ChatGPT

SafeGuard AI 산업안전 시스템 로컬 프로토타입 구현

Octank Chemical사의 **SafeGuard AI** 시스템은 AWS 클라우드에서 IoT와 AI를 활용해 산업 현장의 위험을 실시간 감지/대응하는 플랫폼입니다 [1]. 여기서는 실제 AWS 서비스를 사용하지 않고도 비슷한 기능을 시현하기 위해 **로컬 환경에서 오픈소스 도구들로 구성한 프로토타입 아키텍처**를 제안합니다. 이 프로토타입은 **Retrieval-Augmented Generation (RAG)** 기법을 활용하여 **카메라 이벤트**와 **작업 지침서 문서**를 함께 분석하고 대응 방안을 생성하며, **이메일 알림**과 **대시보드**를 통해 위험 상황을 전달합니다. 전체 시스템은 end-to-end로 구성되며, 개발 기간 4일 이내에 구현할 수 있을 만큼 경량화된 구조입니다.

아키텍처 개요

로컬 프로토타입 아키텍처는 다음과 같은 흐름으로 구성됩니다:

1. **이벤트 발생 (시뮬레이션)** – 공장의 센서나 AI 카메라에서 발생한 이벤트를 실제 장비 대신 **샘플 JSON 파일**로 모사합니다. 예를 들어, `{ "camera_id": "CAM1", "event_type": "fire", "details": "Storage area smoke detected" }` 와 같은 이벤트 JSON이 주어집니다.
2. **이벤트 수신 및 처리** – 로컬 Python 스크립트가 주기적으로 JSON 파일을 읽거나 이벤트 큐를 확인하여 새로운 이벤트를 포착합니다. 이는 AWS IoT Core와 Lambda 등을 통해 이벤트를 받아 처리하는 것을 로컬에서 **스크립트로 대체**한 것입니다 [2]. 이벤트를 받으면 위험 유형(예: 화재, 침입 등)에 따라 간략한 **프롬프트 질의어**를 생성합니다.
3. **백터 DB 문서 검색 (RAG)** – **벡터 데이터베이스**에 미리 저장된 안전 지침서 **문서 임베딩**들에 대해, 이벤트에 맞는 관련 문서를 검색합니다. SafeGuard AI 원본 시스템에서는 작업 지시서 PDF를 업로드하면 **텍스트를 추출하여 벡터 DB(OpenSearch)**에 저장해 두고 필요한 내용을 생성형 AI가 검색 참조했습니다 [3] [4]. 이를 위해 로컬에서는 오픈소스 **벡터 DB** (예: ChromaDB, FAISS 등)와 **임베딩 모델**(예: SBERT)을 사용합니다 [5]. 이벤트 내용을 질의로 변환해 벡터 DB에서 **유사도가 가장 높은 문서 조각**을 가져옵니다.
4. **생성형 AI 분석 및 대응안 생성** – 검색된 문서 내용(컨텍스트)과 이벤트 세부 정보를 결합하여 **LLM**에 프롬프트를 구성합니다. LLM은 해당 상황의 **위험도 평가와 대응 조치 권고** 문장을 생성합니다 [6]. 원본 SafeGuard AI에서는 이 **생성형 AI 분석**을 AWS Bedrock이나 SageMaker 상의 모델로 수행했을 것입니다. 여기서는 오픈소스 LLM(예: **Llama 2 7B**, **Flan-T5** 등)을 로컬 환경에서 구동하여 대체합니다.
5. **알림 및 대시보드** – LLM이 생성한 대응 조치 결과는 **위험 알림 이메일**로 전송되고, 로컬 **대시보드 UI**에도 표시됩니다. 원 시스템에서는 위험 상황 발생 시 SMS로 경보를 전송했습니다 [7]. 우리는 이를 **이메일 전송 (SMTP)**으로 구현하며, AWS SES/SNS를 쓰지 않고도 경보를 받을 수 있게 합니다. 또한 사건 발생 이력과 AI 분석 결과를 볼 수 있는 **간단한 웹 대시보드**를 제공합니다 (원본 시스템에서도 대시보드에서 카메라 상태, 이벤트 이력, AI 분석 리포트를 확인했습니다 [8]).

이상의 구성으로 로컬 PC 한 대에서 **이벤트 시뮬레이션 → AI분석 → 알림/표시**까지 일련의 흐름이 자체적으로 완결됩니다. 아래에서는 각 구성 요소에 사용된 오픈소스 도구와 그것이 **AWS의 어떤 서비스 역할을 대신하는지**를 정리하고, 주요 기능별 **코드 예시**를 보여드립니다.

오픈소스 구성 요소 및 AWS 서비스 대응

- **이벤트 수집 및 처리** – 로컬 Python 스크립트 또는 스케줄러로 구현된 이벤트 처리 파이프라인. (AWS 대응: IoT Core + Lambda) 카메라 장치로부터 이벤트를 수신하는 AWS IoT Core 및 Lambda 함수를 **단일 파이썬 프로그램**으로 대체합니다 [2]. 예를 들어, 일정 주기로 JSON 파일을 읽어 이벤트를 처리합니다.
- **문서 저장 및 임베딩 벡터DB** – 안전 수칙 문서를 저장하는 벡터 데이터베이스로 **ChromaDB** 등을 사용. (AWS 대응: Amazon OpenSearch Service) 원본 시스템에서는 PDF에서 추출한 텍스트 임베딩을 **OpenSearch의**

벡터 인덱스에 저장했습니다 [4]. ChromaDB는 OpenSearch의 벡터 검색 기능을 대체하며, 메모리 내 또는 로컬 파일로도 동작하는 경량 DB입니다 [5].
또한 **문서 임베딩 모델**로 **SentenceTransformer**(예: `all-MiniLM-L6-v2`)를 사용하여 텍스트를 벡터화합니다. 이는 AWS에서 SageMaker JumpStart나 Bedrock에 호스팅된 임베딩 모델을 사용하는 것을 로컬에서 구현한 것입니다.

- **생성형 AI 모델(LLM)** – 오픈소스 **LLM**을 통해 이벤트 상황 분석 및 대응안 생성. (AWS 대응: Amazon Bedrock 또는 SageMaker의 LLM 서비스) SafeGuard AI의 중앙 분석에는 대형 언어 모델 기반 생성 AI가 활용되었습니다 [6]. 본 프로토타입에서는 예시로 Meta의 **Llama 2 7B** 모델을 로컬 GPU/CPU에서 구동하거나, 사전학습된 **FLAN-T5** 모델을 활용합니다. 이를 통해 AWS의 Bedrock/Titan 모델 API 없이도 로컬에서 추론이 가능합니다.
- **알림 서비스** – **이메일 전송 기능**으로 구현. (AWS 대응: Amazon SNS/SES) 위험 이벤트 발생 시 AWS에서는 SMS 등을 통해 알림이 발송되었지만 [7], 로컬 환경에서는 Python `smtplib`을 이용해 **SMTP 이메일**을 보냅니다. 예를 들어 Gmail SMTP 서버나 로컬 메일서버(MailHog 등)를 통해 알림 메일을 발송하여 AWS SNS/SMS를 대체합니다.
- **대시보드 UI** – **간이 웹 대시보드**로 이벤트 이력과 AI 분석 결과를 시각화. (AWS 대응: Amazon QuickSight 또는 커스텀 웹앱) 웹 프레임워크 **Streamlit**을 사용하면 빠르게 대시보드를 구축할 수 있습니다. 이를 통해 AWS 환경에서 사용자들이 보게 될 대시보드(예: CloudWatch 대시보드나 custom 웹 애플리케이션) 역할을 로컬에서 수행합니다. 실시간까지는 아니더라도, 이벤트 처리 스크립트와 연동하여 새 이벤트가 발생하면 대시보드에 반영되도록 구현합니다.

以上의 오픈소스 구성 요소들을 연계함으로써, AWS 클라우드 없이도 SafeGuard AI와 유사한 기능의 **로컬 통합 시스템**을 구현할 수 있습니다. 다음 절에서는 이러한 구성 요소별 코드 구현 예시를 보여줍니다.

주요 구성 요소별 구현 예시

1. 이벤트 수신 및 처리 (JSON 이벤트 시뮬레이션)

이 부분에서는 **샘플 이벤트 JSON 파일**을 읽어 이벤트 객체를 생성하고 처리합니다. Python 코드로 폴링(polling) 또는 파일 시스템 감시를 통해 새로운 이벤트를 감지했다고 가정합니다. 아래 코드는 단순화를 위해 하나의 JSON 파일을 읽어 이벤트를 파싱한 뒤, 이후 단계를 호출하는 예시입니다:

```
import json

# 이벤트 JSON 파일 읽기 (예: events/event1.json)
with open("events/event1.json", "r") as f:
    event = json.load(f)

# 이벤트 정보 추출
event_type = event.get("event_type")    # 예: "fire"
details = event.get("details")          # 예: "Storage area smoke detected"
camera_id = event.get("camera_id")      # 예: "CAM1"

print(f"[이벤트 발생] 타입: {event_type}, 상세: {details}")

# (다음 단계) 이벤트 내용을 기반으로 벡터DB 질의 및 LLM 분석 함수 호출
response = analyze_event(event_type, details)
```

위 코드에서는 `event_type`에 따라 적절한 후속 조치를 취할 수 있습니다. 실제 구현에서는 `analyze_event()` 함수를 정의하여 **벡터 검색**과 **LLM 추론**을 수행하도록 합니다. 이벤트 발생 감지는 파일 대신 메시지 큐나 Webhook으로도 대체 가능하며, AWS IoT Core나 EventBridge 역할을 할 수 있습니다.

2. 문서 임베딩 및 벡터 데이터베이스 구축

안전 작업지침서 등의 도메인 지식 문서를 미리 임베딩하여 **벡터 데이터베이스**에 저장해둡니다. 이렇게 해야 이후에 이벤트 문맥과 가장 관련 높은 지침서 조각을 검색해 LLM에 제공할 수 있습니다 [3]. 다음은 예시 문서(PDF에서 추출된 텍스트라고 가정)의 리스트를 임베딩하고 ChromaDB에 적재하는 코드 예시입니다:

```
from sentence_transformers import SentenceTransformer
import chromadb

# 예시 문서 텍스트들 (이미 적절히 분할된 문장/단락 단위)
documents = [
    "화재 발생 시 작업자는 즉시 비상 경보를 발령하고 대피 절차를 따르십시오.",
    "위험 구역에 무단진입 시 경고 방송 및 보안 요원을 호출합니다.",
    # ... (추가 안전 지침 문장들)
]

# 1. 임베딩 모델 초기화 (Sentence-BERT 등 사용)
embedding_model = SentenceTransformer('sentence-transformers/all-MiniLM-L6-v2')
doc_embeddings = embedding_model.encode(documents)  # 문서별 벡터 생성

# 2. ChromaDB 클라이언트 생성 및 컬렉션 초기화
client = chromadb.Client()
collection = client.create_collection(name="safety_docs")

# 3. 문서와 임베딩을 벡터DB에 추가
collection.add(
    documents=documents,
    embeddings=doc_embeddings,
    ids=[f"doc{i}" for i in range(len(documents))]
)
print(f"{len(documents)}개의 문서 임베딩을 벡터DB에 저장했습니다.")
```

위 예에서는 **SentenceTransformer** 모델로 각 문장을 384차원 임베딩 벡터로 변환하고, ChromaDB의 `collection`에 벡터와 원문을 저장했습니다. 실제 PDF 문서가 있다면 Python의 PyPDF2/PyMuPDF 등을 이용해 **텍스트 추출** 후 문단 단위로 쪼개어 위 작업을 수행합니다. (AWS Textract + OpenSearch 구성을 로컬에서는 PyPDF + ChromaDB로 대체한 것입니다.)

임베딩 저장이 완료되면, **이벤트 분석 시점**에 해당 벡터DB에서 관련 문서를 검색할 수 있습니다. 예를 들어, 이벤트 타입이나 상세 설명을 임베딩하여 **유사도 검색**으로 상위 문서를 찾습니다:

```
# 새 이벤트 쿼리 임베딩 생성 (예: "fire" 키워드 + 세부 내용으로 질의 벡터 생성)
query = f"Relevant safety guidance for {event_type}."  # 영문 질의 예시
query_embedding = embedding_model.encode([query])

# 임베딩으로 유사한 문서 검색 (예: 상위 2개 결과)
results = collection.query(query_embeddings=query_embedding, n_results=2)
relevant_texts = results.get("documents", [])
print("관련 문서 조각:", relevant_texts)
```

위 `collection.query()` 결과로 이벤트와 가장 **코사인 유사도**가 높은 지침서 문서 조각들을 얻습니다. 그런 다음 `relevant_texts` 리스트를 LLM 프롬프트의 컨텍스트로 활용하게 됩니다. (ChromaDB 외에 Milvus, FAISS, Weaviate 등 다른 벡터DB도 사용할 수 있으며, AWS에서는 OpenSearch 벡터 인덱스가 동일한 역할을 합니다 [5].)

3. 생성형 AI를 통한 상황 분석 및 대응안 생성 (LLM 추론)

이제 벡터DB 검색으로 얻은 **관련 지침서 내용**과 **이벤트 세부정보**를 결합하여 **프롬프트**를 만들고 LLM에 전달합니다. LLM은 이 컨텍스트를 참고하여 적절한 대응 조치를 생성합니다 [6]. 오픈소스 LLM은 예시로 **HuggingFace Transformers**를 사용해 로드합니다. 작은 모델(Fine-tuned T5 등)을 사용하거나, 여력이 된다면 Llama2 같은 대형 모델을 로드할 수 있습니다. 아래는 HuggingFace의 `pipeline`으로 **FLAN-T5** 모델을 불러와 질의에 답변하는 예시입니다:

```
from transformers import pipeline

# 1. 사전학습된 생성 모델 불러오기 (예: FLAN-T5 기반)
generator = pipeline("text2text-generation", model="google/flan-t5-base")

# 2. 이벤트와 검색 문서를 포함한 프롬프트 구성
event_desc = f"이벤트 유형: {event_type}\n이벤트 상세: {details}"
context = "\n".join(relevant_texts[0])  # 상위 결과 문서 텍스트 (리스트 -> 문자열)
prompt = (
    f"{event_desc}\n\n"
    f"관련 안전 지침서 내용:\n{context}\n\n"
    f"질문: 위 상황에서 어떤 대응 조치를 취해야 하나요?"
)

# 3. LLM에 프롬프트 전달하여 답변 생성
result = generator(prompt, max_length=100)
recommendation = result[0]["generated_text"]
print("AI 권고사항:", recommendation)
```

위 코드에서 `prompt`에는 **이벤트 정보**와 **관련 지침서 발췌문**을 포함시켰습니다. 예를 들어, 화재(`event_type="fire"`) 발생 이벤트라면 해당 문맥을 담은 지침서 문구("화재 발생 시... 대피 절차를 따르십시오.")가 `context`로 들어갑니다. 그러면 FLAN-T5 모델은 이 정보를 활용하여 "즉시 비상벨을 울리고 인근 작업자들에게 대피를 지시하십시오..." 와 같은 대응 조치를 생성할 것입니다. 이러한 RAG 기반 접근을 통해 LLM이 **훈련되지 않은 사내 지식**도 참고하여 더 정확한 답변을 내도록 합니다 [3].

만약 **Llama 2** 등 로컬 LLM을 사용한다면, `transformers`의 `AutoModelForCausalLM` + `generate()` 형태나 `llama-cpp-python` 등을 활용할 수 있습니다. 또는 `LangChain` 라이브러리를 쓰면 벡터DB와 LLM을 체인으로 쉽게 묶을 수 있지만, 본 예시는 의존성을 줄이기 위해 기본 라이브러리만 사용했습니다.

4. 위험 알림 이메일 전송

LLM이 산출한 대응 조치 결과(및 이벤트 정보)를 관련 담당자에게 **이메일로 전송**하여 알립니다. Python의 내장 `smtplib`와 MIME 라이브러리를 이용해 간단히 구현할 수 있습니다. (실제 이메일 발송에는 Gmail SMTP 등의 계정 정보 설정이 필요합니다.) 예시 코드는 아래와 같습니다:

```
import smtplib
from email.mime.text import MIMEText

# 이메일 콘텐츠 구성
subject = f"[경보] {event_type.upper()} 이벤트 발생 - 즉각 조치 필요"
body = f"카메라 {camera_id}에서 위험 이벤트 발생: {details}\n\nAI 권고 대응: {recommendation}"
msg = MIMEText(body)
msg["Subject"] = subject
msg["From"] = "alert@safeguard.local"
msg["To"] = "manager@example.com"

# SMTP를 통해 이메일 전송 (예: 로컬 SMTP 서버 또는 Gmail SMTP)
with smtplib.SMTP("smtp.gmail.com", 587) as server:
    server.starttls()
    server.login("your_email@gmail.com", "앱비밀번호")
    server.send_message(msg)

print("이메일 알림 발송 완료!")
```

위 코드에서는 **위험 내용과 AI 대응 방안**을 포함한 이메일 본문을 작성한 뒤, SMTP 서버를 통해 수신자(`manager@example.com`)에게 전송합니다. AWS 환경의 Amazon SES 또는 SNS+SMS 조합 대신, 로컬에서 이메일을 사용함으로써 비슷한 알림 기능을 수행합니다. (테스트 시에는 실제 메일을 보내지 않고 콘솔에 출력하거나, 로컬 메일서버인 **MailHog** 등을 사용하면 개발 단계에서 안전하게 확인할 수 있습니다.)

5. 대시보드 웹 UI 구현

마지막으로, **대시보드**를 통해 현재까지 수집된 이벤트 이력과 AI 분석 결과를 한 눈에 볼 수 있게 합니다. 오픈소스 프레임워크인 **Streamlit**을 사용하면 별도 웹서버나 프론트엔드 개발 없이 파이썬 코드만으로 대시보드를 실행할 수 있습니다. 아래는 간단한 대시보드 예시 코드입니다:

```
# 파일: dashboard.py (Streamlit 실행 스크립트)
import streamlit as st
import pandas as pd

st.title("SafeGuard AI 대시보드")
st.markdown("실시간 이벤트 모니터링 및 AI 분석 결과")

# 예시: CSV 또는 리스트에서 이벤트 로그 불러오기
events_df = pd.read_csv("events_log.csv")  # 컬럼: time, camera, type, detail, recommendation

# 이벤트 로그 테이블 표시
st.subheader("이벤트 발생 이력")
st.dataframe(events_df)

# 최신 이벤트 상세 및 AI 권고 사항 강조 표시
latest = events_df.iloc[-1]
st.subheader(f"최근 이벤트 - {latest['type']} (카메라 {latest['camera']})")
st.write(f"발생 시각: {latest['time']}")
```

```
st.write(f"상세 내용: {latest['detail']}")
st.write(f"AI 권고 조치: **{latest['recommendation']}**")
```

Streamlit으로 실행하면 (예: `streamlit run dashboard.py`), 웹 브라우저를 통해 대시보드를 볼 수 있습니다. 상단에는 전체 이벤트 로그를 표 형태로 보여주고, 하단에는 가장 최근 이벤트의 상세 정보와 AI의 권고 조치를 강조하여 표시합니다. UI는 필요에 따라 커스터마이즈할 수 있으며, 차트나 필터 등을 추가해서 위험 유형별 발생 추이 등을 시각화할 수도 있습니다.

> **참고:** Streamlit 사용이 익숙하지 않다면, Flask+D3.js로 커스텀 대시보드를 만들거나, 단순히 HTML 파일을 지속 갱신하는 방식으로도 구현 가능하지만 개발 노력이 증가합니다. 4일 이내 빠른 구축을 위해서는 **Streamlit 같은 고수준 툴**이 적합합니다.

로컬 실행 구조 및 예시 파일 구성

이 프로토타입의 프로젝트 구조 예시는 다음과 같습니다:

```
safeguard-ai-prototype/
├── events/
│   ├── event1.json       # 샘플 이벤트 JSON (여러 이벤트 파일 또는 로그 파일)
│   └── ...
├── docs/
│   ├── safety_guide.pdf  # 샘플 작업 지침서 PDF (텍스트 추출 대상)
│   └── ...
├── embeddings/
│   └── chroma/           # ChromaDB 벡터 저장 폴더 (persist_directory)
├── ingest_docs.py        # 문서 텍스트 추출 및 벡터DB 적재 스크립트
├── analyze_event.py      # 이벤트 처리 및 LLM 분석 스크립트 (메인 로직)
├── dashboard.py          # Streamlit 대시보드 실행 스크립트
└── requirements.txt      # 필요한 파이썬 패키지 목록 (sentence_transformers,
chromadb, streamlit 등)
```

실행 순서: 우선 `ingest_docs.py` (또는 Jupyter 노트북 형태)를 통해 `docs/` 폴더의 PDF들을 읽고 텍스트를 추출한 뒤, 임베딩하여 `chromadb` 벡터DB에 저장합니다. 그런 다음 `analyze_event.py`를 실행하면, 스크립트가 이벤트를 모니터링하다가 새로운 이벤트를 발견하면 위에서 설명한 절차대로 분석을 수행합니다. 결과는 콘솔 출력 및 이메일 전송을 하고, 이벤트와 결과를 `events_log.csv` 등에 누적 기록합니다. 마지막으로 `streamlit run dashboard.py` 명령으로 대시보드를 실행해 웹에서 결과를 모니터링합니다.

이러한 구조는 **완전히 로컬 환경에서 동작**하므로, 인터넷에 연결되지 않은 사내망에서도 테스트할 수 있습니다. (단, 이메일 전송을 위해서는 SMTP를 사용할 수 있는 네트워크 환경이 필요합니다.) 모든 구성 요소가 하나의 호스트에서 돌아가기 때문에, 개발/디버깅이 간편하고 초기 프로토타이핑에 적합합니다.

기술 선택 이유 요약

- **Python 기반 구현:** Python은 풍부한 AI/데이터 처리 라이브러리를 갖추고 있어 이벤트 처리부터 NLP까지 일관된 스택으로 개발할 수 있습니다. 간결한 코드로 빠른 프로토타이핑이 가능하여 4일 미만의 개발 기간에 적합합니다.
- **SentenceTransformers + ChromaDB:** 프리트레인된 임베딩 모델(SBERT)은 도메인 지식 문서를 벡터화하여 의미 유사도 검색을 가능하게 합니다. ChromaDB는 **AWS OpenSearch의 벡터 검색** 기능을 대체하는

오픈소스 DB로, 설정이 간단하고 별도 서버 구축 없이 파이썬으로 임베딩 저장/질의가 가능합니다 [5]. 이 조합은 **신뢰성 높고 속도가 빠르며**, Pinecone 같은 외부 서비스 없이도 동작한다는 장점이 있습니다.

- **오픈소스 LLM (예: Llama2, FLAN 등)**: 데이터가 절대로 외부로 나가지 않으며 [9], API 호출 비용 없이 로컬에서 직접 추론을 수행할 수 있습니다. 최신 LLM들은 수 기가바이트 정도로 상대적으로 가벼운 편이며, quantization 기법을 통해 일반 PC에서도 실행 가능하므로 산업 현장의 프라이버시 요구에도 부합합니다. AWS Bedrock 등을 쓰지 않더라도 **동등한 수준의 AI 분석 기능**을 구현할 수 있습니다.
- **SMTP 이메일 및 Streamlit 대시보드:** 이메일은 표준 프로토콜이므로 AWS SES 없이도 구현이 쉽고, 경보 전파 수단으로서 충분합니다. Streamlit은 데이터프레임과 텍스트를 몇 줄의 파이썬 코드로 웹 UI화할 수 있어 개발 생산성이 높습니다. 이를 통해 **AWS QuickSight** 같은 서비스를 쓰지 않고도 사용자용 모니터링 화면을 빠르게 제공할 수 있습니다.
- **경량 통합 및 확장 용이성:** 제안된 구조는 모듈별로 대체가 쉽습니다. 예를 들어, 향후 벡터DB를 Milvus로 교체하거나, 대시보드를 Flask로 확장하는 등 변경이 자유롭습니다. 또한 AWS 서비스로 이전하려면 각 모듈을 대응 서비스로 바꾸면 되므로, 프로토타입을 토대로 **클라우드 배포로 확장하기도 용이**합니다. 무엇보다 모든 구성 요소를 로컬에서 다루기 때문에 개발과 테스트에 드는 시간과 비용을 크게 줄일 수 있습니다.

요약하면, 상기 오픈소스 조합은 **SafeGuard AI**의 핵심 기능(실시간 이벤트 감지, 지식 기반 AI 분석, 알림, 시각화)을 짧은 기간 내에 **실용적인 수준**으로 구현하게 해줍니다. AWS 환경에서의 구성요소와 1:1로 대응되도록 설계했기에, 프로토타입 검증 후 필요에 따라 해당 부분을 AWS 서비스로 전환할 수 있습니다. 이 로컬 RAG 기반 시스템을 통해 산업현장 안전 모니터링에 필요한 엔드투엔드 흐름을 빠르게 구축하고 시험해볼 수 있을 것입니다 [10] [4].

---

[1] [2] [3] [4] [6] [7] [8] [10] PACE AI-Powered Industrial Safety System PR-FAQ.docx
file://file-3XxX73C3JZbWXPJXcSxJBp

[5] Use modular architecture for flexible and extensible RAG-based generative AI solutions | AWS Public Sector Blog
https://aws.amazon.com/blogs/publicsector/use-modular-architecture-for-flexible-and-extensible-rag-based-generative-ai-solutions/

[9] Setting up a Private Retrieval Augmented Generation (RAG) System with Local Llama 2 model and Vector Database | by Jack | Unstructured | Medium
https://medium.com/unstructured-io/setting-up-a-private-retrieval-augmented-generation-rag-system-with-local-vector-database-d42f34692ca7