

DAY 2 / SESSION 3

# Building Cognitive Digital Twin & LangChain



LLM을 단순한 '챗봇'에서 '지능형 에이전트'로 진화시키는 법



SCHEDULE

13:00 ~ 14:30 (90min)



KEYWORDS

LangChain • LangGraph • LLMOps

# The Limitation of LLM

⚠ IQ 150의 천재지만, 세상을 볼 수 없는 뇌



## "Brain in a Jar"

뛰어난 추론 능력을 가졌지만  
외부 세계와 단절된 고립된 지능



### 최신 데이터 부재 (Knowledge Cutoff)

학습 시점 이후의 사건이나 회사의 실시간 생산 현황(Live Data)을 알지 못함.



### 도구 사용 불가 (No Tools)

복잡한 수학 계산을 틀리거나, 사내 데이터베이스(DB)에 접속하여 조회할 수 없음.



### 지속적 기억 부재 (No Memory)

긴 대화의 맥락을 유지하기 어렵고, 이전 대화 내용을 영구적으로 기억하지 못함.

## 💡 Solution: LangChain Framework

LLM에게 노(Retrieval), 순(Tools), 기억(Memory)을 연결하여 한계 극복



# LangChain: AI의 팔다리를 붙이다

💡 LLM을 지능형 에이전트로 만드는 5대 핵심 컴포넌트



## Chains

여러 작업을 논리적인 순서대로 엮어서 실행.

Example: Prompt → LLM → Output Parser

WORKFLOW



## Agents

어떤 도구를 어떤 순서로 사용할지 스스로 판단.

Example: "질문 분석 → 검색 → 답변 생성"

BRAIN / DECISION



## Memory

대화의 맥락(Context)을 저장하고 기억하여 일관성 유지.

Example: Chat History Buffer

CONTEXT



## Retrievers (RAG)

LLM이 학습하지 않은 외부 데이터(매뉴얼, 사내 DB)를 검색하여 제공.

Example: Vector DB Search, PDF Loader

EYES / KNOWLEDGE



## Tools

검색, 계산, API 호출 등 LLM이 실제로 수행하는 행동(Action).

Example: Google Search, Python REPL, Calculator

HANDS / ACTION

# Chain vs Agent

"시키는 대로" (절차적 실행) vs "알아서" (자율적 판단)



## Chain

Hard-coded Sequence

✓ 절차적 (Procedural)

"A 하고 B 하고 C 해." 순차적 실행

✓ 결정적 (Deterministic)

항상 같은 입력에 같은 결과 보장, 재현성 높음

⚠ 유연성 부족 (Rigid)

예상치 못한 예외 상황 발생 시 대처 어려움



## Agent

Autonomous Loop

✓ 자율적 (Autonomous)

"목표는 C야, 방법은 네가 정해." 도구 선택권 부여

✓ 동적 루프 (Dynamic Loop)

상황에 따라 재시도(Retry), 경로 수정, 반복 가능

⚠ 비용/속도 변동성

무한 루프 위험, 실행 시간 및 토큰 비용 예측 어려움

VS

❖ BEST USE CASES

ETL 파이프라인

문서 요약

단순 번역

RAG (단일 검색)

❖ BEST USE CASES

복잡한 문제 해결

웹 검색/탐색

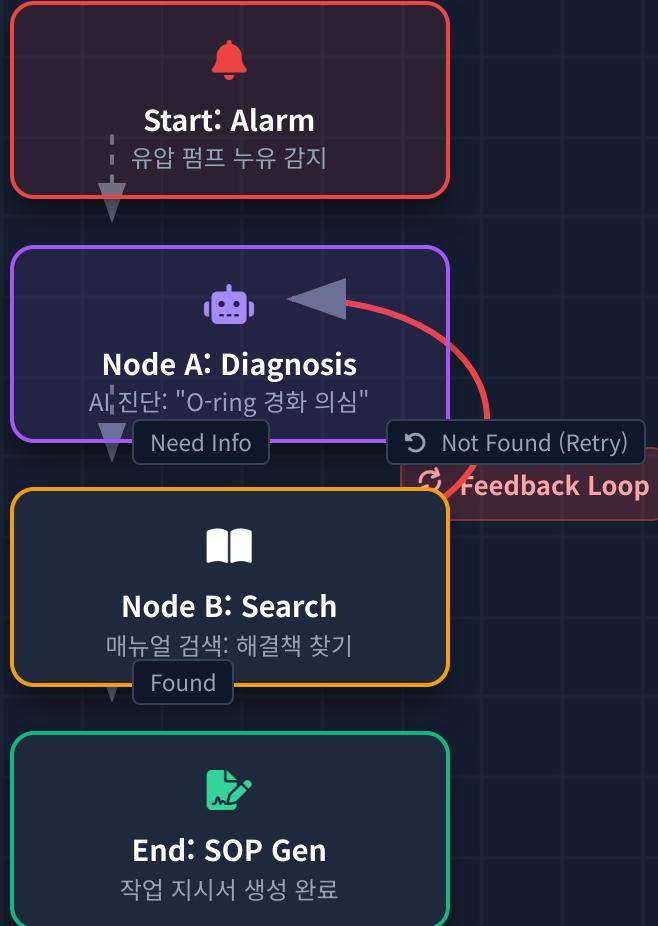
데이터 분석(코딩)

개인 비서

💡 규칙이 명확하고 안정성이 중요하다면 **Chain**을, 문제 해결과 탐색이 필요하다면 **Agent**를 선택하세요.

# LangGraph: Manufacturing Scenario

실제 제조 공정 예제: "유압 펌프 누유 진단 및 복구"



## Why LangGraph?

제조 현장에서 단순한 "질문-답변"이 아니라, 문제 해결이 될 때까지 스스로 반복 (Loop)하는 에이전트가 필요합니다.



## Graph-based Control

"O-ring 경화"라는 진단을 내렸지만 해결책을 못 찾으면, 다시 진단 단계로 돌아가 "필터 막힘" 가능성을 재검토합니다.



## Concrete Scenario

- 감지: RPM 저하 & 토크 증가 감지
- 진단: AI가 "O-ring 손상" 의심
- 검색: 매뉴얼에서 교체 절차 검색
- 완료: 작업자에게 "부품 교체 SOP" 전달

# LLMOps & Monitoring

"AI가 왜 저런 답을 했지?" 블랙박스를 투명하게 만드는 기술

## LLMOps (Large Language Model Operations)

LLM 애플리케이션의 개발, 배포, 모니터링, 관리를 위한 통합 운영 체계.



ANALOGY

제조 현장의 SCADA / MES 시스템



### Tracing (추적)

프롬프트 입력부터 최종 답변까지, 중간에 어떤 도구를 호출했는지 전 과정을 시각화하여 디버깅 지원.



### Metrics (지표)

토큰 사용량(비용), 응답 지연 시간(Latency), 에러 발생률 등 핵심 운영 지표를 실시간 모니터링.



### Evaluation (평가)

프롬프트나 체인 로직 변경 시, 기존 답변 품질이 떨어지지 않는지 자동 회귀 테스트(Regression Test) 수행.



### Versioning (버전 관리)

프롬프트 템플릿, RAG 지식베이스, 모델 파라미터의 변경 이력을 관리하여 언제든 이전 상태로 롤백 가능.



### Tools (LangFuse / LangSmith)

LangChain과 통합된 대표적인 LLMOps 도구들. **LangSmith**(LangChain 공식)와 **LangFuse**(오픈소스)가 업계 표준.

# 실습 개요: 제조 데이터와의 대화

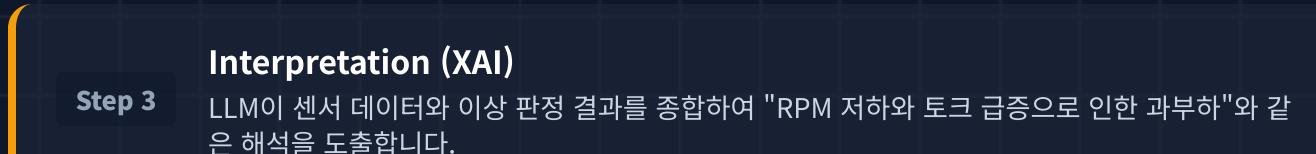
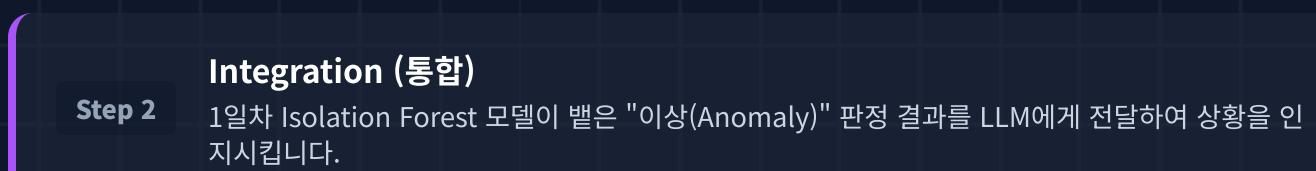
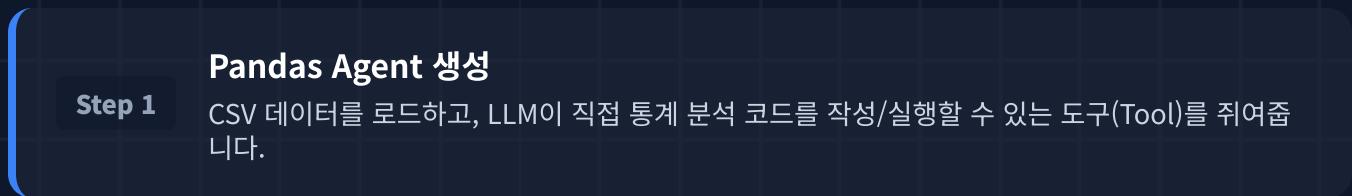
LangChain + Pandas Agent를 활용한 데이터 해석 및 XAI 실습

## Training Goal (학습 목표)



1일차의 '이상 탐지 결과'와 '센서 데이터(CSV)'를 LangChain으로 연결하여,  
AI가 스스로 데이터를 분석하고 "왜 이상인지" 기술적 원인을 설명하도록 구현합니다.

## Scenario Flow



## Configuration (50min)

- Environment:** Google Colab (Python 3.10+)
- Libraries:** langchain, langchain\_experimental, openai
- Data:** Simulated Sensor Data (CSV)
- API Key:** OpenAI API Key (Required)



### 주의사항:

실습 코드의 실행 결과는 LLM의 특성상 매번 조금씩 다를 수 있습니다. (Temperature=0 설정 권장)

# 🐍 코드 셀 1: 환경 설정 및 라이브러리 설치

LangChain 생태계를 Colab 환경에 구성하고, 실습에 필요한 핵심 라이브러리를 로드합니다.

```
● ● ● </> setup.py

# [실습] LangChain 생태계 설치
# -q 옵션: 설치 로그 최소화 (Quiet mode)
!pip install -q langchain langchain_experimental langchain_openai

import pandas as pd
import os
from langchain_experimental.agents.agent_toolkits import
create_pandas_dataframe_agent
from langchain_openai import ChatOpenAI

# API Key 설정 (환경변수)
# 실습 시 본인의 OpenAI API Key를 입력하세요.
# Colab의 '비밀(Secrets)' 기능을 사용하는 것이 보안상 안전합니다.
# os.environ["OPENAI_API_KEY"] = "sk-..."

# 모델 설정 준비
llm = ChatOpenAI(
model="gpt-4o",
```

## 💡 Execution Tips

### ⟳ Runtime Restart

라이브러리 최초 설치 후에는 [런타임] > [세션 다시 시작]이 필요할 수 있습니다. (버전 충돌 방지)

### ⌚ Model Selection

복잡한 데이터 추론에는 **GPT-4o**가 필수적입니다. GPT-3.5는 통계 분석 코드 생성 시 오류가 찾을 수 있습니다.

### ⓧ Temperature = 0

창의성보다는 정확성이 중요한 분석 작업이므로, 무작위성을 제거하여 일관된 답변을 유도합니다.

#### API Key Security

🔑 공용 컴퓨터나 화면 공유 시 API Key가 노출되지 않도록 주의하세요.



## 코드 셀 2: 1일차 결과 데이터 로드

1일차 실습 결과(가상 센서 데이터 + 이상 탐지 결과)를 시뮬레이션 데이터프레임으로 생성합니다.

● ● ● </> load\_data.py

```
# 1. 센서 데이터 (Feature) 생성
# 09:02 ~ 09:03 구간에서 고장 패턴 발생 시뮬레이션
data = {
    'Time': ['09:00', '09:01', '09:02', '09:03', '09:04'],
    'RPM': [2000, 2010, 1400, 1350, 2005], # 속도 급감
    'Torque': [40, 41, 65, 70, 40], # 토크 급증 (과부하)
    'Temperature': [50, 51, 85, 88, 52], # 온도 급상승
    'Vibration': [0.5, 0.6, 2.1, 2.3, 0.5] # 진동 증가
}
df_sensor = pd.DataFrame(data)

# 2. 1일차 이상 탐지 모델(Isolation Forest)의 예측 결과 (가정)
# 1: 정상(Normal), -1: 이상(Anomaly)
df_sensor['Anomaly_Score'] = [1, 1, -1, -1, 1]

# 데이터 확인
print("== [Digital Twin] 센서 데이터 및 이상 탐지 결과 ==")
display(df_sensor)
```



### Data Columns

**Time** 시계열 기준 (Time-series index)

**RPM** 회전 속도 (Rotations Per Minute)

**Torque** 모터 부하 (Newton Meter)

**Anomaly** 1(정상) / -1(이상)

#### ⚠ Anomaly Pattern (09:02~03)

**RPM ↓**

1400 (Low)

**Torque ↑**

70Nm (High)

**Temp ↑**

88°C (Hot)

**Score**

-1 (Fault)

"속도는 줄고 토크는 늘어나는 전형적인 과부하/고착 패턴"

# 🤖 코드 셀 3: Pandas Agent 생성 (데이터 분석가)

LLM에게 데이터프레임을 분석하고 Python 코드를 직접 실행할 수 있는 권한을 부여합니다.

● ● ● </> create\_agent.py

```
# LLM에게 데이터프레임을 다룰 수 있는 도구를 줘여줍니다.  
# temperature=0 : 분석은 창의성보다 정확성이 중요  
  
agent = create_pandas_dataframe_agent(  
    ChatOpenAI(  
        model="gpt-4o",  
        temperature=0  
    ),  
    df_sensor,  
    verbose=True, # 생각의 과정(CoT) 출력  
    allow_dangerous_code=True # 내부적으로 Python 코드 실행 허용  
)  
  
print("Agent Created Successfully!")  
print(f"Agent Tools: {agent.tools}")
```

## 💡 Best Practices

### 🔒 **temperature=0**

코드 생성 시 환각(Hallucination)을 최소화하고, 실행 가능한 정확한 Python 구문을 생성하기 위해 필수적입니다.

### ⌚ **verbose=True (Educational)**

Agent가 "어떤 생각을 하고(Thought), 어떤 코드를 짬는지(Action)" 로그로 보여주어 학습에 매우 유용합니다.

### 🛡 **allow\_dangerous\_code=True**

실습 편의를 위해 허용하지만, 실제 운영 환경(Production)에서는 샌드박스 등 보안 조치가 반드시 필요합니다.

## ⚙️ How it works

1. User asks question → 2. LLM reasons about required pandas functions
3. Generates code (e.g., `df.mean()`) → 4. Executes code & returns result



## 코드 셀 4: 실습 1 - 자연어 질의

SQL이나 파이썬 문법을 몰라도, 자연어로 데이터를 조회하고 통계를 추출합니다.



</> query\_data.py

```
# 자연어 질의 작성
query1 = "지금 데이터에서 '비정상(Anomaly_Score = -1)'으로 판정된 시간대는 언제야? 그리고 그때 RPM 평균은 얼마야?"

print(f"User Query: {query1}\n")

# Agent 실행 (CoT: Chain of Thought)
agent.invoke(query1)

# [Agent 실행 로그 예상 출력]
# > Entering new AgentExecutor chain...
# Thought: I need to filter the dataframe for rows where 'Anomaly_Score' is -1,
# then select the 'Time' column and calculate the mean of 'RPM'.
#
# Action: python_repl_ast
# Action Input: df[df['Anomaly_Score'] == -1]['Time'], df[df['Anomaly_Score'] == -1]
# ['RPM'].mean()
#
# Observation: (09:02, 09:03), 1375.0
```



### Agent Reasoning Process



#### THOUGHT (계획)

"Anomaly\_Score가 -1인 행을 찾고, Time과 RPM 평균을 계산해야겠다."



#### ACTION (코딩)

```
df[df['Anomaly_Score'] == -1]
['RPM'].mean()
```



#### ANSWER (답변)

"비정상 시간대는 09:02, 03분이며 평균 RPM은 1375입니다."



#### Tip: Prompt Engineering

컬럼명(예: RPM, Anomaly\_Score)을 질문에 명확히 포함하면 Agent가 코드를 더 정확하게 작성합니다.

# 🔍 코드 셀 5: 실습 2 - 모델 결과 해석(XAI)

LLM에게 비정상 구간의 데이터를 심층 분석시키고, 기술적인 원인을 3가지 관점에서 추론합니다.

● ● ● </> xai\_analysis.py

```
# 복잡한 해석 요청 (XAI: Explainable AI)
# 단순히 '고장' 판정뿐만 아니라, '왜' 그런지 원인을 물습니다.

query2 = """
09:02에 이상 탐지 모델이 '비정상(-1)'이라고 판정했어.
데이터(RPM, Torque, Temperature, Vibration)를 분석해서,
왜 모델이 이때를 비정상이라고 판단했을지 기술적인 원인을 3가지 관점에서 설명해줘.
정상일 때(09:00)와 수치를 비교해서 설명해.
"""

print(f"User Query: {query2}\n")

# Agent 실행
result = agent.invoke(query2)

print("\n== [AI 해석 결과] ==")
print(result['output'])
```

## 🤖 AI Interpretation Result

### ⚠ 1. 과부하 발생 (Torque 급증)

정상 시 40Nm에서 65Nm로 **약 60% 급증**. 모터에 물리적 부하 발생.

### ⚡ 2. 회전수 저하 (RPM 감소)

2000에서 1400으로 급락. 베어링 고착 또는 이물질 끼임 패턴.

### 🌡 3. 발열 동반 (Temperature 상승)

50°C에서 85°C로 상승. 기계적 마찰열 발생으로 즉시 정지 필요.

## 💡 Instructor's Teaching Guide

🔗 **LangGraph:** "결재 반려 시 재상신"처럼 업무 루프(Loop)를 만드는 도구

🎥 **LLMOps:** AI의 CCTV (LangFuse). 누가 언제 무슨 질문했는지 기록

👤 **Pandas Agent:** 사장님의 "김대리" 대신 AI에게 직접 매출을 묻는 시대

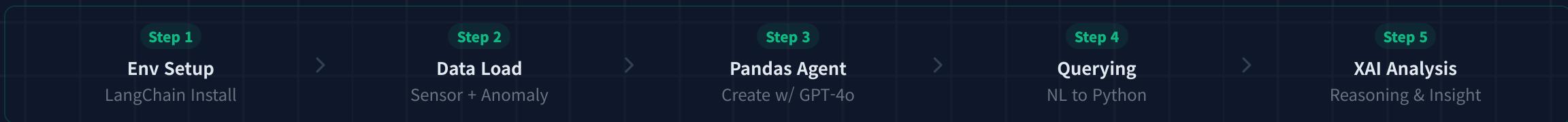
# ☰ 세션 정리 요약: LangChain 심화

이론(Theory)과 실습(Practice)을 통해 디지털 트윈에 지능을 더하는 과정을 학습했습니다.

## ➊ THEORETICAL CORE (이론 핵심)



## ➋ PRACTICAL WORKFLOW (실습 요약)



## ➌ ACHIEVEMENT CHECKLIST

- ✓ Pandas Agent로 비정상 구간 및 통계값 질의 성공
- ✓ AI의 이상 원인 해석(XAI)을 수치 근거로 확인
- ✓ LangGraph의 Loop(반복/재상신) 개념 이해
- ✓ LLMOps(LangFuse)를 통한 모니터링 필요성 인식

## ▶ NEXT STEPS

- ➡ **Real-time & RAG**  
스트리밍 센서 연동 및 사내 매뉴얼 지식베이스 구축
- ➡ **Advanced LLMOps**  
Tracing 도구 도입 및 비용/권한 관리 정책 수립

# Q&A | 질의 응답

궁금한 점을 자유롭게 질문해 주세요. 현장 적용 시 발생할 수 있는 이슈들을 함께 고민합니다.



## 데이터 보안

민감 데이터는 PII 마스킹 처리하거나, 사내 구축형(On-premise) LLM을 검토하세요.



## 비용 관리

사용량 급증 방지를 위해 사용자별 토큰 한도(Quota)를 설정하고 캐싱을 적용하세요.



## 모델 대체

GPT-4o가 비싸다면? 간단한 분석은 GPT-3.5나 Claude Haiku로 대체 가능합니다.



## 디버깅

답변이 이상할 땐? Trace(추적) 로그를 통해 어느 단계(검색/생성) 문제인지 확인하세요.

## \_OPERATION BEST PRACTICES

### ✓ temperature=0

데이터 분석이나 코딩 작업 시에는 '창의성'보다 '정확성'이 우선입니다.

### ✓ dangerous\_code=False

실제 운영 환경(Prod)에서는 Python 실행 권한을 엄격히 제한하거나 샌드박싱해야 합니다.

### ✓ Regression Test

Prompt나 Chain 구조 변경 시, 기존 질문셋에 대한 성능 저하가 없는지 자동 평가하세요.

# Thank You

"Data + Tools + Graph Control + Ops"

오늘 실습한 4가지 핵심 요소를 기억해 주세요.