


# LangGraph




# 1. LangGraph 개념

---



LangChain의 **LangGraph**는 LLM 애플리케이션의 복잡한 워크플로우(Workflow)를 시각적이고 구조적으로 설계할 수 있게 해주는 “**그래프 기반 실행 프레임워크**”이다.

LLM 호출(Chain, Tool, Retriever, Memory 등)을 노드(Node)로, 데이터 흐름을 엣지(Edge)로 정의하여 **AI Agent나 RAG 시스템의 실행 단계를 명확히 구성**할 수 있다.



## 1. LangGraph의 핵심 개념

| 개념                 | 설명   |
|--------------------|--|
| 노드(Node)           | 체인, LLM 호출, 함수, 도구(tool) 등을 하나의 실행 단위로 표현  |
| 엣지(Edge)           | 노드 간의 데이터 흐름(출력 → 입력)을 연결  |
| State (상태)         | 실행 중 데이터의 현재 상태를 저장하는 객체 (예: 대화 히스토리, 검색 결과 등)   |
| Graph State        | 전체 워크플로우의 상태 관리 단위 (Mutable dictionary 형태로 유지)   |
| Runnable Interface | 각 노드가 실행 가능한 함수형 인터페이스를 따름 ( <code>RunnableLambda</code> , <code>RunnableMap</code> , <code>RunnableSequence</code> 등) |
| Conditional Edge   | 조건에 따라 다음 노드를 다르게 실행하도록 분기 설정 가능 (if-else 로직)  |
| LangGraph Studio   | 그래프를 시각적으로 설계하고 테스트할 수 있는 웹 기반 GUI 도구  |



## 2. LangGraph의 주요 기능

### 기능

### 설명

멀티 노드 구성

여러 단계의 LLM, Tool, Retriever 등을 연결한 파이프라인 구성

상태 기반 실행(Stateful Execution)

메모리나 캐시를 통해 이전 결과를 참조하며 실행 가능

조건 분기(Conditional Routing)

응답 내용이나 특정 키워드에 따라 다른 노드로 분기

병렬 실행(Parallel Execution)


여러 노드를 동시에 실행 후 결과를 병합

실패 복구(Fallback / Retry)

오류 발생 시 대체 노드나 재시도 로직 설정

LangSmith 통합

실행 추적, 디버깅, 성능 모니터링 가능



### 3. LangGraph 기본 구조 예제

다음은 "질문 → 검색 → 요약 → 결과 응답" 형태의 간단한 RAG 그래프 예시이다.

```
from typing import TypedDict
from langgraph.graph import StateGraph, END
from langchain_core.runnables import RunnableLambda

# 1) 상태 정의 (TypedDict 사용)
class QASState(TypedDict):
    question: str
    docs: list[str]
    answer: str

# 2) 노드 함수 정의
def retrieve_node(state: QASState) -> QASState:
    state["docs"] = [f"검색된 문서: {state['question']} 관련 내용"]
    return state

def summarize_node(state: QASState) -> QASState:
    docs = "\n".join(state["docs"])
    state["answer"] = f"요약 결과: {docs}"
    return state
```

# 3) 그래프 구성

```
graph = StateGraph(QAState)
graph.add_node("retriever", RunnableLambda(retrieve_node))
graph.add_node("summarizer", RunnableLambda(summarize_node))
```

# 4) 엣지 연결

```
graph.add_edge("retriever", "summarizer")
graph.set_entry_point("retriever")
graph.set_finish_point("summarizer")
```

# 5) 그래프 실행

```
app = graph.compile()
result = app.invoke({"question": "LangGraph란 무엇인가?", "docs": [], "answer": ""})
print(result["answer"])
```

## 출력 예시

요약 결과: 검색된 문서: LangGraph란 무엇인가? 관련 내용



## 4. LangGraph의 응용 예시

### 시나리오

### 설명

멀티 Agent 협업 시스템

Data Agent → Analysis Agent → Report Agent 순으로 그래프 구성

RAG 파이프라인

Document Loader → Text Splitter → Embedding → Retriever → LLM

대화형 챗봇 구조화

사용자 입력 → Intent 분류 → Tool 호출 → 응답 생성

LangChain Memory 연동

Graph State에 메모리를 삽입하여 지속 대화 상태 관리

에러 복구 그래프

실패 시 fallback 또는 재시도(RunnableRetry) 노드 연결

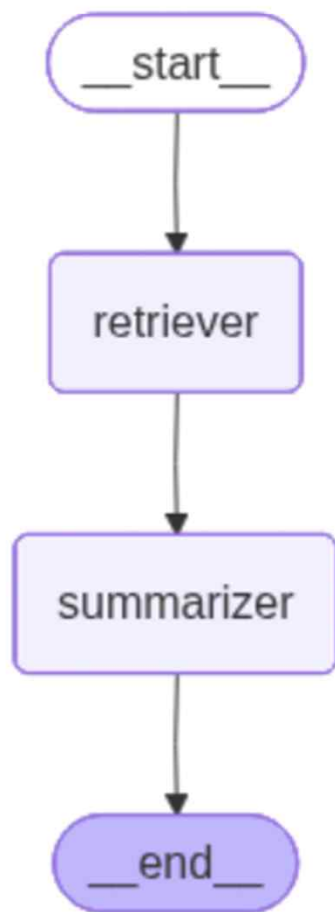




## 5. LangGraph vs 기존 LangChain 비교

| 항목    | LangChain (기존 체인 구조)  | LangGraph (그래프 구조)              |
|-------|-----------------------|---------------------------------|
| 구조    | 순차적 실행 (linear chain) | 병렬/분기 실행 가능 (graph)             |
| 확장성   | 복잡한 워크플로우에 불리         | 복잡한 워크플로우에 최적                   |
| 디버깅   | 코드 레벨 추적 중심           | 시각적 노드 추적 가능 (LangGraph Studio) |
| 상태 관리 | Memory Class 중심       | StateGraph 기반 전체 상태 관리          |
| 사용 목적 | 단순 체인, RAG            | 멀티 Agent, 분기형 챗봇, 워크플로우 오케스트레이션 |

```
1 # LangGraph 시각화
2 from IPython.display import Image, display
3
4 display(Image(app.get_graph().draw_mermaid_png()))
```





감사합니다