

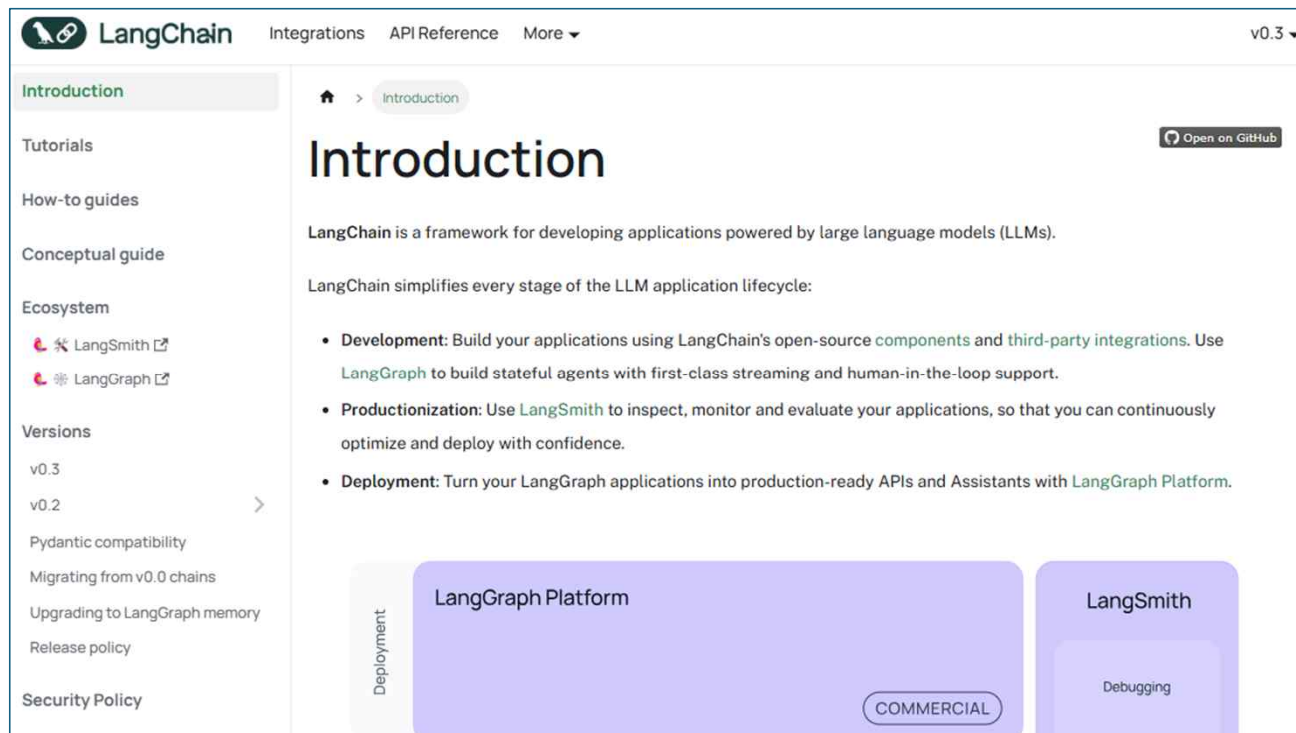
LangChain Overview

가상환경 생성 및 package 설치

- `conda create --name langchain python=3.11`
- `conda activate langchain`
- `pip install -r requirements.txt`

LangChain 문서 구성

- <https://python.langchain.com/docs/introduction/>

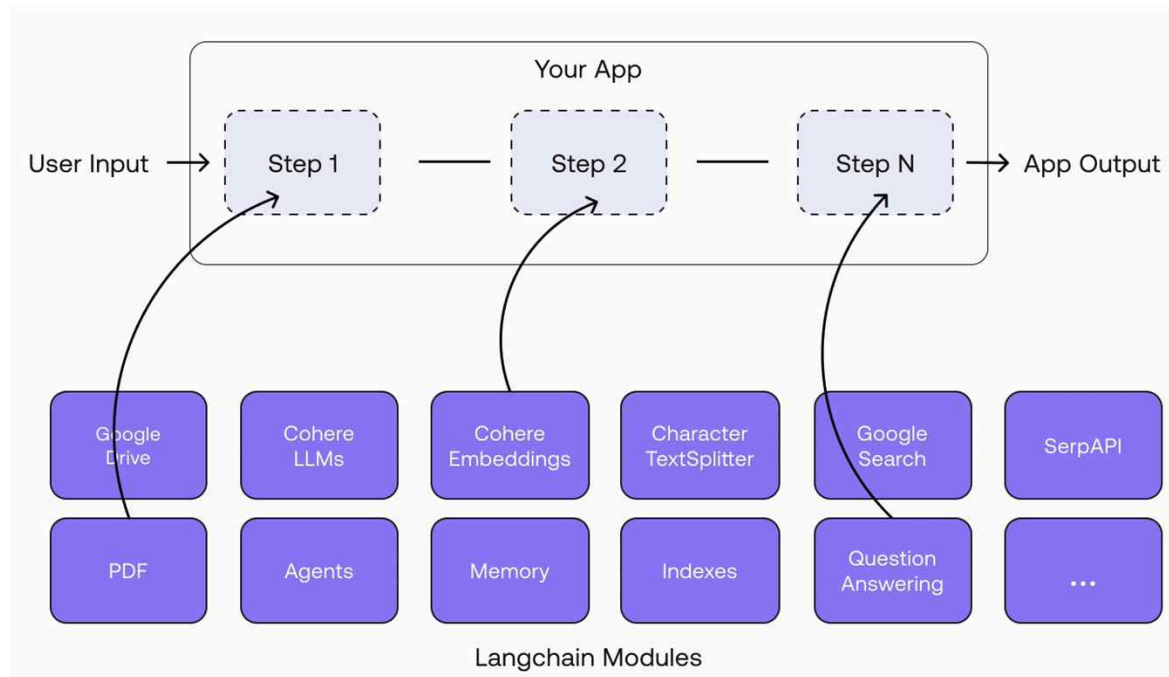


LangChain이란?

- 대규모 언어 모델(LLM)을 활용한 애플리케이션을 개발하기 위한 프레임워크
- LLM 애플리케이션의 전체 라이프사이클을 간소화
- Python 및 JavaScript 지원
- GPT-4와 같은 LLM을 외부 데이터(file, database, APIs)와 결합하여 챗봇, 코드 이해, 요약 등 다양한 애플리케이션 개발 지원
- 사전 구축된 도구 및 프롬프트 템플릿 제공: 다양한 사용 사례에 맞는 입력 예제와 모델 응답 형식을 쉽게 지정

What is Chain ?

- 특정 목표를 달성하기 위해 서로 연결되는 일련의 행동 또는 작업
- 여러 개의 컴포넌트를 결합해 하나의 질서 정연한 애플리케이션을 생성
- 사용자 입력을 받아 언어 모델로 처리한 다음 응답을 생성하는 체인을 만들 수 있다



사용자의 입력이 여러 단계를 거쳐 최종 결과를 만들어내는 **일련의 처리 흐름**

각 Step에서 활용 가능한 **LangChain의 모듈(기능 요소)들**

LangChain 프레임워크 개요



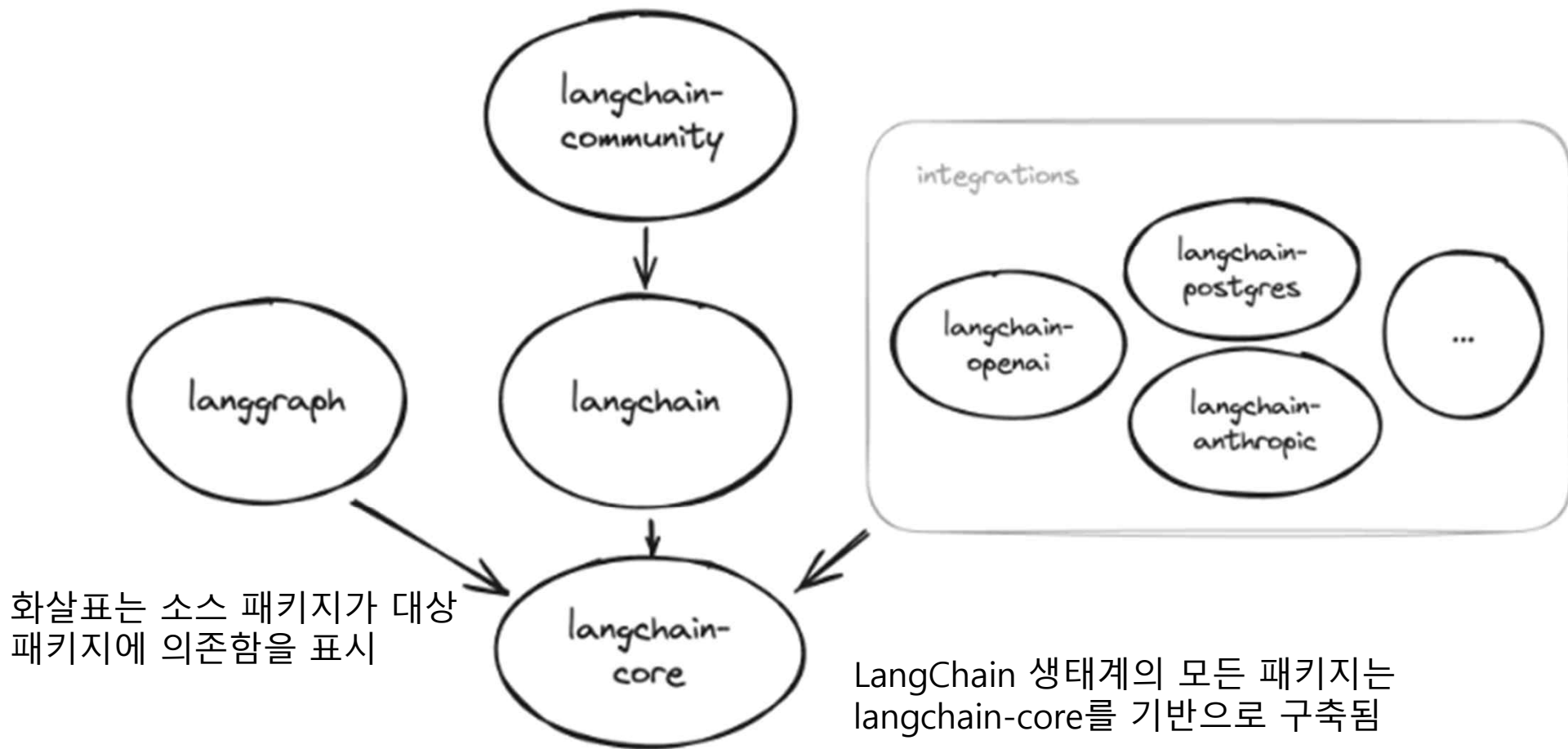
OSS

자유롭게 쓸 수 있는 기본 도구 (프레임워크 및 연결)

COMMERCIAL

상업적 목적의 고급 기능 (운영, 관리, 디버깅 등)

LangChain Packages 의존성 관계



LangChain의 장점과 단점

- 장점

- 오픈 소스이므로 누구나 자유롭게 이용 가능하고 빠른 업데이트 가능
- 다양한 LLM 에 표준화된 인터페이스 지원 → LLM 공급자 전환 용이
- 모듈화된 설계: 프롬프트 템플릿, 문서 로더, 출력 파서 등 다양한 모듈 제공
- 관찰 가능성 및 평가 기능 제공

- 단점

- 언어 지원 제한: Python 및 JavaScript 이외의 언어 지원 부족
- 복잡한 초기 설정과 다양한 옵션으로 초보자에게는 진입 장벽이 있을 수 있음
- 복잡한 기능과 다양한 옵션으로 인해 학습 곡선이 가파를 수 있음
- API 업데이트 반영 지연: OpenAI 등의 새로운 기능이 즉각 반영되지 않을 수 있음

실습: 000_LangChain_Overview

- LLM 연결 – OpenAI
- Messages – AIMessage, HumanMessage, SystemMessage
- 프롬프트 템플릿과 로더를 사용하여 체인 구성
- Runnables 와 LCEL (LangChain 표현 언어)
- Chain 이 달린 도구 사용

Langgraph 란?

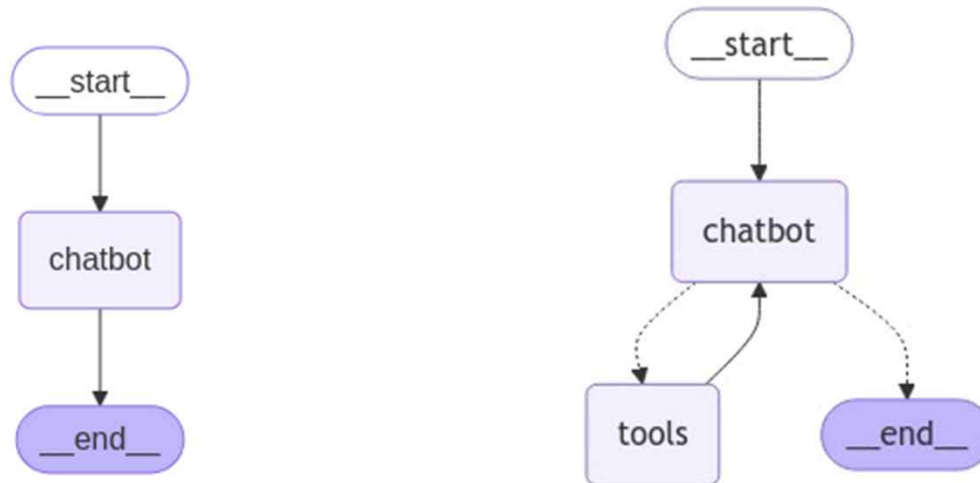
- LLM을 활용해 상태 기반의 다중 에이전트 애플리케이션을 구축
- 그래프 구조를 통해 복잡한 워크플로우를 효율적으로 설계 및 관리
- 상태를 지속적으로 유지해 반복 작업과 복잡한 상태 관리를 지원
- 인간의 개입이 필요한 워크플로우와 메모리 기능을 유연하게 구현
- 주요 특징
 - 상태 지속성
 - 워크플로우 최적화
 - 복잡한 작업 처리
- 활용 사례 - 고객 서비스 챗봇, 복잡한 데이터 분석, 교육용 튜터링 시스템 등

LangGraph와 LangChain의 차이점

특징	LangGraph	LangChain
주요 목적	복잡한 워크플로우 및 의사결정 프로세스 구현	LLM 통합 및 체인 구성
구조	그래프 기반	체인 및 에이전트 기반
상태 관리	명시적이고 세밀한 제어	암시적이고 자동화된 관리
유연성	높음 (커스텀 로직 쉽게 구현)	중간 (미리 정의된 컴포넌트 중심)
학습 곡선	상대적으로 더 가파름	가파름 (상대적으로 완만함)
용도	복잡한 AI 시스템, 다중 에이전트	간단한 LLM 애플리케이션, RAG

Graph 핵심 요소

- State – 애플리케이션의 현재 스냅샷을 나타내는 공유 데이터 구조. TypedDict 나 Pydantic BaseModel로 구성.
- Nodes – Agent 의 logic을 가진 Python 함수. 현재 값을 State 입력으로 받고, 업데이트된 값을 반환
- Edges – 현재 State에 따라 다음에 실행할 Node 를 결정하는 Python 함수. 조건부 분기 또는 고정 전환 가능.



State 구성

- State는 여러 키를 가진 딕셔너리(TypedDict)
- 각 키는 독립적인 Reducer(병합 규칙)를 가집니다.
- Reducer를 명시하지 않으면 기본값은 덮어쓰기(override)입니다.

```
** 기본 State - no Reducer **  
from typing_extensions import TypedDict  
  
class State(TypedDict):  
    foo: int  
    bar: list[str]
```

```
** State - operator.add Reducer **  
  
from typing import Annotated  
from typing_extensions import TypedDict  
from operator import add  
  
class State(TypedDict):  
    foo: int  
    bar: Annotated[list[str], add]
```

Node 구성

- 파이썬 함수(동기 또는 비동기)이며, 첫 번째 위치 인수는 state 이고(선택적으로) 두 번째 위치 인수는 선택적 구성 가능한 매개변수 (예: thread_id)를 포함하는 "config"

```
def my_node(state: State, config: RunnableConfig):  
    print("In node: ", config["configurable"]["user_id"])  
    return {"results": f"Hello, {state['input']}!"}
```

- START node – 사용자 입력을 그래프로 전송하는 특수 노드
- END node – 완료를 나타내는 특수 노드

Edge 구성

- 일반 에지: 한 노드에서 다음 노드로 직접 이동
- 조건부 에지: 다음에 어느 노드로 이동할지 결정하는 함수를 호출
- 진입점: 사용자 입력이 도착했을 때 가장 먼저 호출할 노드 결정
- 조건부 진입점: 사용자 입력이 도착했을 때 먼저 호출할 노드를 결정하는 함수를 호출

```
graph.add_edge(START, "node_a")
```

```
graph.add_conditional_edges(START, routing_function, {True: "node_b", False: "node_c"})
```

실습: 050. Langgraph - Chatbot 만들기

- 기본 chatbot 구축 – 외부 정보 접근 X, state 기억 X
- 외부 도구를 활용한 chatbot 강화
- 메모리 기능 추가
- Prompt Template 적용

Agent 란 ?

- LLM을 핵심 엔진으로 사용하여 주어진 목표를 달성하기 위해 독립적으로 작업(추론, 판단, 실행, 피드백)을 수행하는 인공지능 시스템
- 주로 LLM(대규모 언어 모델)의 능력을 기반으로 동작하며, 사용자의 명령을 이해하고, 판단하며, 실행
- 즉, LLM은 두뇌, Agent는 이 두뇌를 활용해 행동을 실행하는 작업자 역할
- 활용 사례
 - 고객 지원 챗봇: 전자상거래 플랫폼에서 환불 요청을 처리하거나 배송 상태 확인
 - 연구 도우미: 과학 논문에서 특정 주제에 대한 핵심 내용 추출
 - 개인 비서: 이메일 초안을 작성하고, 미팅 일정 조율
 - 데이터 분석 에이전트: 주식 시장 데이터를 분석해 투자 전략 제안
 - 교육용 튜터: 수학 문제 풀이 과정을 단계별로 설명

Agent의 핵심 기술 스택

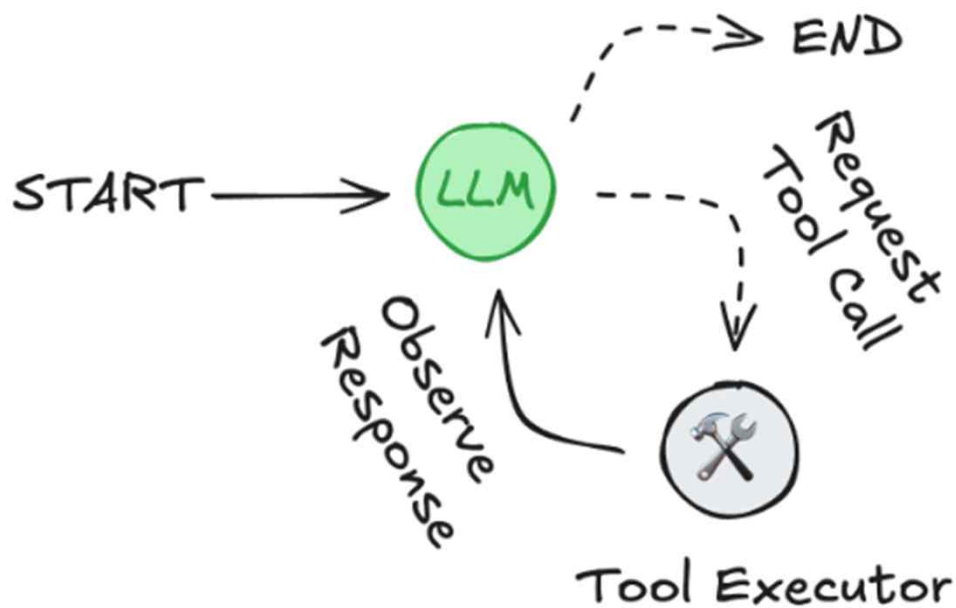
- LLM (예: GPT-4, Claude): 언어 이해 및 생성
- LangChain / LangGraph: 에이전트 워크플로우 및 상태 관리
- Vector Database (예: Chroma, Pinecone): 정보 저장 및 검색
- Memory: 상태 유지 및 맥락 관리
- API 통합 (예: Tavily, SerpAPI): 외부 도구와 연결

ReAct (Reasoning + Acting)

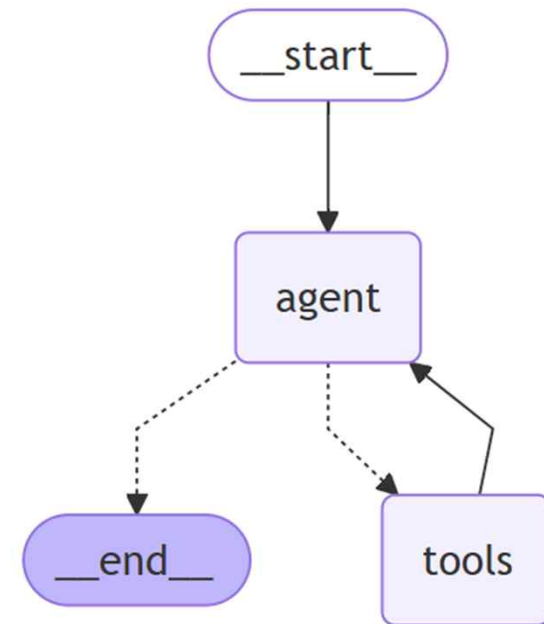
- LLM이 추론(Reasoning)과 행동(Acting)을 결합하여 더 강력한 성능을 발휘하도록 하는 프레임워크. 즉, LLM이 논리적으로 사고하면서 필요한 경우 외부 도구를 활용할 수 있도록 설계된 방법론
- Reasoning (추론): 자연어 기반으로 문제 해결을 위한 논리적 사고 수행
- Acting (행동): API 호출, 데이터베이스 검색, 웹 검색 등 외부 도구를 활용하여 필요한 정보 획득
- 예) 내가 사는 곳의 날씨를 알려줘
 - Reasoning(추론) - 사용자의 위치를 알아야 하고, 날씨 API에서 날씨 정보를 가져와야 한다. (LLM)
 - Acting(행동) - TavilySearchResults API 호출 (LangChain)
 - Observation(관찰) - API로부터 받은 정보를 LLM에 다시 전달. (LangChain)
 - Reasoning(추론) - 추가적인 정보를 이용하여 추론 (LLM)
 - Answer(응답) - "서울의 날씨는 맑음입니다." (LLM)

에이전트 루프(Agent loop)

- LLM이 도구들을 선택하고, 그 도구들의 출력을 활용하여 사용자의 요청을 수행



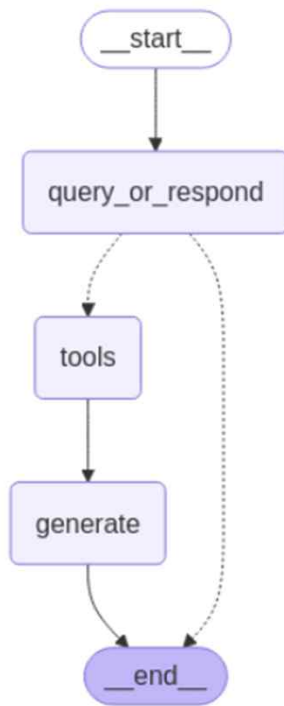
Graph



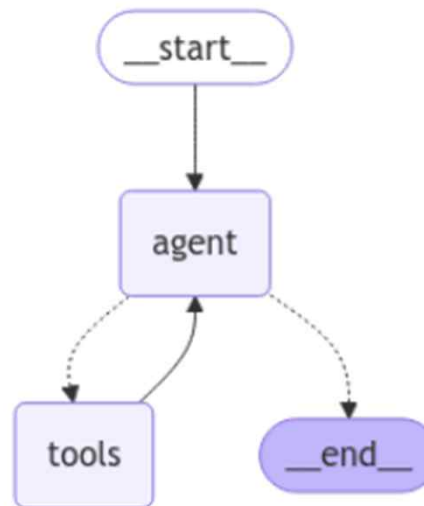
Chain vs. Agent

- 체인(Chains) - 한 번의 검색 단계만 실행
- 에이전트(Agents) - LLM이 필요에 따라 여러 번의 검색 단계 (다중 단계 검색) 를 수행하도록 자유롭게 설정 가능

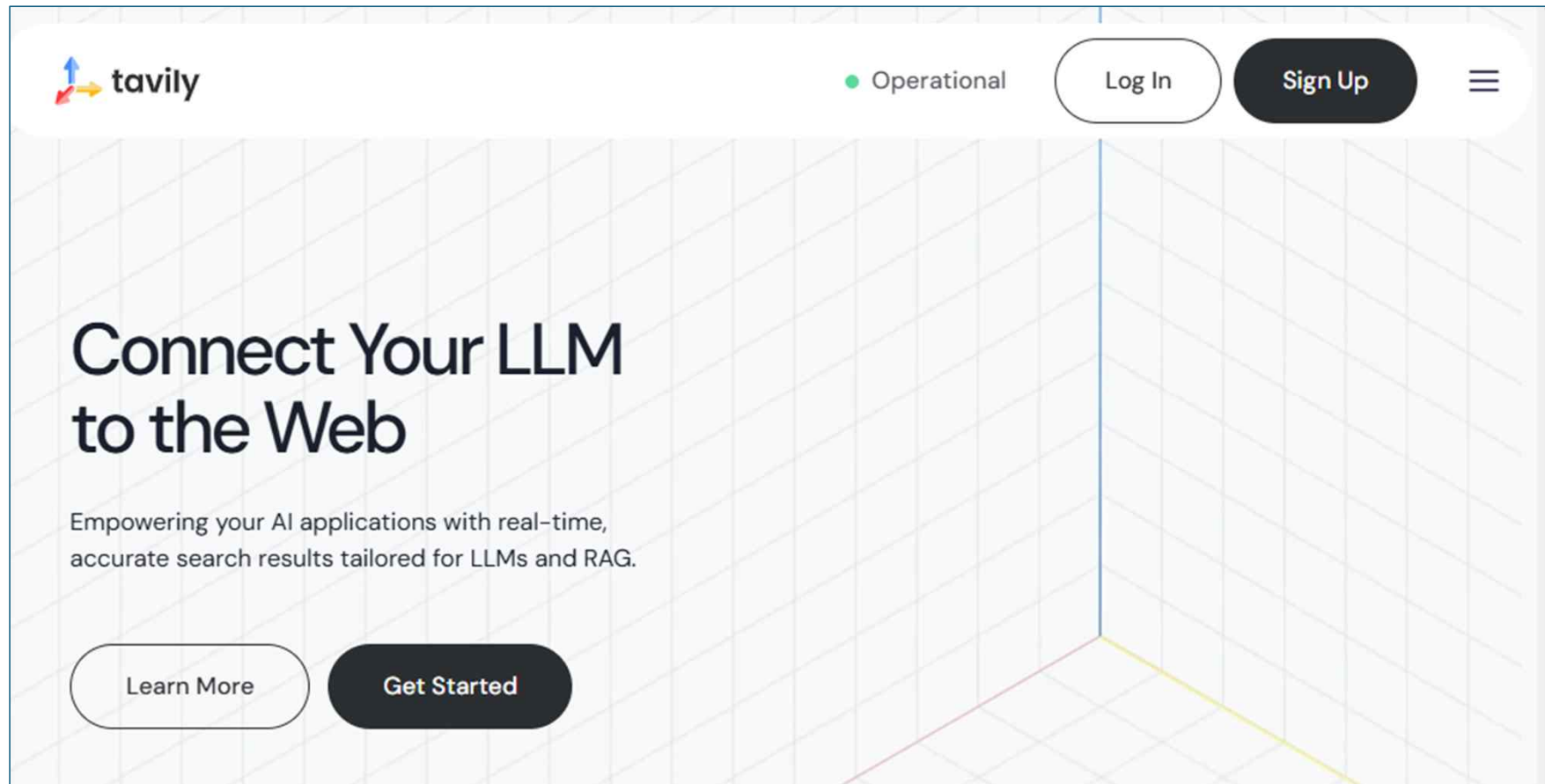
- Chain 구조
- 도구호출 1회



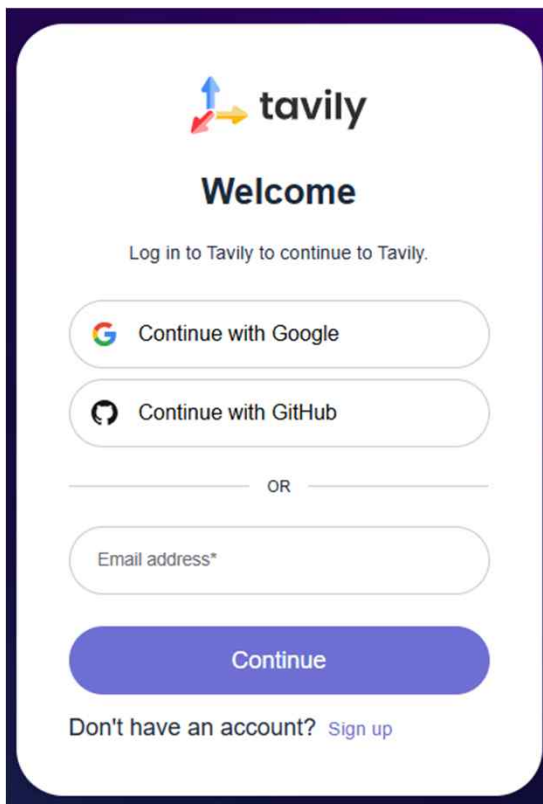
- `create_react_agent()` 혹은 `add_conditional_edges()`로 구현
- 여러 번 도구 호출 가능



Tavily – AI Agent 용 검색 엔진



tavily API Key





The image shows the Tavily welcome screen. At the top is the Tavily logo, which consists of a stylized 't' with a blue arrow pointing up and a red arrow pointing down, followed by the word 'tavily'. Below the logo is the word 'Welcome' in a bold, sans-serif font. Underneath 'Welcome' is the text 'Log in to Tavily to continue to Tavily.' There are two buttons for social login: 'Continue with Google' and 'Continue with GitHub'. Below these is a horizontal line with the word 'OR' in the center. Under the line is a text input field labeled 'Email address*'. Below the input field is a large blue button labeled 'Continue'. At the bottom of the screen is the text 'Don't have an account? Sign up'.

tavily

Welcome

Log in to Tavily to continue to Tavily.

 Continue with Google

 Continue with GitHub

OR





Email address*

Continue

Don't have an account? [Sign up](#)

API Keys +

The key is used to authenticate your requests to the [Research API](#). To learn more, see the [documentation](#) page.

NAME	USAGE	KEY	OPTIONS
default	0	tvly-*****	   

TAVILY_API_KEY=tvly-*****Oh

실습: 060. Langgraph – Agent 구축하기

- 언어 모델과 도구 정의
- Agent 생성 방법
- 스트리밍 메시지
- 메모리가 필요한 질문
- ReAct 패턴의 질문/응답
- [LangSmith trace](#)