의 사실상 표준 프레임워크로, 방대한 생태계를 자랑합니다. 그러나 그 복잡한 추상화 계층과 독자적인 실행 구조로 인해 OpenTelemetry와의 통합은 역사적으로 많은 어려움을 겪어왔습니다.

### 2.1 콜백(Callback) 시스템에서 네이티브 계측(Native Instrumentation)으로의 전환

초기 LangChain은 관측 가능성을 지원하기 위해 CallbackSystem을 도입했습니다. 개발자가 CallbackHandler를 구현하고 이를 체인(Chain)이나 모델(LLM)에 주입하면, 실행 시점에 on\_llm\_start, on\_chain\_end와 같은 메서드가 호출되는 방식이었습니다.8 많은 서드파티 모니터링 도구(Arize Phoenix, Langfuse 등)들이 이 방식을 통해 데이터를 수집했습니다.9

그러나 콜백 방식은 몇 가지 치명적인 한계를 가지고 있습니다. 첫째, \*\*침습적(Intrusive)\*\*입니다. 모든 코드에서 콜백 핸들러를 명시적으로 전달하거나 전역 설정(set\_global\_handler)을 해야 하므로, 벤치마크 코드와 애플리케이션 코드가 강하게 결합됩니다. 둘째, \*\*비표준(Non-standard)\*\*입니다. 콜백 이벤트는 LangChain 내부 객체를 전달하므로, 이를 OTel Span으로 변환하려면 복잡한 매핑 로직이 필요하며, 이 과정에서 정보의 손실이나 왜곡이 발생하기 쉽습니다.

이에 대응하여 LangChain은 2025년 3월, **LangSmith SDK에 완전한 End-to-End OpenTelemetry 지원**을 추가했습니다.10 이는 LangChain 내부 코드가 실행될 때, OTel API를 직접 호출하여 트레이스를 생성함을 의미합니다. 이제 개발자는 LANGSMITH\_OTEL\_ENABLED=true 환경 변수 설정만으로 별도의 코드 수정 없이 표준 OTel 트레이스를 생성할 수 있게 되었습니다.11 벤치마크 러너 구축 시, 구형 콜백 방식보다는 이 네이티브 계측 방식을 사용하는 것이 호환성과 유지보수 측면에서 훨씬 유리합니다.

### 2.2 Python 비동기(Async) 컨텍스트 전파(Context Propagation)의 기술적 난제

LangChain을 활용한 고성능 RAG 벤치마크 러너를 구축할 때, 엔지니어들이 가장 빈번하게 마주치며 해결하기 어려운 문제는 바로 **비동기 환경에서의 컨텍스트 전파 실패**입니다.12 이는 벤치마크 데이터의 무결성을 훼손하는 결정적인 요인입니다.

#### 문제의 본질: contextvars와 asyncio의 불협화음

OpenTelemetry Python SDK는 현재 실행 중인 Span의 정보(Trace ID, Span ID)를 저장하고 전파하기 위해 Python 표준 라이브러리인 contextvars를 사용합니다. contextvars는 스레드 로컬 스토리지(Thread Local Storage)의 비동기 버전으로, 코루틴(Coroutine) 간에 컨텍스트를 격리하고 전달하는 역할을 합니다.

문제는 Python 3.10 이하 버전이나, asyncio.create\_task, asyncio.gather 등을 사용하여 명시적으로 태스크를 분기할 때 발생합니다. asyncio 이벤트 루프가 새로운 태스크를 스케줄링할 때, 현재의 contextvars 컨텍스트를 자동으로 복사하여 하위 태스크에 전달해야 하는데, 특정 상황에서 이 연결고리가 끊어지는 현상이 발생합니다. 이로 인해 RAG 파이프라인의 하위 작업(예: 병렬 문서 검색, 비동기 LLM 호출)이 부모 트레이스(Root Trace)와 연결되지 않고 **'고아 트레이스(Orphaned Trace)'** 또는 \*\*'분리된 트레이스(Fragmented Trace)'\*\*로 기록됩니다.13

벤치마크 관점에서 이는 치명적입니다. 사용자의 질문(Root Span)과 그에 따른 검색 결과(Retriever Span)가 서로 다른 Trace ID를 가지게 되면, "어떤 질문에 대해 어떤 문서가 검색되었는지"를 파악할 수 없게 되어 Context Relevance와 같은 핵심 지표를 계산할 수 없게 됩니다.

#### 해결 방안 및 아키텍처적 대응 전략

1. 런타임 환경 업그레이드 (가장 강력한 해결책)

Python 3.11 이상 버전에서는 asyncio 모듈 내부적으로 Task 생성 시 contextvars를 복사하는 로직이 대폭 개선되었습니다 (PEP 678 등 참조). 따라서 벤치마크 러너의 실행 환경을 \*\*Python 3.11 이상(권장 3.12)\*\*으로 구성하는 것만으로도 대부분의 컨텍스트 전파 문제를 해결할 수 있습니다.12

2. 수동 컨텍스트 전파 (Manual Propagation) 패턴

레거시 환경이나 복잡한 비동기 패턴(커스텀 스레드 풀 등)을 사용해야 한다면, OTel API를 사용하여 명시적으로 컨텍스트를 주입해야 합니다. 다음은 이를 구현하는 코드 패턴입니다.14

Python

import asyncio  
from opentelemetry import context  
  
async def run\_benchmark\_task(chain, inputs):  
 # 현재 실행 컨텍스트(부모 Span 정보 포함)를 캡처  
 ctx = context.get\_current()  
   
 # 별도 태스크로 실행할 때 컨텍스트를 명시적으로 전달  
 return await asyncio.create\_task(run\_in\_context(ctx, chain, inputs))  
  
async def run\_in\_context(ctx, chain, inputs):  
 # 하위 태스크 내부에서 캡처된 컨텍스트를 활성화(Attach)  
 token = context.attach(ctx)  
 try:  
 return await chain.ainvoke(inputs)  
 finally:  
 # 작업 완료 후 컨텍스트 분리(Detach) - 메모리 누수 방지  
 context.detach(token)

3. LangSmith SDK의 헬퍼 활용

LangChain 팀은 이 문제를 인지하고, LangSmith SDK 내에 @traceable 데코레이터나 ContextThreadPoolExecutor와 같은 유틸리티를 제공합니다. 이들은 내부적으로 컨텍스트 전파 로직을 캡슐화하고 있어, 개발자가 복잡한 contextvars 조작 없이도 트레이스 연결성을 유지할 수 있게 돕습니다.12

### 2.3 LangChain 속성 매핑의 복잡성

LangChain의 Native OTel 계측은 표준 OTel 속성보다는 langsmith.\* 네임스페이스를 주로 사용합니다. 예를 들어, Span의 유형을 나타내는 속성은 langsmith.run\_type에 chain, llm, tool, retriever 등의 문자열 값으로 저장됩니다.11 이는 OpenInference의 openinference.span.kind와 개념적으로 대응되지만, 키(Key)와 값(Value)이 다릅니다.

벤치마크 러너는 이러한 이질적인 속성을 정규화해야 합니다. 수집 파이프라인(OpenTelemetry Collector의 transform 프로세서)이나 데이터 처리 단계에서 다음과 같은 매핑 로직을 적용해야 합니다.

* langsmith.run\_type == "retriever" $\rightarrow$ openinference.span.kind = "RETRIEVER"
* langsmith.run\_type == "llm" $\rightarrow$ openinference.span.kind = "LLM"
* inputs (JSON String) $\rightarrow$ 파싱 후 input.value

## 3. LlamaIndex와의 호환성 심층 분석: 설계 철학의 차이와 데이터 추출

LlamaIndex는 데이터 중심(Data-centric)의 프레임워크로, RAG 구현에 최적화되어 있습니다. 관측 가능성 측면에서도 LangChain과는 다른 접근 방식을 취하고 있어 이에 대한 이해가 필요합니다.

### 3.1 Instrumentation 기반 아키텍처로의 전환

LlamaIndex 역시 초기에는 CallbackManager를 통한 관측 가능성을 제공했으나, v0.10.20 이후부터는 **openinference-instrumentation-llama-index** 패키지를 통한 자동 계측(Auto-instrumentation)을 공식 권장하고 있습니다.15

이 방식은 Python의 데코레이터나 메타클래스 훅을 사용하는 것이 아니라, 런타임에 라이브러리의 핵심 함수들을 래핑(Wrapping)하는 방식으로 동작합니다. 이를 활성화하기 위해서는 다음과 같은 초기화 코드가 필요합니다.16

Python

from openinference.instrumentation.llama\_index import LlamaIndexInstrumentor  
from opentelemetry import trace  
from opentelemetry.sdk.trace import TracerProvider  
from opentelemetry.sdk.trace.export import SimpleSpanProcessor, OTLPSpanExporter  
  
# 1. Tracer Provider 설정 (메모리 또는 원격 엔드포인트)  
tracer\_provider = TracerProvider()  
tracer\_provider.add\_span\_processor(  
 SimpleSpanProcessor(OTLPSpanExporter(endpoint="http://localhost:4317"))  
)  
  
# 2. LlamaIndex 계측 활성화  
LlamaIndexInstrumentor().instrument(tracer\_provider=tracer\_provider)

이 방식의 가장 큰 장점은 **OpenInference 표준을 네이티브로 준수**한다는 점입니다. LlamaIndex 내부의 QueryEngine, Retriever, ResponseSynthesizer 컴포넌트들이 실행될 때, 자동으로 RETRIEVER, EMBEDDING, LLM 등의 Span Kind를 가진 Span이 생성됩니다. 이는 별도의 속성 변환 없이도 벤치마크 데이터를 즉시 활용할 수 있음을 의미합니다.

### 3.2 검색(Retrieval) 데이터의 구조적 추출

RAG 벤치마크의 핵심은 "LLM이 생성한 답변" 뿐만 아니라 "검색 엔진이 가져온 문서"를 평가하는 것입니다. LlamaIndex의 계측 라이브러리는 검색된 문서의 내용을 document.content, 문서 ID를 document.id, 메타데이터를 document.metadata (JSON 문자열), 그리고 검색 유사도 점수를 document.score라는 속성으로 Span에 기록합니다.5

벤치마크 러너는 RETRIEVER Span을 감지했을 때, 이러한 속성들을 추출하여 Ragas의 contexts 필드(문자열 리스트)로 변환해야 합니다. 이때 주의할 점은 LlamaIndex가 여러 단계의 검색(예: 1차 검색 후 Re-ranking)을 수행할 경우, RETRIEVER Span 하위에 RERANKER Span이 중첩되거나 연속될 수 있다는 점입니다. 벤치마크 러너는 최종적으로 LLM에게 전달된 컨텍스트가 무엇인지 정확히 식별하기 위해 트레이스 트리(Trace Tree)를 순회(Traverse)하는 로직을 갖추어야 합니다.

## 4. RAG 벤치마크 러너 아키텍처 설계 및 구현 전략

앞서 분석한 프레임워크별 특성과 호환성 이슈를 종합하여, 확장 가능하고 견고한 RAG 벤치마크 러너의 아키텍처를 제안합니다. 이 아키텍처는 **'수집은 범용적으로(Generic Collection), 분석은 정규화하여(Normalized Analysis)'** 수행하는 원칙을 따릅니다.

### 4.1 전체 아키텍처 개요

벤치마크 러너는 논리적으로 **1) 실행 및 수집(Execution & Collection)**, **2) 데이터 정규화 및 어댑터(Normalization & Adapter)**, **3) 평가 및 분석(Evaluation & Analysis)** 세 계층으로 구성됩니다.

#### Layer 1: Execution & Collection

* **Target Application:** LangChain 또는 LlamaIndex로 구현된 RAG 파이프라인입니다.
* **Instrumentation:** 각 프레임워크에 맞는 OTel/OpenInference 계측 라이브러리를 주입합니다.
* **Trace Context Manager:** 비동기 실행 시 컨텍스트 전파를 보장하는 래퍼(Wrapper) 모듈입니다.
* **OTLP Exporter:** 생성된 트레이스 데이터를 Protobuf 형식으로 내보냅니다. 로컬 테스트 시에는 메모리 내 버퍼(InMemorySpanExporter)를, 대규모 테스트 시에는 OpenTelemetry Collector를 사용할 수 있습니다.

#### Layer 2: Normalization & Adapter (The Core)

이 계층은 이종 프레임워크에서 수집된 트레이스 데이터를 단일한 '벤치마크 데이터셋' 포맷으로 변환하는 핵심 역할을 수행합니다.

* **Trace Reassembler:** 스트림 형태로 들어오는 Span들을 TraceID 기준으로 묶어 하나의 온전한 트레이스 객체로 조립합니다. 스트리밍 응답의 경우, 모든 청크(Chunk)가 수신된 후 완료된 텍스트를 재구성하는 로직이 필요합니다.
* **Attribute Mapper:** LangChain의 langsmith 속성, OTel GenAI 속성 등을 OpenInference 표준 속성으로 변환합니다. arizeai/openinference-genai와 같은 라이브러리의 로직을 Python으로 구현하여 적용합니다.
* **Dataset Builder:** 트레이스에서 question (Root Span의 Input), answer (Root Span의 Output), contexts (Retriever Span의 Documents), ground\_truth (벤치마크 데이터셋에서 매핑)를 추출하여 Pandas DataFrame으로 변환합니다.17

#### Layer 3: Evaluation & Analysis

* **Offline Evaluator:** 변환된 DataFrame을 Ragas 또는 DeepEval 라이브러리에 입력하여 평가를 수행합니다.
* **Metric Store:** 계산된 점수(Faithfulness, Answer Relevancy 등)를 다시 OTel 메트릭으로 변환하여 저장하거나, 리포트로 출력합니다.

### 4.2 오프라인 평가(Offline Evaluation) 패턴의 도입

RAG 벤치마크 러너는 실시간 모니터링 도구가 아닙니다. 벤치마크 실행 중 실시간으로 평가(LLM-as-a-judge)를 수행하면, 평가를 수행하는 LLM 호출의 지연 시간과 비용이 전체 벤치마크 성능에 영향을 미칠 수 있습니다. 따라서 **비동기식 오프라인 평가(Asynchronous Offline Evaluation)** 패턴이 권장됩니다.19

1. **Phase 1 (Execution):** 벤치마크 데이터셋(질문 100개 등)에 대해 RAG 파이프라인을 실행합니다. 이때는 오직 트레이스 데이터만 수집하고 저장합니다. (평가 로직 실행 X)
2. **Phase 2 (Extraction):** 저장된 트레이스에서 평가에 필요한 필드(질문, 답변, 검색 문서)를 추출하여 데이터셋을 구성합니다.
3. **Phase 3 (Evaluation):** 추출된 데이터셋에 대해 병렬로 Ragas/DeepEval 평가를 수행합니다.

이 방식은 벤치마크 실행 속도를 극대화하고, 평가 로직이 애플리케이션 로직에 영향을 주지 않도록 격리(Isolation)합니다. 또한, 수집된 트레이스 데이터를 재사용하여 다양한 평가 지표(Metric)를 실험해 볼 수 있는 유연성을 제공합니다.

## 5. 평가 프레임워크(Ragas, DeepEval)와의 연동 전략

수집된 OTel 트레이스 데이터를 실제 평가 점수로 변환하기 위해서는 Ragas 및 DeepEval과의 정교한 연동이 필요합니다.

### 5.1 Ragas와의 연동: DataFrame을 통한 브리지

Ragas는 기본적으로 datasets.Dataset 객체를 입력으로 받습니다. OTel 트레이스에서 추출한 데이터를 Ragas가 요구하는 컬럼(question, answer, contexts, ground\_truth)을 가진 Pandas DataFrame으로 변환하는 것이 연동의 핵심입니다.21

Python

import pandas as pd  
from datasets import Dataset  
from ragas import evaluate  
from ragas.metrics import faithfulness, answer\_relevancy  
  
def evaluate\_traces\_with\_ragas(traces):  
 # 1. 트레이스 데이터를 리스트 딕셔너리로 변환  
 data\_rows =  
 for trace in traces:  
 root\_span = find\_root\_span(trace)  
 retriever\_spans = find\_retriever\_spans(trace)  
   
 contexts =  
 for span in retriever\_spans:  
 # OpenInference 속성에서 문서 내용 추출  
 if "document.content" in span.attributes:  
 contexts.append(span.attributes["document.content"])  
   
 data\_rows.append({  
 "question": root\_span.attributes.get("input.value"),  
 "answer": root\_span.attributes.get("output.value"),  
 "contexts": contexts,  
 "ground\_truth": root\_span.attributes.get("ground\_truth") # 별도 주입 필요  
 })  
  
 # 2. DataFrame 및 Dataset 변환  
 df = pd.DataFrame(data\_rows)  
 ragas\_dataset = Dataset.from\_pandas(df)  
   
 # 3. Ragas 평가 실행  
 results = evaluate(ragas\_dataset, metrics=[faithfulness, answer\_relevancy])  
 return results

여기서 중요한 점은 ground\_truth의 관리입니다. OTel 트레이스에는 정답 데이터가 포함되어 있지 않으므로, 벤치마크 실행 시 Trace Context의 Baggage나 커스텀 속성에 ground\_truth를 주입하거나, 평가 단계에서 Trace ID나 Question을 키(Key)로 하여 원본 데이터셋과 조인(Join)하는 로직이 필요합니다.

### 5.2 DeepEval와의 연동: @observe 데코레이터와 수동 연동

DeepEval은 자체적인 @observe 데코레이터를 통한 트레이싱을 강력하게 지원하지만 18, 이미 OTel로 계측된 데이터가 있는 경우 이를 DeepEval의 테스트 케이스(LLMTestCase)로 변환하여 평가할 수 있습니다.

DeepEval은 평가 결과를 Confident AI 클라우드 플랫폼으로 전송하여 시각화하는 기능을 제공하므로, 벤치마크 러너가 OTel 트레이스를 DeepEval 객체로 변환하여 전송하는 '싱크(Sink)' 역할을 수행할 수 있습니다. 이 경우, OTel의 Span ID와 DeepEval의 Trace ID를 일치시켜 시스템 간의 추적성을 유지하는 것이 중요합니다.

## 6. 결론 및 향후 전망

OpenTelemetry 기반의 RAG 벤치마크 러너 구축은 단순한 기술적 통합을 넘어, AI 애플리케이션의 신뢰성을 확보하기 위한 인프라 구축 작업입니다.

**본 보고서의 핵심 제언:**

1. **표준 선택의 실용성:** 현재 시점(2025년)에서는 OTel GenAI 표준의 과도기적 성격을 고려할 때, **OpenInference 표준을 내부 데이터 모델로 채택**하는 것이 RAG 벤치마킹에 가장 유리합니다.
2. **LangChain 호환성 확보:** 비동기 컨텍스트 전파 문제는 벤치마크 데이터의 신뢰성을 위협하는 가장 큰 위험 요소입니다. **Python 3.11+ 환경**을 강제하거나, 철저한 수동 전파 로직을 구현해야 합니다.
3. **데이터 정규화 계층 필수:** LangChain과 LlamaIndex, 그리고 다양한 OTel 라이브러리가 혼재된 환경에서는 **강력한 속성 매핑(Attribute Mapping) 및 정규화 레이어**가 벤치마크 러너의 핵심 경쟁력이 됩니다.
4. **평가와 실행의 분리:** OTel 트레이스를 매개체로 하여 실행(Execution)과 평가(Evaluation)를 비동기적으로 분리하는 **오프라인 평가 아키텍처**를 통해 성능과 유연성을 동시에 확보해야 합니다.

향후 OTel GenAI 시맨틱 규칙이 안정화(Stable) 단계에 도달하면, 주요 프레임워크들이 이를 네이티브로 지원하게 될 것입니다. 벤치마크 러너는 이러한 변화에 유연하게 대응할 수 있도록, 속성 매핑 로직을 코드에서 분리하여 설정 파일(Configuration)이나 플러그인 형태로 관리하는 것이 바람직합니다. 이것이 지속 가능한 RAG 벤치마크 시스템을 구축하는 최선의 전략입니다.

### [참고 문헌 및 데이터 출처]

* **OTel GenAI Standard:** 1
* **OpenInference:** 4
* **LangChain Integration:** 10
* **LlamaIndex Integration:** 15
* **Ragas/DeepEval:** 17

#### 참고 자료

1. Semantic conventions for generative AI systems - OpenTelemetry, 12월 30, 2025에 액세스, <https://opentelemetry.io/docs/specs/semconv/gen-ai/>
2. Semantic conventions for generative client AI spans - OpenTelemetry, 12월 30, 2025에 액세스, <https://opentelemetry.io/docs/specs/semconv/gen-ai/gen-ai-spans/>
3. Semantic conventions for generative AI metrics - OpenTelemetry, 12월 30, 2025에 액세스, <https://opentelemetry.io/docs/specs/semconv/gen-ai/gen-ai-metrics/>
4. What is openinference - Arize AX Docs, 12월 30, 2025에 액세스, <https://arize.com/docs/ax/observe/tracing-concepts/what-is-openinference>
5. Semantic Conventions | openinference - GitHub Pages, 12월 30, 2025에 액세스, <https://arize-ai.github.io/openinference/spec/semantic_conventions.html>
6. Openinference Semantic Conventions - Arize AX Docs, 12월 30, 2025에 액세스, <https://arize.com/docs/ax/observe/tracing-concepts/openinference-semantic-conventions>
7. arizeai/openinference-genai - NPM, 12월 30, 2025에 액세스, <https://www.npmjs.com/package/%40arizeai%2Fopeninference-genai>
8. Callbacks Improvements - LangChain Blog, 12월 30, 2025에 액세스, <https://blog.langchain.com/callbacks/>
9. OpenTelemetry auto-instrumentation traces appearing in Langfuse v3 alongside LangChain callbacks #9136 - GitHub, 12월 30, 2025에 액세스, <https://github.com/orgs/langfuse/discussions/9136>
10. Introducing End-to-End OpenTelemetry Support in LangSmith - LangChain Blog, 12월 30, 2025에 액세스, <https://blog.langchain.com/end-to-end-opentelemetry-langsmith/>
11. Trace with OpenTelemetry - Docs by LangChain, 12월 30, 2025에 액세스, <https://docs.langchain.com/langsmith/trace-with-opentelemetry>
12. Troubleshoot trace nesting - Docs by LangChain, 12월 30, 2025에 액세스, <https://docs.langchain.com/langsmith/nest-traces>
13. bug: "Failed to detach context" error using async with langchain CallbackHandler · Issue #8780 - GitHub, 12월 30, 2025에 액세스, <https://github.com/langfuse/langfuse/issues/8780>
14. Troubleshooting LangChain/LangGraph Traces: Common Issues and Fixes | Last9, 12월 30, 2025에 액세스, <https://last9.io/blog/troubleshooting-langchain-langgraph-traces-issues-and-fixes/>
15. Observability | LlamaIndex Python Documentation, 12월 30, 2025에 액세스, <https://developers.llamaindex.ai/python/framework/module_guides/observability/>
16. OpenInference LlamaIndex Instrumentation - GitHub Pages, 12월 30, 2025에 액세스, <https://arize-ai.github.io/openinference/python/instrumentation/openinference-instrumentation-llama-index/>
17. Evaluating and Analyzing Your RAG Pipeline with Ragas - Arize Phoenix, 12월 30, 2025에 액세스, <https://phoenix.arize.com/evaluating-and-analyzing-your-rag-pipeline-with-ragas-and-phoenix/>
18. LLM Tracing | DeepEval - The Open-Source LLM Evaluation Framework, 12월 30, 2025에 액세스, <https://deepeval.com/docs/evaluation-llm-tracing>
19. AI Agent Evaluation | DeepEval - The Open-Source LLM Evaluation Framework, 12월 30, 2025에 액세스, <https://deepeval.com/guides/guides-ai-agent-evaluation>
20. Tracing and Evaluating Your LLM Applications with MyScale Telemetry: An Open-Source Alternative to LangSmith - Medium, 12월 30, 2025에 액세스, <https://medium.com/@myscale/tracing-and-evaluating-your-llm-applications-with-myscale-telemetry-an-open-source-alternative-to-33543cdea896>
21. Evaluation of RAG pipelines with Ragas - Langfuse, 12월 30, 2025에 액세스, <https://langfuse.com/guides/cookbook/evaluation_of_rag_with_ragas>
22. Evaluating Using Your Test Set - Ragas, 12월 30, 2025에 액세스, <https://docs.ragas.io/en/v0.1.21/getstarted/evaluation.html>
23. Evolving OpenTelemetry's Stabilization and Release Practices, 12월 30, 2025에 액세스, <https://opentelemetry.io/blog/2025/stability-proposal-announcement/>
24. openinference-instrumentation-llama-index - PyPI, 12월 30, 2025에 액세스, <https://pypi.org/project/openinference-instrumentation-llama-index/>
25. Ragas Evaluations - Datadog Docs, 12월 30, 2025에 액세스, <https://docs.datadoghq.com/llm_observability/evaluations/ragas_evaluations/>