



인공지능특론

ARTIFICIAL INTELLIGENCE PRACTICE

Agentic AI III - LangGraph

이건영 교수



 학습목표

- 💡 LangGraph
- 💡 MCP, A2A, Langfuse

 학습목표

- 💡 LLM 개발 프레임워크인 LnagChain의 LangGraph을 활용해 간단한 LLM 프로그램 구현할 수 있다.
- 💡 MCP, A2A, Langfuse 를 이해하고 구현할 수 있다.



TODAY'S LESSON

LANGGRAPH





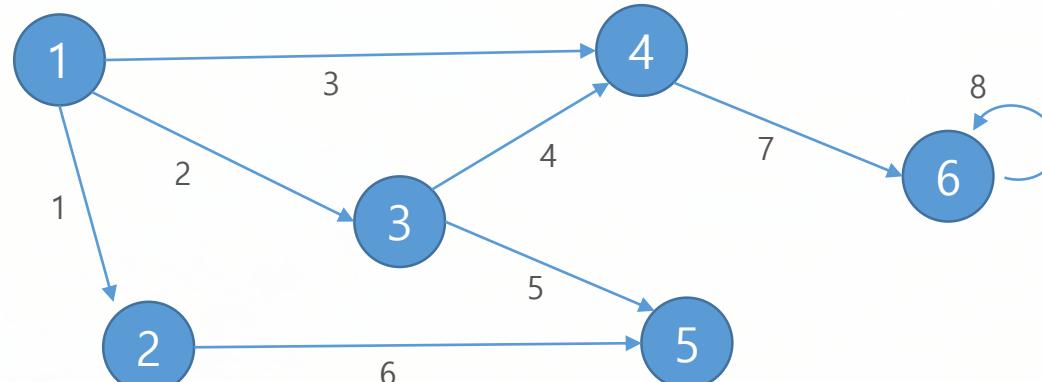
1) LangGraph

Graph Algorithm

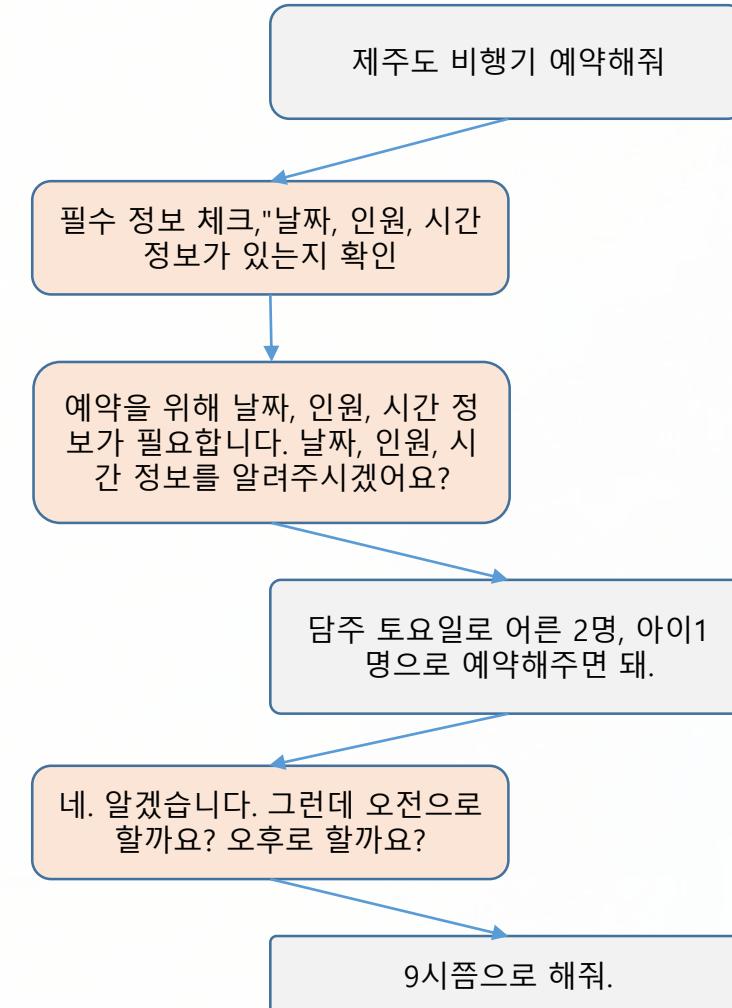
그래프는 정점과 간선으로 구성하는 자료구조를 뜻한다. 상당히 많은 상황을 그래프로 표현할 수 있으며, 많은 알고리즘 문제 역시 그래프로 표현하여 문제를 해결할 수 있다.

- **정점(Vertex , Node)** : Vertex라고도 표현하고 Node라는 용어를 사용하기도 한다. 같은 것이라고 생각하면 되겠다. 정점은 위 그래프에서 동그라미 친 1,2,3,4,5,6 과 같은 각각의 지점을 의미한다.

- **간선(Edge, Link)** : 역시 Edge와 Link 두가지 용어를 사용한다. 그래프에서 각 정점들을 연결하는 선이다. 화살표 표시가 된 것도 있고 그냥 직선으로 표시된 것도 있는데, 둘 다 '간선' 이라고 사용한다.



Directed Graph





1) LangGraph

Graph Node, Edge, State

State : The first thing you do when you define a graph is define the State of the graph. The State consists of the schema of the graph as well as reducer functions which specify how to apply updates to the state.

Node : A unit of execution in which a node receives data, processes it, and returns the result

Edge

- 1. **Normal Edges**: These edges are straightforward as they directly connect one node to another.
- 2. **Conditional Edges**: These edges must process a function to determine which node to go to next based on application conditions and needs.

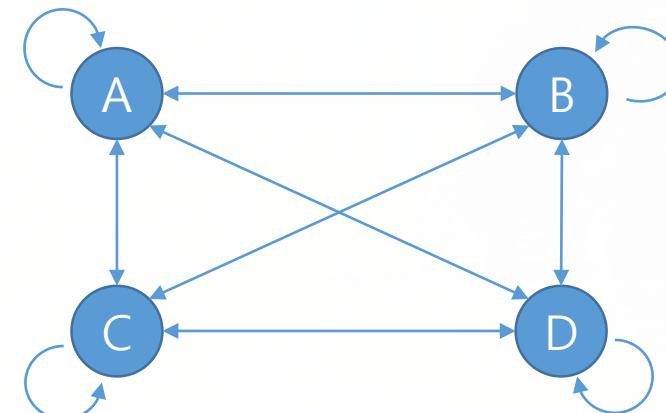




1) LangGraph

LangChain vs LangGraph

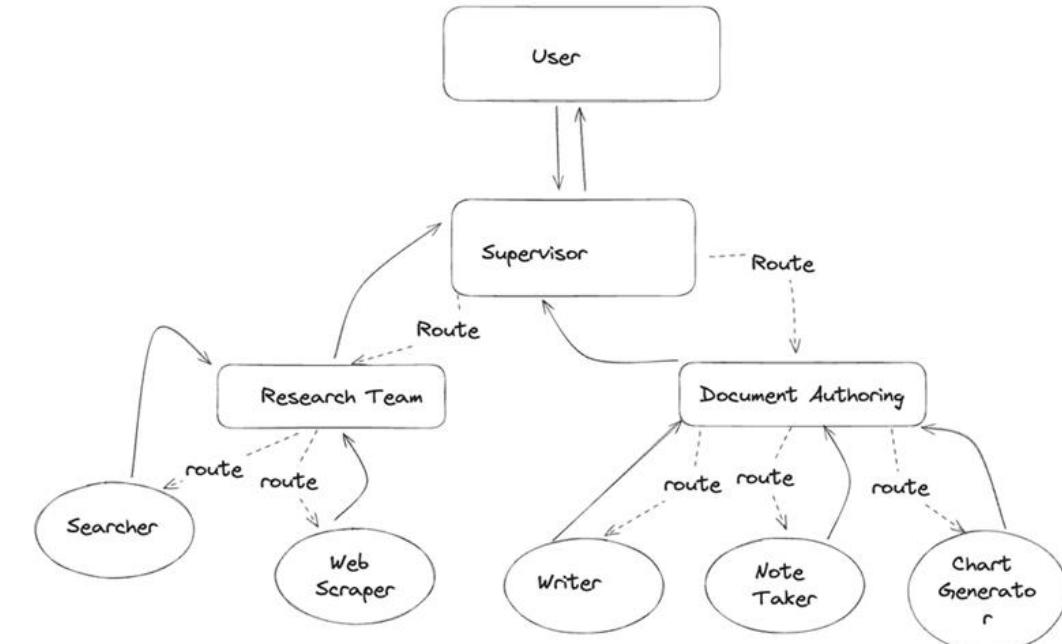
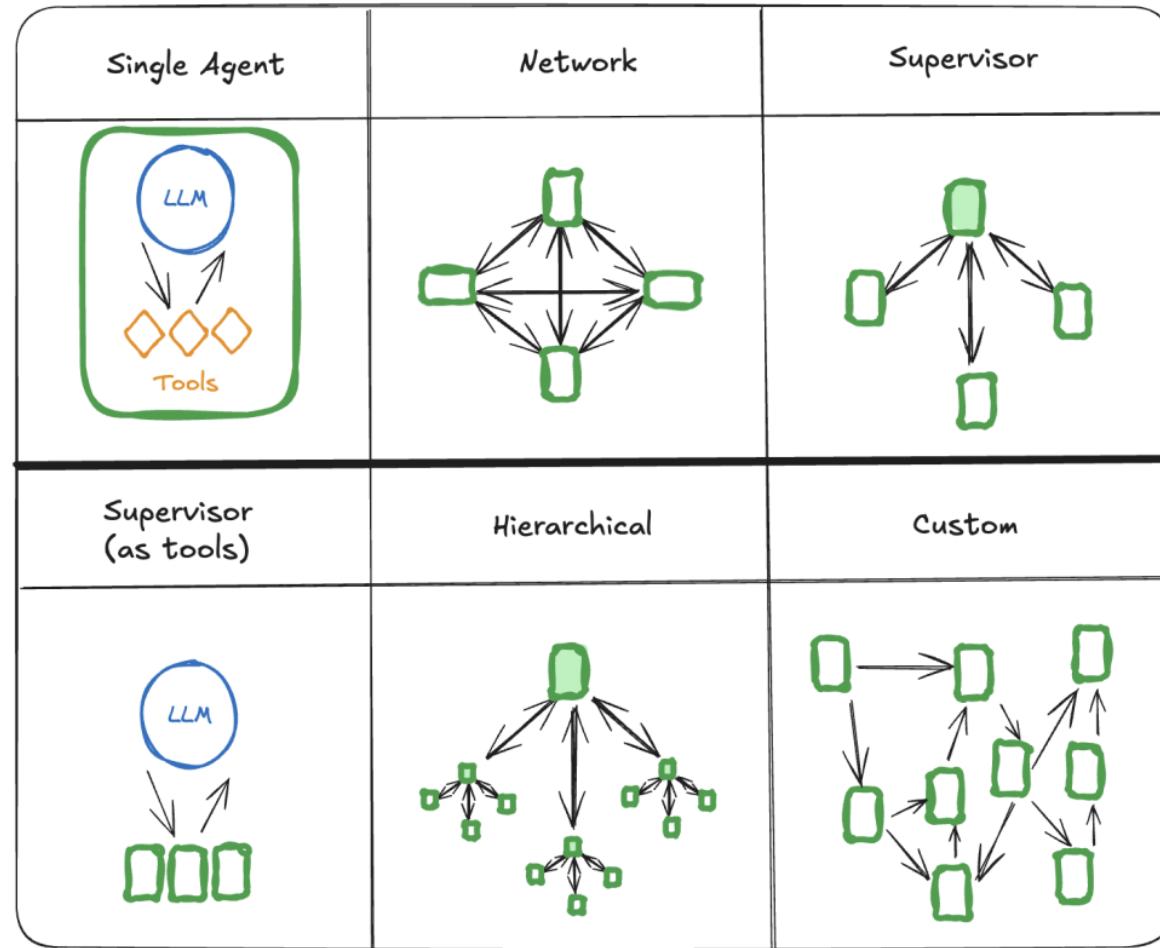
특징	LangGraph	LangChain
주요 목적	복잡한 워크플로우 및 의사결정 프로세스 구현	LLM 통합 및 체인 구성
구조	그래프 기반	체인 및 에이전트 기반
상태 관리	명시적이고 세밀한 제어	암시적이고 자동화된 관리
유연성	높음 (커스텀 로직 쉽게 구현)	중간 (미리 정의된 컴포넌트 중심)
학습 곡선	상대적으로 가파름	상대적으로 완만함
용도	복잡한 AI 시스템, 다중 에이전트	간단한 LLM 애플리케이션, RAG





1) LangGraph

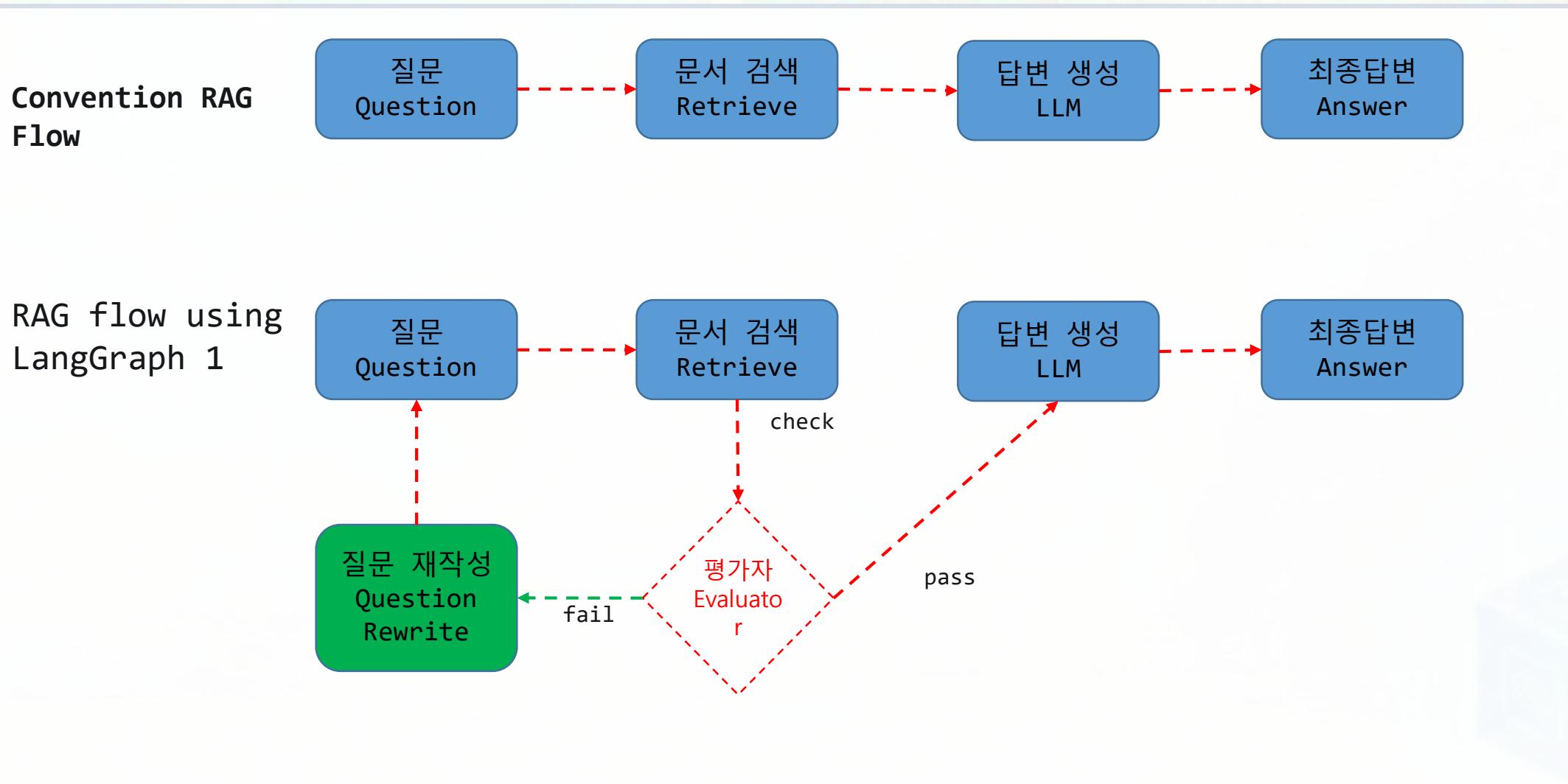
Graph Workflow Types



1) LangGraph



Graph – Evaluator 1/2

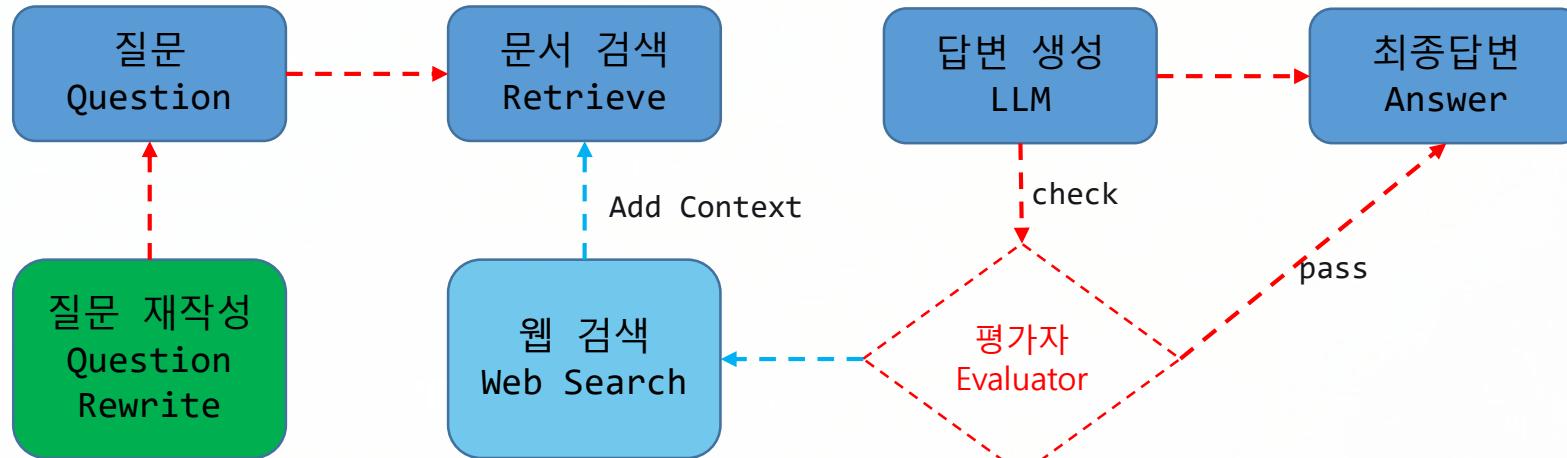


1) LangGraph

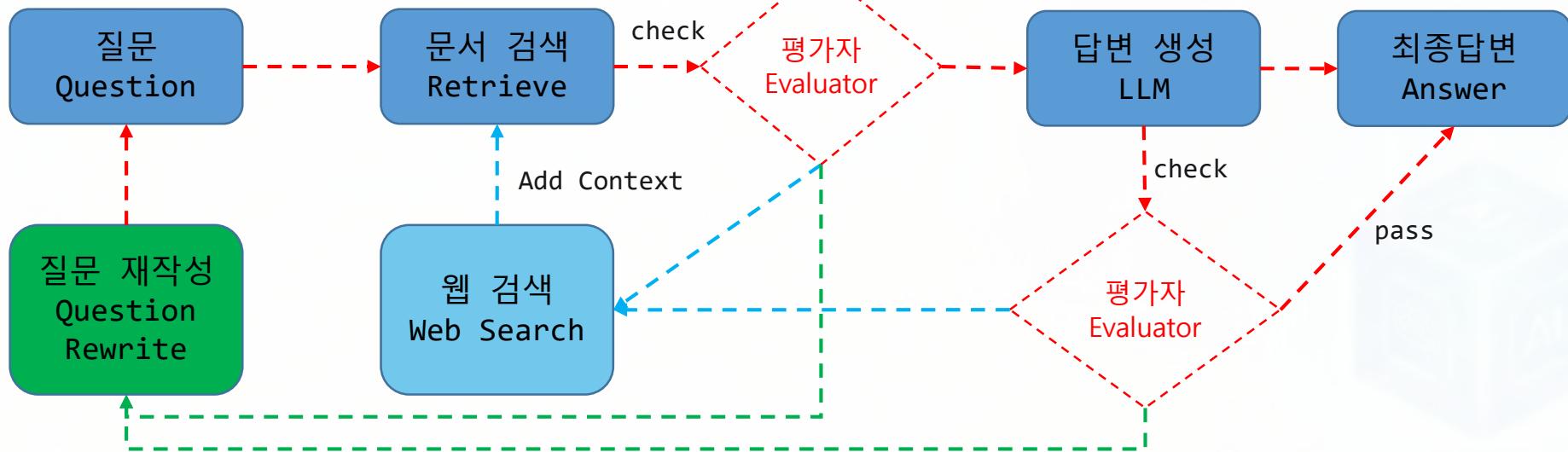


Graph – Evaluator 2/2

RAG flow using
LangGraph 2



RAG flow using
LangGraph 3

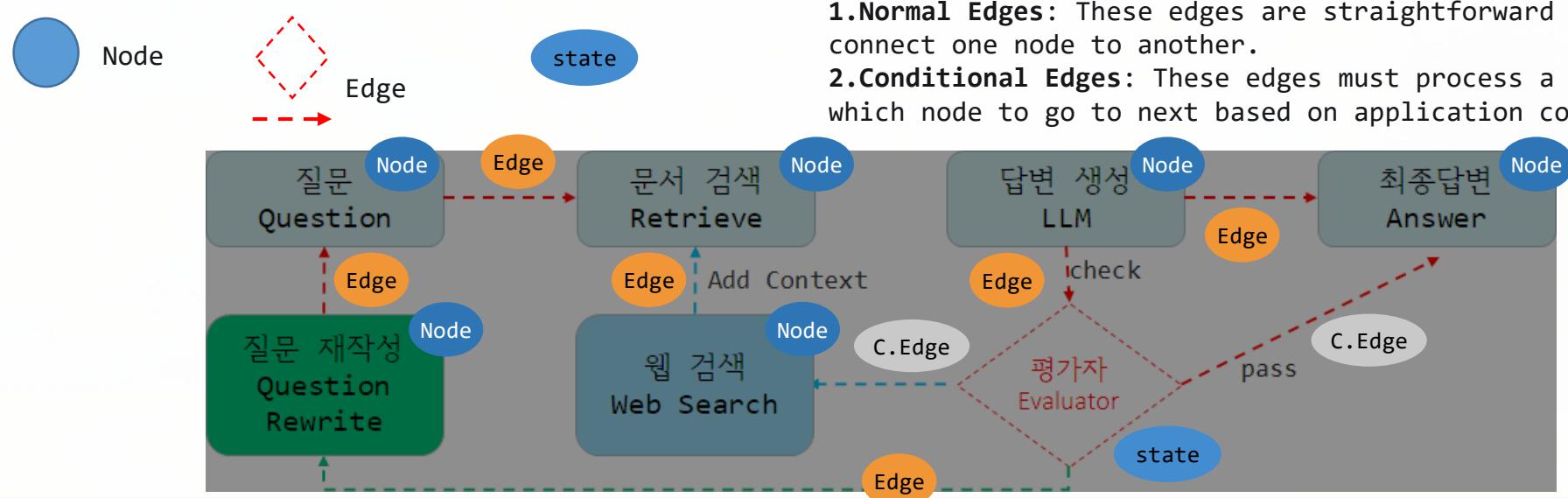


1) LangGraph



Graph Node, Edge, State

요소	정의	역할
State	애플리케이션의 현재 상태를 나타내는 공유 데이터 구조	전체 워크플로우의 컨텍스트 유지; 노드 간 정보 공유; TypedDict 또는 Pydantic BaseModel로 정의
Nodes	실제 작업을 수행하는 Python 함수들	특정 로직 수행 (예: LLM 호출, 도구 사용); 상태 업데이트; 입력으로 현재 상태를 받고, 업데이트된 상태 반환
Edges	노드 간의 연결을 정의하는 요소	워크플로우의 흐름 제어; 조건부 라우팅 가능; 다음에 실행할 노드 결정





1) LangGraph

Example 1_graph_chatbot.py

```

from dotenv import load_dotenv
load_dotenv()

from typing import Annotated, TypedDict
from langgraph.graph.message import add_messages

# State 정의
class State(TypedDict):
    messages: Annotated[list, add_messages]

# from langchain_openai import ChatOpenAI
from langchain_llama import ChatOllama

# LLM 초기화
llm = ChatOllama(model="mistral:latest", temperature=0)

question = "서울의 유명한 맛집 TOP 10 추천해줘"
llm.invoke(question)

```

5m 48s 소요

```

# 챗봇 함수 정의
def chatbot(state: State):
    # 메시지 호출 및 반환
    return {"messages": [llm.invoke(state["messages"])]}

# 그래프 생성 및 노드를 추가
from langgraph.graph import StateGraph, START, END

graph_builder = StateGraph(State)          # 상태 그래프 초기화
graph_builder.add_node("chatbot", chatbot)  # 노드 추가
graph_builder.add_edge(START, "chatbot")     # 시작 노드에서 챗봇 노드로의 엣지 추가
graph_builder.add_edge("chatbot", END)        # 그래프에 엣지 추가
graph = graph_builder.compile()

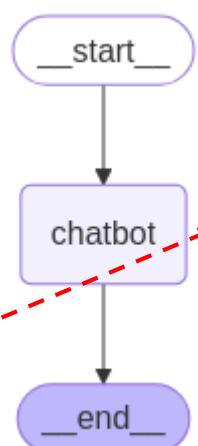
```

그래프 시작화

```

# from IPython.display import Image, display
# display(Image(graph.get_graph().draw_mermaid_png()))

```



AIMessage(content=' 서울에서 유명한 맛집으로 알려진 10가지를 소개하겠습니다. 각 음식점은 다양한 분야와 가격대를 포함하고 있어 여러분의 취향에 따라 선택이 가능합니다.\n\n1. 삼계탕 서울타워 삼계탕 (Samgyetang Seoul Tower Samgyetang)\n- 분야: 한식, 삼계탕\n- 가격대: 중상위\n- 주소: 서울특별시 강남구 역삼동 609-12 34F\n- 전화: +82-2-517-4114\n\n2. 돈까스 맛집 빅오리 (Dakgalbi Bigori)\n- 분야: 한식, 닭갈비\n- 가격대: 저가\n- 주소: 서울특별시 성동구 성수동 127-3\n- 전화: +82-2-546-0999\n\n3. 삼계탕 맛집 삼계탕 옹 (Samgyetang Samgyetang Ong)\n- 분야: 한식, 삼계탕\n- 가격대: 중하위\n- 주소: 서울특별시 용산구 이촌동 12-3\n- 전화: +82-2-794-6055\n\n4. 데지갈비 맛집 빅오리 (Dweji Galbi Bigori)\n- 분야: 한식, 닭갈비\n- 가격대: 저가\n- 주소: 서울특별시 강동구 도봉구 128-3\n- 전화: +82-2-596-7000\n\n5. 삼계탕 맛집 삼계탕 옹\n...')

7m 9.0s 소요

question = "서울의 유명한 맛집 TOP 10 추천해줘"

```

# 그래프 이벤트 스트리밍
for event in graph.stream({"messages": [("user", question)])}):
    # 이벤트 값 출력
    for value in event.values():
        print("Assistant:", value["messages"][-1].content)

```

Assistant:

1. 삼계탕 삼계탕 (Samgyetang Gukjeon): 한국의 대표적인 여름 음식으로, 치킨, 밥, 양념, 당근, 깡샘백등이 담긴 육수에 끓인 삼계탕.
2. 강철집 강철집 (Gangjeoljip Gangjeoljip): 서울특별시 종구 대문로5가 36-10. 전통적인 한국 음식을 맛보실 수 있는 곳으로, 강철집은 고기 조리에 특화되어 있습니다.
3. 서울탕수육 서울탕수육 (Seoul Tangsuyuk): 한국의 대표적인 치킨 요리로, 곁에는 달콤하고 속에는 맛있습니다.
4. 서울시장 서울시장 (Seoul Market): 서울특별시 종구 대문로5가 36-10. 한국의 전통적인 음식을 맛보실 수 있는 곳으로, 여기에서는 다양한 음식을 즐길 수 있습니다.
5. 삼계탕 삼계탕 (Samgyetang Gukjeon): 한국의 대표적인 여름 음식으로, 치킨, 밥, 양념, 당근, 깡샘백등이 담긴 육수에 끓인 삼계탕.
6. 서울탕수육 서울탕수육 (Seoul Tangsuyuk): 한국의 대표적인 치킨 요리로, 곁에는 달콤하고 속에는 맛있습니다.
7. 서울시장 서울시장 (Seoul Market): 서울특별시 종구 대문로5가 36-10. 한국의 전통적인 음식을 맛보실 수 있는 곳으로, 여기에서는 다양한 음식을 즐길 수 있습니다.



1) LangGraph

Example 2_simple_test.py

```
# STATE 정의
from typing_extensions import TypedDict

class State(TypedDict): # TypedDict는 일종의 dictionary
    graph_state: str # 상태(진행중, 다음 작업, 종료)
    goal : str # 목표
    todo : list[str] # 할 일 목록
    current_job : str # 현재 작업
    total_time : int # 총 소요시간(시간)
    time_spend : int # 소요시간(시간)

# NODE 정의
def node_1(state):
    print("---Node 1---")
    return {"graph_state": state['graph_state'] + " I am"}

def node_2(state):
    print("---Node 2---")
    return {"graph_state": state['graph_state'] + " happy!"}

def node_3(state):
    print("---Node 3---")
    return {"graph_state": state['graph_state'] + " sad!"}

# EDGE 정의
import random
from typing import Literal

def decide_mood(state) -> Literal["node_2", "node_3"]:
    user_input = state['graph_state']
    if random.random() < 0.5:

        return "node_2"

    return "node_3"
```

```
# In[8]:
from langgraph.graph import StateGraph, START, END

# Build graph
builder = StateGraph(State)
builder.add_node("node_1", node_1)
builder.add_node("node_2", node_2)
builder.add_node("node_3", node_3)

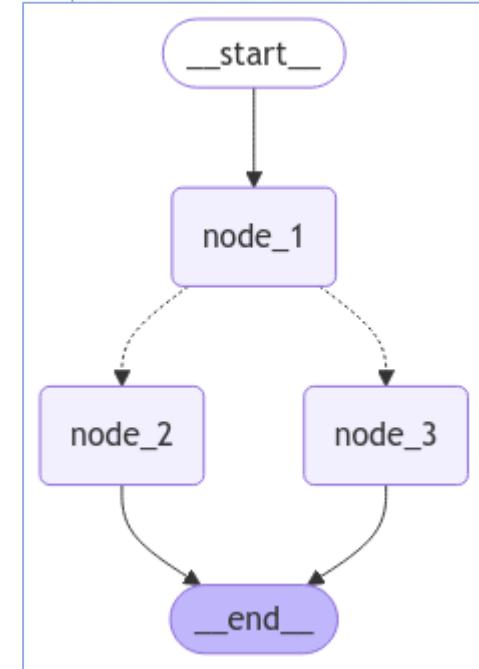
# Logic
builder.add_edge(START, "node_1")
builder.add_conditional_edges("node_1", decide_mood)
builder.add_edge("node_2", END)
builder.add_edge("node_3", END)

# Add
graph = builder.compile()

# View
# from IPython.display import Image, display
# display(Image(graph.get_graph().draw_mermaid_png()))
```

In[9]:
graph.invoke({"graph_state" : "Hi, this is Intel"})

In[10]:
graph.invoke({"graph_state" : "Hi, this is Intel"})



---Node 1---
---Node 2---
{'graph_state': 'Hi, this is Intel I am happy!'}

---Node 1---
---Node 3---
{'graph_state': 'Hi, this is Intel I am sad!'}





1) LangGraph

Example 3_function_call.py - 1/3

```
# function_call_tool.py
import json
from langchain_core.tools import tool

@tool
def get_shipping_info(product:str) -> str:
    """주문한 상품에 대한 배송 정보 조회
    Args:
        product: 상품명"""
    shipping_info = "배송준비중"
    return shipping_info

@tool
def get_grade_info() -> str:
    """유저의 등급 조회"""
    grade = "VIP"
    return grade

@tool
def get_order_info() -> list:
    """주문한 상품 정보 조회"""
    orders = [
        {
            "name": "스트라이프 셔츠",
            "price": "12,000원"
        },
        {
            "name": "트레이닝 바지",
            "price": "35,000원"
        }
    ]
    return orders

tools = [get_order_info, get_shipping_info, get_grade_info]
```

'함수 호출(Function Call)'은 프로그래밍에서 정의된 코드 블록(함수)을 실행하도록 지시하는 행위로, 특히 최근에는 대규모 언어 모델(LLM)이 사용자의 질문에 맞춰 날씨 조회, 데이터베이스 검색, 외부 API 연동 등 실제 동작을 수행하기 위해 미리 정의된 도구(함수)를 스스로 호출하는 기술

```
# 3_function_call.ipynb

from typing import Annotated, TypedDict
from langgraph.graph import StateGraph, START, END
from langgraph.graph.message import add_messages
from langchain_core.messages import HumanMessage
from langchain_ollama import ChatOllama
from langgraph.prebuilt import ToolNode, tools_condition

# function_call_tool.py에서 tools를 가져온다고 가정합니다.
from function_call_tool import tools

# 1. 그래프 상태(State) 정의
class State(TypedDict):
    # add_messages는 기존 메시지 리스트에 새 메시지를 누적(append)해주는 역할을 합니다.
    messages: Annotated[list, add_messages]

# 2. LLM 초기화 및 도구 바인딩
llm = ChatOllama(model="mistral:latest")
# llm = ChatOllama(model="qwen3:8b:latest")
llm_with_tools = llm.bind_tools(tools)

# 3. 노드 함수 정의
def chatbot(state: State):
    """LLM이 상황을 판단하여 도구를 호출하거나 응답합니다."""
    return {"messages": [llm_with_tools.invoke(state["messages"])]}

# 4. 그래프 구축
builder = StateGraph(State)

# 노드 추가
builder.add_node("chatbot", chatbot)
# ToolNode는 도구 실행을 자동화해주는 특수 노드입니다.
builder.add_node("tools", ToolNode(tools))
```



1) LangGraph

Example 3_function_call.py - 2/3

```

# 엣지(흐름) 연결
builder.add_edge(START, "chatbot")

# tools_condition: LLM 응답에 tool_calls가 있으면 "tools" 노드로, 없으면 종료(END)
builder.add_conditional_edges(
    "chatbot",
    tools_condition,
)

# 도구 실행 후 다시 챗봇에게 돌아가 최종 답변을 생성하게 합니다.
builder.add_edge("tools", "chatbot")

# 컴파일
graph = builder.compile()

# from IPython.display import Image, display
# display(Image(graph.get_graph().draw_mermaid_png()))

# 5. 실행 테스트
QUESTION_LIST = [
    "주문 조회해줘",
    "내 등급 조회",
    "스트라이프 셔츠 주문한거 언제와",
    "내 등급이랑 주문정보 알려줘", # multi tool use
    "한국의 수도가 어디야"
]

```

```

def run_langgraph_example():
    for q in QUESTION_LIST:
        print(f"Q: {q}")

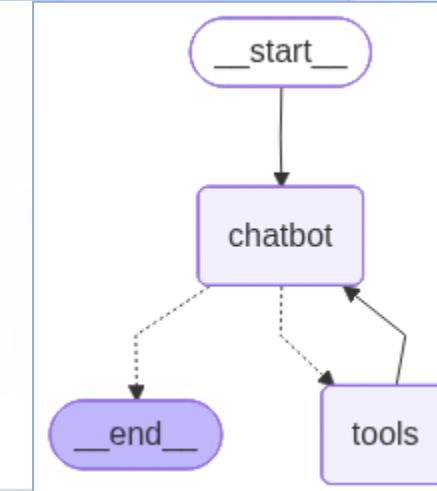
    # 그래프 실행 (초기 메시지 주입)
    initial_state = {"messages": [HumanMessage(content=q)]}

    # stream 모드를 사용하면 내부 과정을 실시간으로 볼 수 있습니다.
    # 여기서는 최종 결과만 출력합니다.
    final_state = graph.invoke(initial_state)

    # 마지막 메시지가 AI의 최종 답변입니다.
    print(f"A : {final_state['messages'][-1].content}\n")
    print("-" * 50)

if __name__ == "__main__":
    run_langgraph_example()

```



1 LangGraph

1) LangGraph



Example 3_function_call.py - 2/3

Q: 주문 조회해줘

A : To get the order information, you can use the following code snippet:

```
```python
get_order_info()
```
```

If you need more specific details like shipping information or grade information, you can call the respective functions as follows:

- Shipping Info:

```
```python
product = "your_product_name" # replace with your product name
get_shipping_info(product=product)
```
```

- Grade Info:

```
```python
get_grade_info()
```
```

Q: 내 등급 조회

A : It seems like your account has a VIP status. Great! Enjoy the perks that come with it. If you need any further assistance, feel free to ask.

Q: 스트라이프 셔츠 주문한거 언제와

A : 제가 저장된 데이터에서 주문한 '스트라이프 셔츠'의 구매일은 없습니다. 따라서 그 정보를 확인하실 수 없습니다.

Q: 내 등급이랑 주문정보 알려줘

A :

Q: 한국의 수도가 어디야

A : Seoul, South Korea는 한국의 수도입니다.

(venv) PS D:\Wgit\W50_AgenticAI\Wgraph> ollama pull qwen3:8b:latest

3_function_call.ipynb

```
# 2. LLM 초기화 및 도구 바인딩
# llm = ChatOllama(model="mistral:latest")
llm = ChatOllama(model="qwen3:8b")
llm_with_tools = llm.bind_tools(tools)
```



15m 8.0s 소요

Q: 주문 조회해줘

A : 주문한 상품 정보입니다:

- **스트라이프 셔츠** - 12,000원
- **트레이닝 바지** - 35,000원

더 궁금한 점이 있으면 알려주세요! ☺

Q: 내 등급 조회

A : 당신의 등급은 VIP입니다.

Q: 스트라이프 셔츠 주문한거 언제와

A : 스트라이프 셔츠 주문은 현재 배송 준비 중입니다. 배송 완료 후 배송 추적 정보를 확인하실 수 있습니다.
추가 문의사항이 있으신가요?

Q: 내 등급이랑 주문정보 알려줘

A : 등급: VIP

주문한 상품 정보:

- 스트라이프 셔츠 - 12,000원
- 트레이닝 바지 - 35,000원

Q: 한국의 수도가 어디야

A : 한국의 수도는 서울입니다.



1 LangGraph

1) LangGraph



Example 4_graph_agent.py 1/3

```
from dotenv import load_dotenv
load_dotenv()

# ### 1. TavilySearchResults 검색 API 도구 사용
from langchain_tavily import TavilySearch

tool = TavilySearch(max_results=3)          # 검색 도구 생성
tools = [tool]                               # 도구 목록에 추가
tool.invoke({"query": "What happened Korean President"})

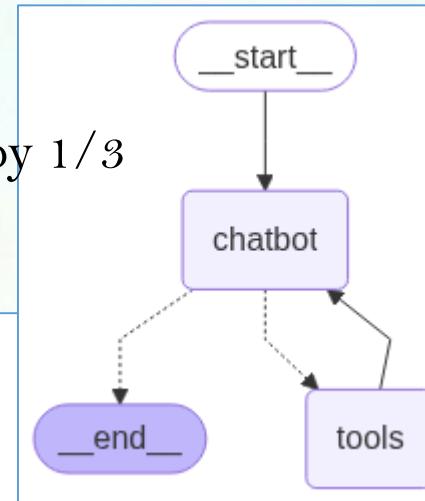
import json
from typing import Annotated
from typing_extensions import TypedDict
from langgraph.graph.message import add_messages

# State 정의
class State(TypedDict):
    messages: Annotated[list, add_messages]

from langchain_ollama import ChatOllama

# LLM 초기화
llm = ChatOllama(model="mistral:latest", temperature=0)
# LLM 에 도구 바인딩
llm_with_tools = llm.bind_tools(tools)

# 노드 함수 정의
def chatbot(state: State):
    answer = llm_with_tools.invoke(state["messages"])
    return {"messages": [answer]} # 자동으로 add_messages 적용
```



간단한 테스트

```
# 그래프 생성 및 노드를 추가
from langgraph.graph import StateGraph
graph_builder = StateGraph(State)           # 상태 그래프 초기화
graph_builder.add_node("chatbot", chatbot)   # 노드 추가

# #### 2. 도구 노드(Tool Node)
# 1) 도구가 호출될 경우 실제로 실행할 수 있는 함수 새로운 노드에 도구를 추가
# 2) 가장 최근의 메시지를 확인하고 메시지에 tool_calls가 포함되어 있으면 도구를 호출하는 BasicToolNode를 구현
# 3) 추후 LangGraph의 pre-built 되어있는 ToolNode로 대체 가능

from langchain_core.messages import ToolMessage

class BasicToolNode:
    """Run tools requested in the last AIMessage node"""

    def __init__(self, tools: list) -> None:
        self.tools_list = {tool.name: tool for tool in tools} # 도구 리스트

    def __call__(self, inputs: dict):
        if messages := inputs.get("messages", []): # 가장 최근 메시지 1개 추출
            message = messages[-1]
        else:
            raise ValueError("No message found in input")

        # 도구 호출 후 결과 저장
        outputs = []
        for tool_call in message.tool_calls:
            tool_result =
                self.tools_list[tool_call["name"]].invoke(tool_call["args"])
            outputs.append(                                # 도구 호출 결과를 메시지로 저장
                ToolMessage(
                    content=json.dumps(
                        tool_result, ensure_ascii=False
                    ), # 도구 호출 결과를 문자열로 변환
                    name=tool_call["name"],
                    tool_call_id=tool_call["id"],
                )
            )
        return {"messages": outputs}
```

1 LangGraph

인공지능특론

1) LangGraph



Example 4_graph_agent.py 1/3

The screenshot shows the Tavily AI Platform's Overview page. It displays a 'Researcher' plan with a monthly plan of 2/1,000 Credits. The API Keys section shows a single key named 'default' of type 'dev'. The sidebar includes links for Overview, API Playground, Use Cases, Billing, Settings, Certification, Documentation, and Tavily MCP.

The screenshot shows a code editor with the file '4_graph_agent.py' open. The code imports dotenv and langchain_tavily, defines a TavilySearch tool, and creates a State class. It then initializes an LLM (ChatOllama) and binds tools to it. Finally, it defines a chatbot function that uses the LLM and tools to respond to messages.

```
from dotenv import load_dotenv
load_dotenv()

# 1. TavilySearchResults 검색 API 도구 사용
from langchain_tavily import TavilySearch

tool = TavilySearch(max_results=3) # 검색 도구 생성
tools = [tool] # 도구 목록에 추가
tool.invoke({"query": "한국의 다음 대통령 선거 일정은 언제야?"}) # 간단한 테스트

import json
from typing import Annotated
from typing_extensions import TypedDict
from langgraph.graph.message import add_messages

# State 정의
class State(TypedDict):
    messages: Annotated[list, add_messages]

from langchain_ollama import ChatOllama

# LLM 초기화
llm = ChatOllama(model="mistral:latest", temperature=0)
# LLM에 도구 바인딩
llm_with_tools = llm.bind_tools(tools)

# 노드 함수 정의
def chatbot(state: State):
    answer = llm_with_tools.invoke(state["messages"])
    return {"messages": [answer]} # 자동으로 add_messages 적용

# 그래프 생성 및 노드를 추가
from langgraph.graph import StateGraph
graph_builder = StateGraph(State) # 상태 그래프 초기화
graph_builder.add_node("chatbot", chatbot) # 노드 추가
```

```
from dotenv import load_dotenv
load_dotenv()

# 1. TavilySearchResults 검색 API 도구 사용
from langchain_tavily import TavilySearch

tool = TavilySearch(max_results=3) # 검색 도구 생성
tools = [tool] # 도구 목록에 추가
tool.invoke({"query": "한국의 다음 대통령 선거 일정은 언제야?"}) # 간단한 테스트

import json
from typing import Annotated
from typing_extensions import TypedDict
from langgraph.graph.message import add_messages

# State 정의
class State(TypedDict):
    messages: Annotated[list, add_messages]

from langchain_ollama import ChatOllama

# LLM 초기화
llm = ChatOllama(model="mistral:latest", temperature=0)
# LLM에 도구 바인딩
llm_with_tools = llm.bind_tools(tools)

# 노드 함수 정의
def chatbot(state: State):
    answer = llm_with_tools.invoke(state["messages"])
    return {"messages": [answer]} # 자동으로 add_messages 적용

# 그래프 생성 및 노드를 추가
from langgraph.graph import StateGraph
graph_builder = StateGraph(State) # 상태 그래프 초기화
graph_builder.add_node("chatbot", chatbot) # 노드 추가
```



1) LangGraph

Example 4_graph_agent.py 2/3

```

# ### 2. 도구 노드(Tool Node)
# 1) 도구가 호출될 경우 실제로 실행할 수 있는 함수 새로운 노드에 도구를 추가
# 2) 가장 최근의 메시지를 확인하고 메시지에 tool_calls가 포함되어 있으면 도구를 호출하는 BasicToolNode를 구현
# 3) 추후 LangGraph의 pre-built 되어있는 ToolNode로 대체 가능
from langchain_core.messages import ToolMessage

class BasicToolNode:
    """Run tools requested in the last AIMessage node"""

    def __init__(self, tools: list) -> None:
        self.tools_list = {tool.name: tool for tool in tools} # 도구 리스트

    def __call__(self, inputs: dict):
        if messages := inputs.get("messages", []): # 가장 최근 메시지 1개 추출
            message = messages[-1]
        else:
            raise ValueError("No message found in input")

        # 도구 호출 후 결과 저장
        outputs = []
        for tool_call in message.tool_calls:
            tool_result =
self.tools_list[tool_call["name"]].invoke(tool_call["args"])
            outputs.append( # 도구 호출 결과를 메시지로 저장
                ToolMessage(
                    content=json.dumps(
                        tool_result, ensure_ascii=False
                    ), # 도구 호출 결과를 문자열로 변환
                    name=tool_call["name"],
                    tool_call_id=tool_call["id"],
                )
            )
        return {"messages": outputs}

```

```

tool_node = BasicToolNode(tools=[tool]) # 도구 노드 생성
graph_builder.add_node("tools", tool_node) # 그래프에 도구 노드 추가

# ### 3. 조건부 엣지(Conditional Edge)
# 1) Edges는 한 노드에서 다음 노드로 제어 흐름을 라우팅
# 2) Conditional edges는 "if" 문을 포함하여 현재 그래프 상태에 따라 다른 노드로 라우팅
# 3) 그래프 state를 받아 다음에 호출할 Node를 나타내는 문자열 또는 문자열 목록을 반환
# 4) 조건은 도구 호출이 있으면 tools로, 없으면 END로 라우팅

from langgraph.graph import START, END

# route_tools라는 라우터 함수를 정의하여 챗봇의 출력에서 tool_calls를 확인
def route_tools(
    state: State,
):
    if messages := state.get("messages", []):
        ai_message = messages[-1] # 가장 최근 메시지를 가져옴
    else:
        # 입력 상태에 메시지가 없는 경우 예외 발생
        raise ValueError(f"No messages found in input state to tool_edge: {state}")

    # AI 메시지에 도구 호출이 있는 경우 "tools" 반환
    if hasattr(ai_message, "tool_calls") and len(ai_message.tool_calls) > 0:
        return "tools" # 도구 호출이 있는 경우 "tools" 반환
    return END # 도구 호출이 없는 경우 END 반환

# tools_condition 함수는 챗봇이 도구 사용을 요청하면 "tools"를 반환하고, 직접 응답이 가능한 경우 "END"를 반환
graph_builder.add_conditional_edges(
    source="chatbot",
    path=route_tools,
    # route_tools의 반환값이 "tools"인 경우 "tools" 노드로, if not END 노드로 라우팅
    path_map={"tools": "tools", END: END},
)

graph_builder.add_edge("tools", "chatbot") # tools > chatbot
graph_builder.add_edge(START, "chatbot") # START > chatbot
graph = graph_builder.compile() # 그래프 컴파일

# from IPython.display import Image, display
# display(Image(graph.get_graph().draw_mermaid_png()))

```

1 LangGraph

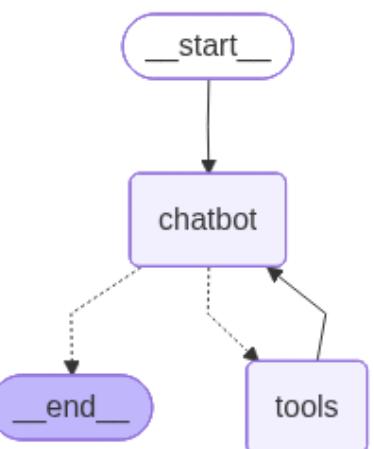
1) LangGraph



Example 4_graph_agent.py 3/3

```
question = "한국의 다음 대통령 선거"

for event in graph.stream({"messages": [("user", question)]}):
    for key, value in event.items():
        print(f"\n=====STEP: {key}=====\n")
        print(value["messages"][-1])
```



{'query': '한국의 다음 대통령 선거 일정은 언제야?',
'follow_up_questions': None, 'answer': None, 'images': [],
'results': [{['url']: '<https://www.korea.kr/news/policyNewsView.do?newsId=148943084>', 'title': '제21대 대통령선거 선거일정 한눈에 보기 – 정책브리핑', 'content': '4. 4.(금)부터 예비후보자 등록 신청, 5. 6.(화) ~ 5. 10.(토) 선거인명부 작성, 5. 10.(토) ~ 5. 11.(일) 후보자등록 신청, 5. 12.', 'score': 0.9987551, 'raw_content': None}, {['url']: '<https://www.youtube.com/watch?v=kXj0Mv0rOvM>', 'title': '6월 3일 제21대 대선…공식 선거운동 5월 12일부터 / KBS 2025.04.09.', 'content': '윤석열 전 대통령 파면으로 치러지는 조기 대선은 6월 3일 화요일로 확정됐습니다. 대통령이 파면되면 60일 이내에 선거를 치러야 하는데, 그 기한의', 'score': 0.97966766, 'raw_content': None}, {['url']: 'https://www.mois.go.kr/frt/bbs/type010/commonSelectBoardArticle.do?bbsId=BBSMSTR_000000000008&nttId=116942', 'title': '제21대 대통령 선거일은 6월 3일(화) – 행정안전부', 'content': '자세한 내용은 첨부를 참고하시기 바랍니다. ☆ 담당자 : 선거의회자치법규과 명노근(044-205-3385). 첨부파일.', 'score': 0.9764905, 'raw_content': None}], 'response_time': 0.67, 'request_id': '55389fc4-1373-4ae4-a896-3909499c2ba3'}

=====

STEP: chatbot

=====

```
content=' [{"name":"tavily_search", "arguments": {"query": "When is the next presidential election in South Korea?", "time_range": "year"}}]' additional_kwargs={} response_metadata={'model': 'mistral:latest', 'created_at': '2026-01-16T23:58:52.224104Z', 'done': True, 'done_reason': 'stop', 'total_duration': 159298632300, 'load_duration': 5757934100, 'prompt_eval_count': 946, 'prompt_eval_duration': 144043443500, 'eval_count': 36, 'eval_duration': 9391586800, 'model_name': 'mistral:latest'} id='run--97623acf-ea24-4fd9-9489-8d2d2d1e6506-0' usage_metadata={'input_tokens': 946, 'output_tokens': 36, 'total_tokens': 982}
```

```
# llm = ChatOllama(model="mistral:latest", temperature=0)
llm = ChatOllama(model="qwen3:8b")
```

=====

STEP: chatbot

=====

```
content=" additional_kwarg={}) response_metadata={'model': 'qwen3:8b', 'created_at': '2026-01-16T23:32:48.7083863Z', 'done': True, 'done_reason': 'stop', 'total_duration': 293833683100, 'load_duration': 3327621000, 'prompt_eval_count': 925, 'prompt_eval_duration': 139804519700, 'eval_count': 479, 'eval_duration': 150380882900, 'model_name': 'qwen3:8b'} id='run--407d20c1-70c3-4957-ab01-36952364984f-0' tool_calls=[{'name': 'tavily_search', 'args': {'query': 'South Korea presidential election schedule 2027', 'search_depth': 'advanced', 'topic': 'general'}, 'id': '148d4fbe-6bc9-4bcf-aacc-234ade5d5e93', 'type': 'tool_call'}] usage_metadata={'input_tokens': 925, 'output_tokens': 479, 'total_tokens': 1404}
```

STEP: tools

content={"query": "South Korea presidential election schedule 2027", "follow_up_questions": null, "answer": null, "images": [], "results": [{"url": "https://www.aei.org/foreign-and-defense-policy/will-washington-be-blindsided-by-south-koreas-june-3-presidential-election/", "title": "Will Washington Be Blindsided by South Korea's June 3 Presidential ...", "content": "South Korea's June 3 snap Presidential election—the “regular” election was originally scheduled for March 2027—was precipitated by the impeachment and removal this past April of its now disgraced former President, Yoon Suk-yeol. Last December, in a bizarre night-time fiasco that lasted just six hours, Yoon—a staunch friend of the United States and former top prosecutor for the ROK government—first declared martial law to suspend constitutional democracy in his country, and then reversed himself in a clown-show climb-down.", "score": 0.9999076, "raw_content": null}, {"url": "https://en.wikipedia.org/wiki/2025_South_Korean_presidential_election", "title": "possible for Yoon to remain in office, chances are very low.", "score": 0.99890125, "raw_content": null}], "response_time": 2.79, "request_id": "414c64c9-c19e-45fd-99a0-865ba04a6c36"}' name='tavily_search' id='30fe4197-832a-47be-9fb8-fd99e609f524' tool_call_id='148d4fbe-6bc9-4bcf-aacc-234ade5d5e93'

STEP: chatbot

content='한국의 다음 대통령 선거는 **2025년 6월 3일**에 실시되었습니다. 이는 원래 2027년 3월에 열릴 예정이었으나, 2024년 4월에 윤석열 대통령이 탄핵되어 대통령 직무가 중단된 후 조기 선거로 일정이 변경되었기 때문입니다. ↪ 현재 (2023년 기준)으로 보자면, 다음 선거는 2025년에 진행되었으며, 이후 **2027년 3월**에 다시 선거가 열릴 예정입니다. 다만, 2025년 선거가 이미 진행된 상황이라, 만약 2025년 이후에 질문을 하신다면 2027년이 다음 선거일입니다.' additional_kwargs={} response_metadata={'model': 'qwen3:8b', 'created_at': '2026-01-16T23:42:44.3970063Z', 'done': True, 'done_reason': 'stop', 'total_duration': 592166750100, 'load_duration': 154744500, 'prompt_eval_count': 1961, 'prompt_eval_duration': 212845273000, 'eval_count': 902, 'eval_duration': 378522884700, 'model_name': 'qwen3:8b'}

1 LangGraph



1) LangGraph



Example 5_multi_agent.py 1/3

```
from dotenv import load_dotenv
from typing import Annotated
from langchain_community.tools.tavily_search import TavilySearchResults
from langchain_core.tools import tool
from langchain_experimental.utilities import PythonREPL

load_dotenv()
tavily_tool = TavilySearchResults(max_results=5)

# This executes code locally, which can be unsafe
repl = PythonREPL()

@tool
def python_repl_tool(
    code: Annotated[str, "The python code to execute to generate your chart."],
):
    """Use this to execute python code and do math. If you want to see the output of a value,
    you should print it out with `print(...)`. This is visible to the user."""
    try:
        result = repl.run(code)
    except BaseException as e:
        return f"Failed to execute. Error: {repr(e)}"
    result_str = f"Successfully executed:\n```\n{code}\n```\\nStdout: {result}"
    return result_str
```

```
##### Create Agent Supervisor
from typing import Literal
from typing_extensions import TypedDict
from langchain_ollama import ChatOllama
from langgraph.graph import MessagesState, END
from langgraph.types import Command

members = ["researcher", "coder"]
options = members + ["FINISH"]

system_prompt = (
    "You are a supervisor tasked with managing a conversation between the"
    f" following workers: {members}. Given the following user request,"
    " respond with the worker to act next. Each worker will perform a"
    " task and respond with their results and status. When finished,"
    " respond with FINISH."
)
class Router(TypedDict):
    """Worker to route to next. If no workers needed, route to FINISH."""
    next: Literal[*options]
    # next: Literal["researcher", "coder", "FINISH"]

llm = ChatOllama(model="qwen3:8b")

class State(MessagesState):
    next: str

def supervisor_node(state: State) -> Command[Literal[*members, "__end__"]]:
    messages = [
        {"role": "system", "content": system_prompt},
    ] + state["messages"]
    response = llm.with_structured_output(Router).invoke(messages)
    goto = response["next"]
    if goto == "FINISH":
        goto = END
    return Command(goto=goto, update={"next": goto})
```

1 LangGraph

1) LangGraph

Example 5_multi_agent.py 2/3

```
##### Construct Graph

from langchain_core.messages import HumanMessage
from langgraph.graph import StateGraph, START, END
from langgraph.prebuilt import create_react_agent

research_prompt = "You are a researcher. DO NOT do any math."
research_agent = create_react_agent(
    llm, tools=[tavily_tool], prompt=research_prompt
)

def research_node(state: State) -> Command[Literal["supervisor"]]:
    result = research_agent.invoke(state)
    return Command(
        update={
            "messages": [HumanMessage(content=result["messages"][-1].content,
name="researcher")]
        },
        goto="supervisor",
    )

# NOTE: THIS PERFORMS ARBITRARY CODE EXECUTION, WHICH CAN BE UNSAFE WHEN NOT SANDBOXED
code_agent = create_react_agent(llm, tools=[python_repl_tool])
```

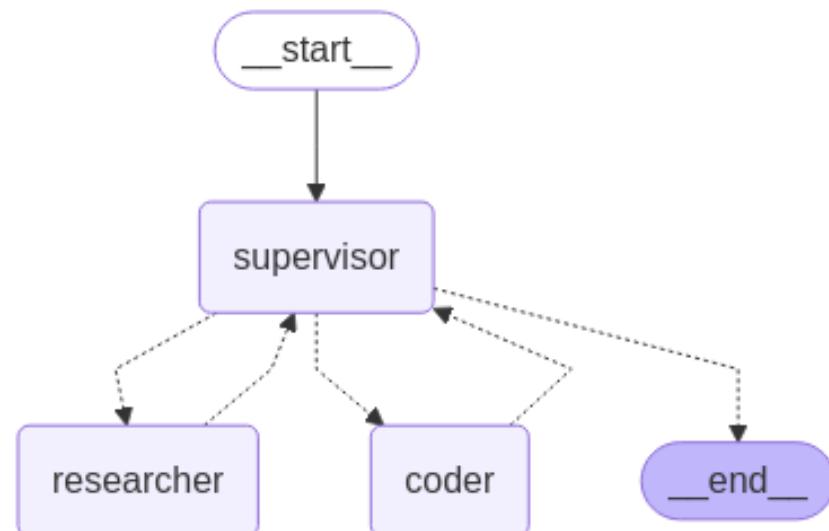


```
def code_node(state: State) -> Command[Literal["supervisor"]]:
    result = code_agent.invoke(state)
    return Command(
        update={
            "messages": [HumanMessage(content=result["messages"][-1].content,
name="coder")]
        },
        goto="supervisor",
    )

builder = StateGraph(State)
builder.add_edge(START, "supervisor")
builder.add_node("supervisor", supervisor_node)
builder.add_node("researcher", research_node)
builder.add_node("coder", code_node)
graph = builder.compile()

# from IPython.display import Image, display
# display(Image(graph.get_graph().draw_mermaid()))
```

```
for s in graph.stream(
    {
        "messages": [
            (
                "user",
                "Find the latest GDP of New York and California, then calculate the
average",
            )
        ],
        subgraphs=True,
    ):
    print(s)
    print("----")
```





1) LangGraph

 Example 5_multi_agent.py 3/3

96m 20.0s 소요

```

(), {'supervisor': {'next': 'researcher'}})
-----
({'researcher:ed8ed0fb-bdce-65bb-54a5-76b03155131c'}, {'agent': {'messages': [AIMessage(content='', additional_kwargs={}, response_meta
-----  

({'researcher:ed8ed0fb-bdce-65bb-54a5-76b03155131c'}, {'tools': {'messages': [ToolMessage(content='[{"title": "Breaking News, Latest Ne
-----  

({'researcher:ed8ed0fb-bdce-65bb-54a5-76b03155131c'}, {'agent': {'messages': [AIMessage(content='', additional_kwargs={}, response_meta
-----  

({'researcher:ed8ed0fb-bdce-65bb-54a5-76b03155131c'}, {'tools': {'messages': [ToolMessage(content='[{"title": "California Remains the W
-----  

({'researcher:ed8ed0fb-bdce-65bb-54a5-76b03155131c'}, {'agent': {'messages': [AIMessage(content="In 2023, **California's nominal GDP**  

-----  

({'researcher': {'messages': [HumanMessage(content="In 2023, **California's nominal GDP** was **$3.9 trillion**, marking a **6.1% gr
-----  

({'supervisor': {'next': 'researcher'}})
-----  

({'researcher:56b26e9b-21d9-0f2e-ef08-7c4700f3b57e'}, {'agent': {'messages': [AIMessage(content='The latest available GDP figures are:\n
-----  

({'researcher': {'messages': [HumanMessage(content='The latest available GDP figures are:\n- **California (2023)**: $3.9 trillion\n-
-----  

({'supervisor': {'next': 'coder'}})
-----  

({'coder:69da1dde-82cc-5082-38b0-c1d5c7c462a3'}, {'agent': {'messages': [AIMessage(content="The latest available GDP figures are:\n- **  

-----  

({'coder': {'messages': [HumanMessage(content="The latest available GDP figures are:\n- **California (2023)**: $3.9 trillion\n- **Ne
-----  

({'supervisor': {'next': 'coder'}})
-----  

({'coder:72529c00-6911-0a42-6da7-7f4b3da982e5'}, {'agent': {'messages': [AIMessage(content="The latest GDP figures available are:\n- **  

-----  

({'coder': {'messages': [HumanMessage(content="The latest GDP figures available are:\n- **California (2023)**: $3.9 trillion\n- **Ne
-----  

({'supervisor': {'next': 'coder'}})
-----  

({'coder:e8fef411-e317-4184-c475-d65e726d0c36'}, {'agent': {'messages': [AIMessage(content="The latest GDP figures available are:\n- **  

-----  

({'coder': {'messages': [HumanMessage(content="The latest GDP figures available are:\n- **California (2023)**: $3.9 trillion\n- **Ne
-----  

({'supervisor': {'next': '__end__'}})
-----
```



1 LangGraph

1) LangGraph



Example 6_plan_and_execute.py 1/3

```
from dotenv import load_dotenv
load_dotenv()

from langchain_ollama import ChatOllama
from langgraph.graph import MessagesState, StateGraph, START, END
from langgraph.prebuilt import create_react_agent

import random
from typing import Annotated, List, Tuple
from typing_extensions import TypedDict

from langchain_community.tools.tavily_search import TavilySearchResults
from langchain_core.tools import tool

l1m = ChatOllama(model="qwen3:8b")

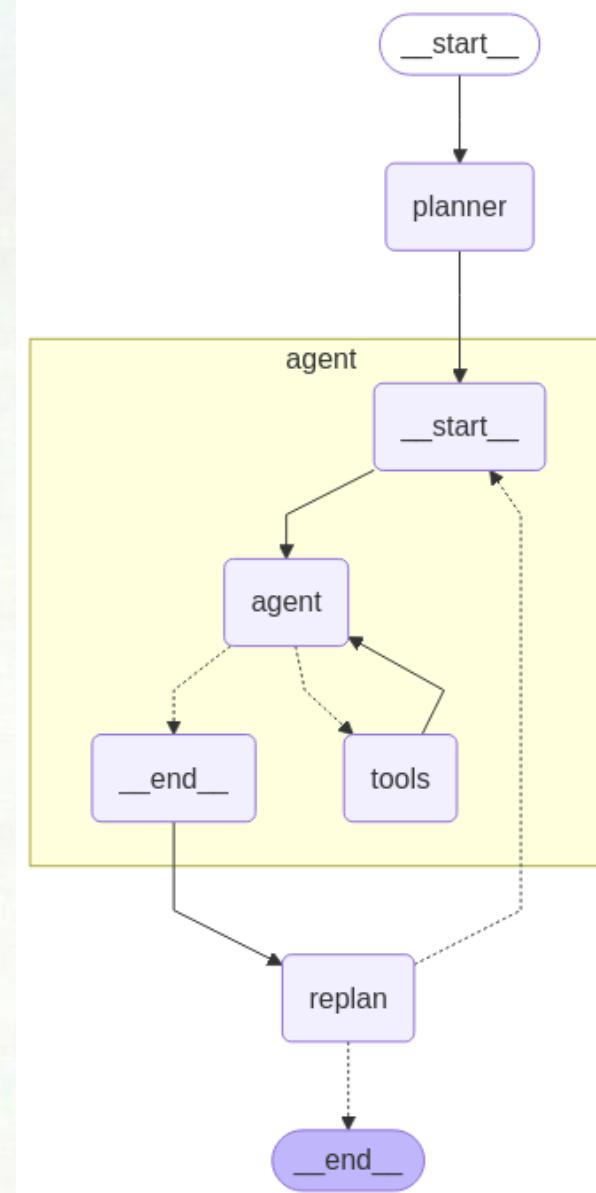
from langchain_community.tools.tavily_search import TavilySearchResults
tools = [TavilySearchResults(max_results=3)]

prompt = "You are a helpful assistant."
agent_executor = create_react_agent(l1m, tools, prompt=prompt)
agent_executor.invoke({"messages": [{"user": "who is the winner of the us open"}]})

import operator

class PlanExecute(TypedDict):
    input: str
    plan: List[str]
    past_steps: Annotated[List[Tuple], operator.add]
    response: str

from pydantic import BaseModel, Field
```



```
agent_executor.invoke({"messages": [{"user": "who is the winner of the us open"}]})  
✓ 20m 15.8s  
{'messages': [HumanMessage(content='who is the winner of the us open', additional_kwargs={}, response_metadata=AIMessage(content='', additional_kwargs={}, response_metadata={'model': 'qwen3:8b', 'created_at': '2026-01-19T12:00:00Z'}), ToolMessage(content='[{"title": "Golf US Open Winners List - Complete Championship History", "url": "https://www.usopen.org/championship-history/golf-us-open"}, {"title": "US Open History - The Official Site of the US Open", "url": "https://www.usopen.org/history"}, {"title": "US Open History - The Official Site of the US Open", "url": "https://www.usopen.org/history"}, {"title": "US Open History - The Official Site of the US Open", "url": "https://www.usopen.org/history"}, {"title": "US Open History - The Official Site of the US Open", "url": "https://www.usopen.org/history"}], response=AIMessage(content='The winner of the US Open depends on the sport, as "US Open" refers to multiple tournaments'))}
```



1) LangGraph

 Example 6_plan_and_execute.py 2/3

```
class Plan(BaseModel):
    """Plan to follow in future"""

    steps: List[str] = Field(
        description="different steps to follow, should be in sorted order"
    )

from langchain_core.prompts import ChatPromptTemplate
planner_prompt = ChatPromptTemplate.from_messages(
    [
        (
            "system",
            """For the given objective, come up with a simple step by step plan. \
This plan should involve individual tasks, that if executed correctly will yield \
the correct answer. Do not add any superfluous steps. \
The result of the final step should be the final answer. Make sure that each step \
has all the information needed - do not skip steps.""",
        ),
        ("placeholder", "{messages}"),
    ]
)
planner = planner_prompt | llm.with_structured_output(Plan)

planner.invoke(
    {
        "messages": [
            ("user", "what is the hometown of the current Australia open winner?")
        ]
    }
)
```

14m 13.0s 소요

Plan(steps=['Identify the most recent Australian Open champion (2023).', 'Determine the hometown of the 2023 Australian Open men's singles winner, Novak Djokovic, which is Belgrade, Serbia.'])

```
from typing import Union

class Response(BaseModel):
    """Response to user."""
    response: str

class Act(BaseModel):
    """Action to perform."""
    action: Union[Response, Plan] = Field(
        description="Action to perform. If you want to respond to user, use Response. \
        If you need to further use tools to get the answer, use Plan."
    )

replanner_prompt = ChatPromptTemplate.from_template(
    """For the given objective, come up with a simple step by step plan. \
This plan should involve individual tasks, that if executed correctly will yield \
the correct answer. Do not add any superfluous steps. \
The result of the final step should be the final answer. Make sure that each \
step has all the information needed - do not skip steps.

Your objective was this:
{input}

Your original plan was this:
{plan}

You have currently done the follow steps:
{past_steps}

Update your plan accordingly. If no more steps are needed and you can return \
to the user, then respond with that. Otherwise, fill out the plan. Only add \
steps to the plan that still NEED to be done. Do not return previously done \
steps as part of the plan."""
)

replanner = replanner_prompt | llm.with_structured_output(Act)

from typing import Literal
from langgraph.graph import END
```

1 LangGraph

1) LangGraph



Example 6_plan_and_execute.py 3/3

```
async def execute_step(state: PlanExecute):
    plan = state["plan"]
    plan_str = "\n".join(f"{i+1}. {step}" for i, step in enumerate(plan))
    task = plan[0]
    task_formatted = f"""For the following plan:
{plan_str}\n\nYou are tasked with executing step {1}, {task}."""
    agent_response = await agent_executor.ainvoke(
        {"messages": [("user", task_formatted)]})
)
return {
    "past_steps": [(task, agent_response["messages"][-1].content)],
}

async def plan_step(state: PlanExecute):
    plan = await planner.ainvoke({"messages": [("user", state["input"])]}))
    return {"plan": plan.steps}

async def replan_step(state: PlanExecute):
    output = await replanner.ainvoke(state)
    if isinstance(output.action, Response):
        return {"response": output.action.response}
    else:
        return {"plan": output.action.steps}

def should_end(state: PlanExecute):
    if "response" in state and state["response"]:
        return END
    else:
        return "agent"

from langgraph.graph import StateGraph, START
```

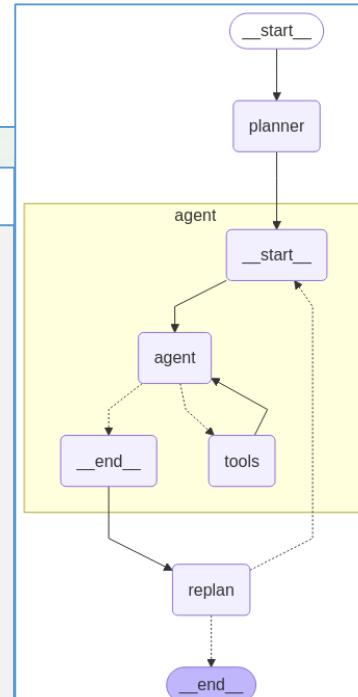
```
workflow = StateGraph(PlanExecute)
workflow.add_node("planner", plan_step)
workflow.add_node("agent", execute_step)
workflow.add_node("replan", replan_step)
workflow.add_edge(START, "planner")
workflow.add_edge("planner", "agent")
workflow.add_edge("agent", "replan")

workflow.add_conditional_edges(
    "replan",
    # Next, we pass in the function that will determine which node is called next.
    should_end,
    ["agent", END],
)
app = workflow.compile()

from IPython.display import Image, display
display(Image(app.get_graph(xray=True).draw_mermaid_png()))

config = {"recursion_limit": 500}
inputs = {"input": "what is the hometown of the mens 2024 Australia open winner?"}
async for event in app.astream(inputs, config=config):
    for k, v in event.items():
        if k != "__end__":
            print(v)
```

{'plan': ["Identify the winner of the men's singles title at the 2024 Australia Open by checking official tournament results or reputable sports news sources.", 'Determine the hometown or place of birth of the identified winner by referencing biographical information from credible sources (e.g., official player profiles, sports databases, or news articles).']},
'past_steps': [{"Identify the winner of the men's singles title at the 2024 Australia Open by checking official tournament results or reputable sports news sources.", "The winner of the men's singles title at the 2024 Australia Open was **Jannik Sinner**. He defeated Daniil Medvedev in the final with a score of 3-6, 3-6, 6-4, 6-4, 6-3. This marked Sinner's first Grand Slam singles title and the first Italian man to win an Australian Open singles title in the Open Era."}],
'plan': ['Determine the hometown or place of birth of the identified winner (Jannik Sinner) by referencing biographical information from credible sources (e.g., official player profiles, sports databases, or news articles).', 'The hometown of Jannik Sinner is **Sorrento**, Italy.'],
'past_steps': [{"Determine the hometown or place of birth of the identified winner (Jannik Sinner) by referencing biographical information from credible sources (e.g., official player profiles, sports databases, or news articles).", 'Jannik Sinner was born in **Innichen** (also known as San Candido), a town in the South Tyrol region of northern Italy. This is confirmed by credible sources such as Wikipedia and Vedantu's biography, which explicitly state his birthplace as Innichen, part of the Italian province of South Tyrol. While his family later resided in **Sexten** (Sesto), his place of birth is clearly documented as Innichen. WnWnThe mention of "Sorrento" in the original plan appears to be an error, as Sorrento is a distinct town in southern Italy (Campania region) and is not associated with Jannik Sinner's biographical data.'}],
'response': "The hometown of the men's 2024 Australia Open winner, Jannik Sinner, is **Innichen** (San Candido), Italy."}



TODAY'S LESSON

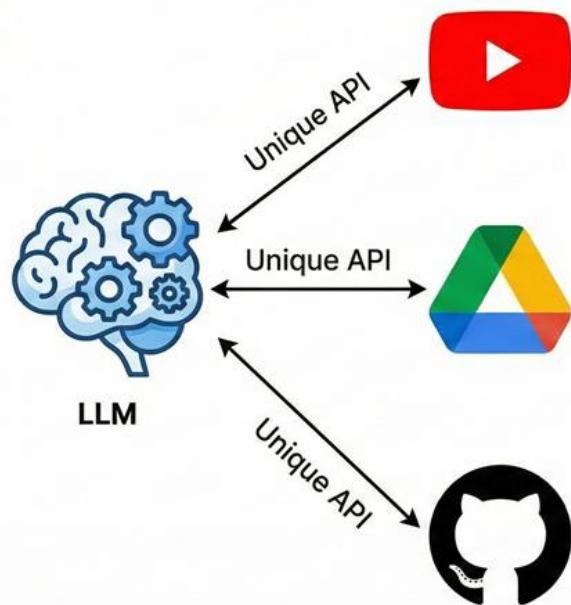
MCP, A2A, Langfuse



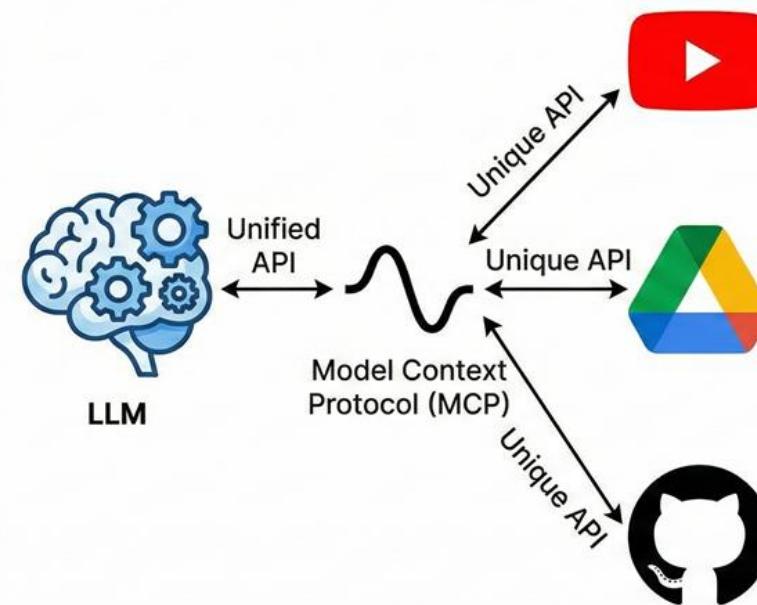
1) MCP

 MCP(Model Context Protocol)는 AI 모델이 외부 도구, 데이터, 애플리케이션과 표준화된 방식으로 소통하고 연결할 수 있도록 돋는 개방형 통신 규약(프로토콜)

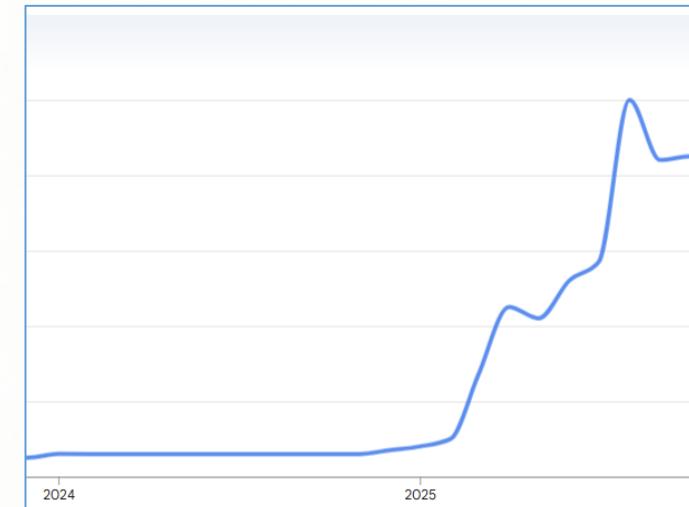
Before MCP



After MCP



핫한 'MCP' 키워드 2025 구글 트렌드



1) MCP

 MCP 종류 확인

<https://github.com/modelcontextprotocol/servers>

<https://smithery.ai/>

Filter by
Search apps...
Apps
Skills
Docs
Pricing
Chat
Publish
Login

 **Youtube** 

youtube

YouTube is a video-sharing platform with user-generated content, live streaming, and monetization opportunities.

 Remote  786

 **Bitbucket** 

bitbucket

Bitbucket is a Git-based code hosting and collaboration platform supporting private and public repositories.

 Remote  643

 **Google Docs** 

googledocs

Google Docs is a cloud-based word processor with real-time collaboration, version history, and integration with...

 Remote  593

 **Miro** 

miro

Miro is a collaborative online whiteboard enabling teams to brainstorm ideas, design wireframes, plan workflows...

 Remote  550

 **Box** 

box

Cloud content management and file sharing service for businesses.

 Remote  513

 **Calendly** 

calendly

Calendly is an appointment scheduling tool that automates meeting invitations, availability checks, and...

 Remote  443

 **Microsoft teams** 

microsoft_teams

Microsoft Teams integrates chat, video meetings, and file storage within Microsoft 365, providing virtual...

 Remote  430

 **LinkedIn** 

linkedin

LinkedIn is a professional networking platform enabling job seekers, companies, and thought leaders to connect,...

 Remote  398

 **Instagram** 

instagram

Instagram is a social media platform for sharing photos, videos, and stories. Only supports Instagram Business...

 Remote  385

 **Canva** 

canva

Canva offers a drag-and-drop design suite for creating social media graphics, presentations, and marketing...

 **AgentMail** 

agentmail

AgentMail is the email inbox API for AI agents. It gives agents their own email inboxes, like Gmail does for...

 **Unicorn or Bust** 

@smithery/unicorn

A choose your own adventure game where you play as a startup founder trying to build a unicorn against all odds.

 Third-Party Servers

 Official Integrations

Official integrations are maintained by companies building production

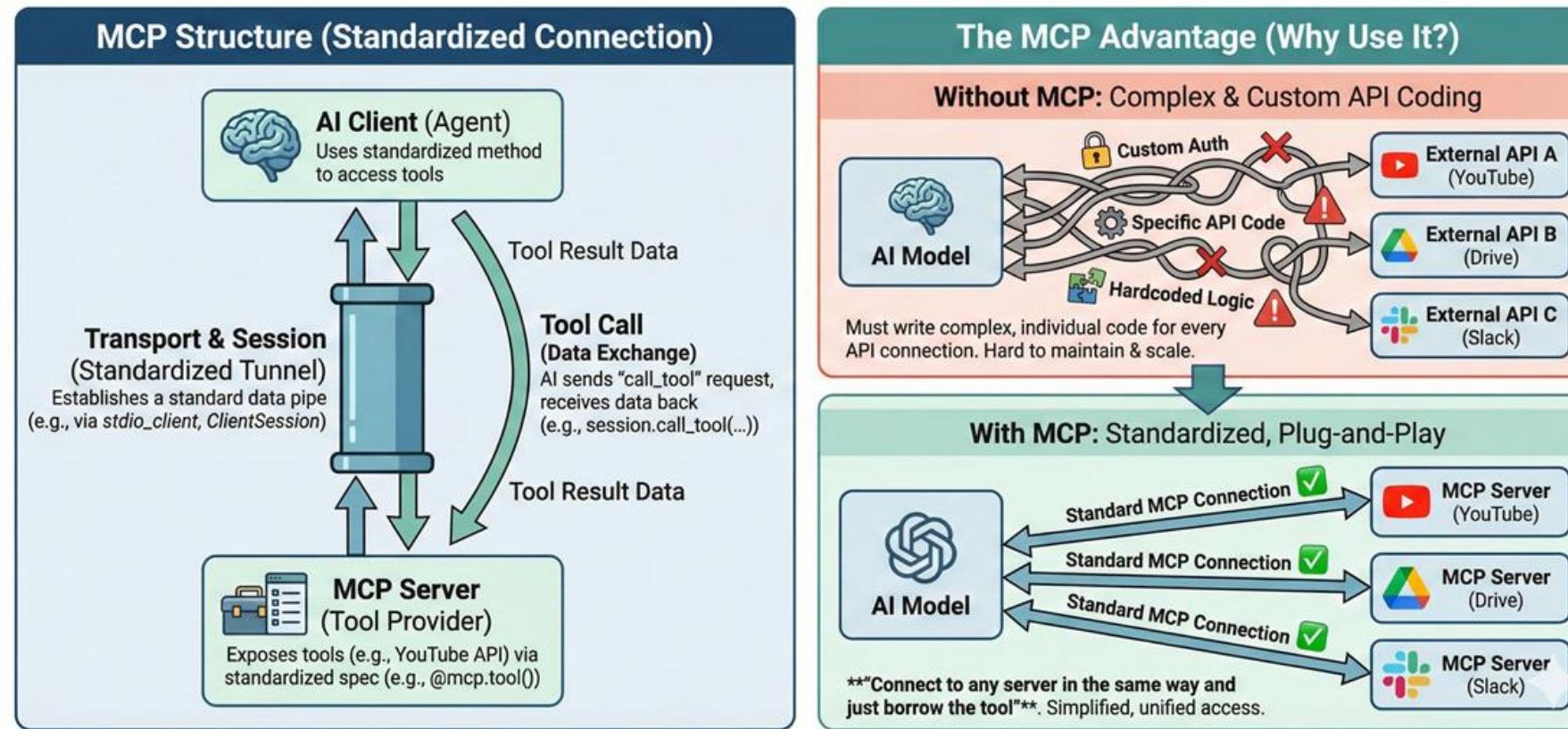
- [21st.dev Magic](#) - Create crafted UI components inspired by the best
- [Adfin](#) - The only platform you need to get paid - all payments in reconciliations with Adfin.
- [AgentQL](#) - Enable AI agents to get structured data from unstructured text.
- [AgentRPC](#) - Connect to any function, any language, across networks.
- [Aiven](#) - Navigate your Aiven projects and interact with the PostgreSQL® services
- [Alation](#) - Unlock the power of the enterprise Data Catalog by having AI serve it.
- [Algolia MCP](#) Algolia MCP Server exposes a natural language interface to indices and configs. Useful for monitoring, debugging and optimizing workflows. See [demo](#).
- [Alibaba Cloud RDS](#) - An MCP server designed to interact with the programmatic management of RDS resources via an LLM.
- [Alibaba Cloud AnalyticDB for MySQL](#) - Connect to a [AnalyticDB for MySQL](#) table metadata, querying and analyzing data. It will be supported to the future.
- [Alibaba Cloud OPS](#) - Manage the lifecycle of your Alibaba Cloud [Service](#) and Alibaba Cloud OpenAPI.
- [Alibaba Cloud OpenSearch](#) - This MCP server equips AI Agents with the ability to search through a standardized and extensible interface.
- [Apache IoTDB](#) - MCP Server for [Apache IoTDB](#) database and its ecosystem.
- [Apify - Actors MCP Server](#): Use 3,000+ pre-built cloud tools to extract data from media, search engines, maps, and more
- [APIMatic MCP](#) - APIMatic MCP Server is used to validate OpenAPI files and processes OpenAPI files and returns validation summaries by leveraging AI.
- [Apollo MCP Server](#) - Connect your GraphQL APIs to AI agents.
- [Arize Phoenix](#) - Inspect traces, manage prompts, curate datasets, and analyze open-source AI and LLM observability tool.
- [Asgardeo](#) - MCP server to interact with your [Asgardeo](#) organization.
- [Astra DB](#) - Comprehensive tools for managing collections and databases with a full range of operations such as create, update, delete, and query.
- [Atlan](#) - The Atlan Model Context Protocol server allows you to interact with your data assets.
- [Audience Insights](#) - Marketing insights and audience analysis from cultural, influencer, and content engagement analysis.
- [AWS](#) - Specialized MCP servers that bring AWS best practices directly to your AI applications.
- [Axiom](#) - Query and analyze your Axiom logs, traces, and all other data in your system.

1) MCP

 MCP(Model Context Protocol) deep dive

Infographic lecture slide

What is MCP? The Standard for Connecting AI to External Tools



1) MCP

MCP(Model Context Protocol) hand on 1/3

```
# youtube_mcp_server.py 소스
import re
import sys
from mcp.server.fastmcp import FastMCP
from youtube_transcript_api import YouTubeTranscriptApi

# [1]. FastMCP 프레임워크 초기화
mcp = FastMCP("YouTube Summarizer") # 서버 이름을 지정하면 클라이언트 연결 시 식별자로 사용

def extract_video_id(url: str) -> str:
    """유튜브 URL에서 비디오 고유 ID(11자리)를 추출하는 유틸리티 함수"""
    pattern = r'(?:(?:v|\/)([0-9A-Za-z_-]{11})|.*'
    match = re.search(pattern, url)
    return match.group(1) if match else None

# [2]. MCP 도구(Tool) 등록
# @mcp.tool() 데코레이터는 아래 함수를 AI가 호출 가능한 '도구'로 변환.
# 함수의 Docstring(주석)은 AI가 "이 도구를 언제 써야 할지" 판단하는 설명서로 필수
@mcp.tool()
def get_youtube_transcript(video_url: str) -> str:
    """유튜브 URL을 입력받아 해당 영상의 자막 전체를 추출합니다."""

    video_id = extract_video_id(video_url)
    if not video_id:
        return "에러: 유효한 유튜브 URL이 아닙니다."
```

```
try:
    ytt_api = YouTubeTranscriptApi() # 인스턴스 생성
    transcript_list = ytt_api.list(video_id) # 자막 목록 조회

    try:
        transcript = transcript_list.find_generated_transcript(['ko'])
    except:
        transcript = list(transcript_list)[0]

    data = transcript.fetch() # 실제 자막 텍스트 데이터 획득

    texts = []
    for item in data:
        if hasattr(item, 'text'):
            texts.append(item.text)
        else:
            texts.append(item['text'])

    full_text = " ".join(texts)

    return full_text if full_text.strip() else "자막 내용이 비어 있습니다."

except Exception as e:
    return f"자막 추출 실패: {str(e)}"

if __name__ == "__main__":
    mcp.run() # MCP 서버 실행
```

1) MCP

MCP(Model Context Protocol) hand on 2/3

```
# agent_mcp.py 소스
import asyncio
import os
import sys
from mcp import ClientSession, StdioServerParameters
from mcp.client.stdio import stdio_client

from langchain_ollama import ChatOllama
from langchain_core.messages import SystemMessage, HumanMessage, ToolMessage

async def run_youtube_agent():
    # [1]. 외부 도구(MCP 서버) 실행을 위한 환경 설정
    current_dir = os.path.dirname(os.path.abspath(__file__))
    server_script = os.path.join(current_dir, "youtube_mcp_server.py")

    # 서버 실행 파라미터: 현재 가상환경의 python으로 서버 파일을 실행하도록 설정
    server_params = StdioServerParameters(
        command=sys.executable,
        args=[server_script],
        env=os.environ.copy()
    )

    try:
        # [2]. MCP 서버 프로세스 연결 (표준 입출력 통로 개설)
        async with stdio_client(server_params) as (read, write):
            async with ClientSession(read, write) as session:
                await session.initialize() # MCP 프로토콜 초기화 (핸드쉐이크)
                print("--- MCP 서버 연결 완료 ---")

                llm = ChatOllama(model="qwen3:8b")
```

```
# [3]. 도구 정의 (LLM에게 "너는 이런 도구를 쓸 수 있어"라고 명세 전달)
# JSON 구조 -> LLM의 도구 이름을 부르고 인자를 채우는 기준
tools_spec = [
    {
        "name": "get_youtube_transcript",
        "description": "유튜브 비디오 URL에서 자막을 가져옵니다.",
        "parameters": {
            "type": "object",
            "properties": {"video_url": {"type": "string"}},
            "required": ["video_url"]
        }
    }
]

llm_with_tools = llm.bind_tools(tools_spec) # 모델 도구 정보를 바인딩

youtube_url = "https://www.youtube.com/watch?v=fToUPQ_WRaY"
messages = [
    SystemMessage(content=(
        "너는 유튜브 분석 전문가야."
        "사용자가 링크를 주면 무조건 'get_youtube_transcript' 도구를 사용해 자막을 먼저 가져와."
        "자막 데이터가 없으면 절대로 요약하지 말고 모른다고 답해."
    )),
    HumanMessage(content=f"이 영상 한글로 요약해줘: {youtube_url}")
]

# [4]. [에이전트 루프 - 1단계] 추론(Reasoning)
# LLM이 질문을 받고 "도구를 실행해야겠다"고 결정하는 단계
print("\n[에이전트 생각 중...]")
ai_msg = await llm_with_tools.invoke(messages)
messages.append(ai_msg)
```

용해 자막을 먼저 가져와.

1) MCP

 MCP(Model Context Protocol) hand on 3/3

```

# [5]. [에이전트 루프 - 2단계] 행동(Action)
# LLM이 내린 도구 실행 명령(tool_calls)이 있는지 확인
if ai_msg.tool_calls:
    for tool_call in ai_msg.tool_calls:
        print(f"--- 도구 실제 실행 중: {tool_call['name']} ---")

        # MCP 서버 세션을 통해 실제로 자막 데이터를 긁어옴
        mcp_result = await session.call_tool(
            tool_call["name"],
            arguments=tool_call["args"]
        )

        # 도구로부터 받은 자막 원문을 대화 기록에 추가 (Observation)
        transcript_text = mcp_result.content[0].text
        messages.append(ToolMessage(
            content=transcript_text,
            tool_call_id=tool_call["id"]
        ))

```

[6]. [에이전트 루프 - 3단계] 최종 답변(Final Answer)

획득한 자막 데이터를 바탕으로 LLM이 다시 요약을 수행

```

print("[최종 답변 생성 중...]")
final_answer = await llm_with_tools.invoke(messages)

print("\n" + "="*50)
print("[최종 요약 결과]\n")
print(final_answer.content)
print("=".join(["="]*50))
else:
    # 도구 호출을 안 하고 대답하는 경우(Hallucination)에 대한 예외 처리
    print("\n⚠️ AI가 도구를 호출하지 않고 답변했습니다:")
    print(ai_msg.content)

```

```

except Exception as e:
    print(f"⚠️ 예상 발생: {e}")

if __name__ == "__main__":
    # Windows에서 비동기 입출력을 지원하기 위한 이벤트 루프 정책 설정
    if sys.platform == 'win32':
        asyncio.set_event_loop_policy(asyncio.WindowsSelectorEventLoopPolicy())
        asyncio.run(run_youtube_agent())

```

12m 40.0s 소요

```
(venv) PS D:\git\W50_AgenticAI\etc> python .\Wagent_mcp.py
-- 도구 실제 실행 중: get_youtube_transcript --
[최종 답변 생성 중...]
=====
[최종 요약 결과]
```

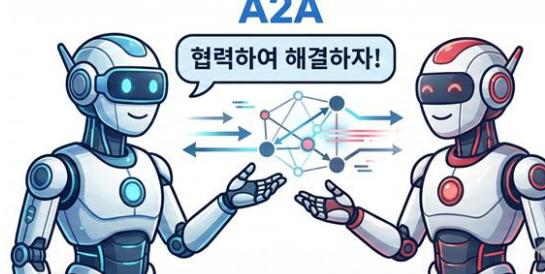
영상 요약 (한글):

이 영상은 파이브레인 에듀키트 판매 및 개발 관련 내용을 다룹니다. 제작자는 새 사무실을 역삼동으로 이전했으며, DIY 키트를 통해 로봇 제어 시스템을 판매하고 있습니다. 키트에는 리더암, 팔로암, 카메라 마운트, 카메라 등이 포함되어 있어 해외 직구보다 편리하며, KC 인증을 받은 안전한 어댑터도 제공합니다.

또한, AI 모델을 활용한 실시간 모션 제어 시스템을 개발 중이며, 3D 프린터로 부품을 빠르게 제작해 연구 소 작업을 지원합니다. 연구원 모집도 진행 중으로, 로봇 연구에 관심 있는 학생 및 전문가들이 자유롭게 참여할 수 있습니다. 마지막으로, 키트 구성품과 조립 방법, 모터 사양 등 구체적인 정보도 안내하고 있습니다.

2) A2A

A2A(Agent-to-Agent)는 AI 에이전트들이 서로 소통하고 협력하기 위한 구조

| MCP | A2A |
|--|---|
| 외부 도구와 연결 Protocol | AI 간 협업 패턴 |
|  <p>MCP</p> <ul style="list-style-type: none"> - Single Agent tasks - Tool Integration - External data access - JSON-RPC 기반 |  <p>A2A</p> <ul style="list-style-type: none"> - Multi Agent collaboration - Cross-platform workflows - Specialized agent coordination |
| https://smithery.ai/ | https://a2acatalog.com/agents |
| Anthropic (Claude) | 구글(Google) |

2) A2A

 A2A(Agent-to-Agent)

<https://a2acatalog.com/agents>

A2A Catalog Work Tools Agents Workflows About Sign In Get Started

A2A Agent Catalog

Discover and integrate A2A-compliant AI agents for seamless interoperability. Browse the most comprehensive collection of AI agents supporting the Agent-to-Agent protocol from AutoGen, LangGraph, CrewAI, LlamaIndex, Semantic Kernel, and more frameworks.

Search agents, skills, or descriptions...
Filters
Most Popular
grid icon
list icon

100 agents found

Categories

- Accessibility
- Audio Processing
- Business Intelligence
- Communication
- Content Generation
- Creative
- Customer Service
- Data & Analytics
- Development
- Environment
- Finance
- Image Processing
- Language
- Marketing
- Other
- Productivity
- Research
- Social Media

A2A Python Official SDK 

Google

Official Python SDK for the Agent2Agent Protocol

[Python SDK](#) [Official implementation](#) [A2A protocol](#)

35 0 0 

A2A JavaScript SDK 

Google A2A

Official JavaScript SDK for running agentic applications as A2A servers

[JavaScript SDK](#) [TypeScript](#) [A2A server](#)

30 0 0 

Elkar Task Manager 

Elkar AI

Open-source task management layer for AI agents based on A2A...

[Task orchestration](#) [Agent coordination](#) [Workflow management](#)

30 0 0 

tRPC A2A Go 

tRPC Group

Go implementation by tRPC team with full client/server support and...

[Go implementation](#) [Multiple auth](#)

Hello World Agent 

Google A2A Samples

Basic example agent for getting started

[Getting started](#) [Basic example](#) [Tutorial](#)

Google Calendar Agent 

Inference Gateway

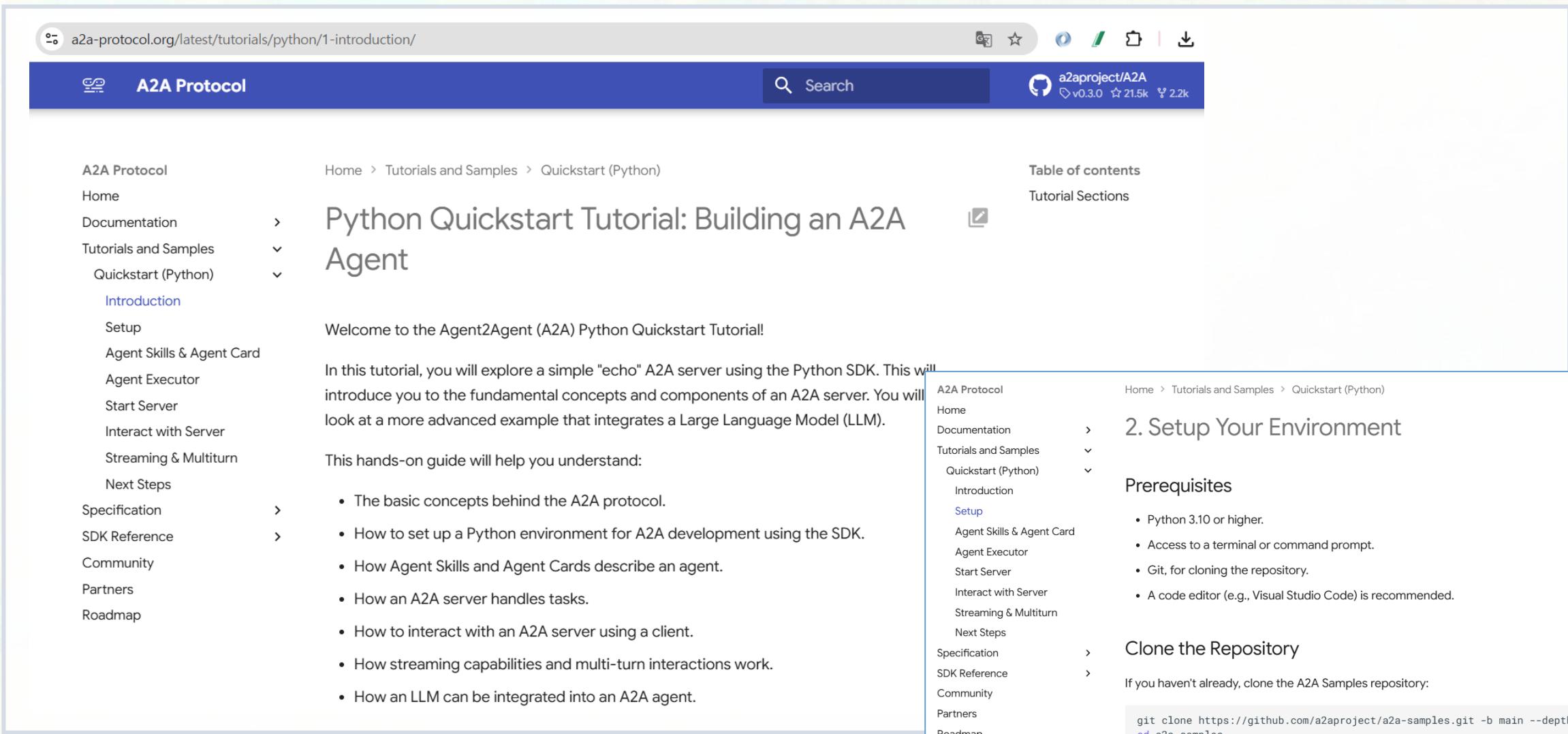
Standalone A2A agent for comprehensive Google Calendar...

[Google Calendar](#) [OpenAI API](#) [Calendar management](#)

2) A2A

A2A(Agent-to-Agent)

<https://a2a-protocol.org/latest/tutorials/python/1-introduction/>



The screenshot shows a browser window displaying the A2A Protocol Python Quickstart Tutorial. The URL in the address bar is <https://a2a-protocol.org/latest/tutorials/python/1-introduction/>. The page title is "Python Quickstart Tutorial: Building an A2A Agent". The left sidebar contains a navigation menu with sections like Home, Documentation, Tutorials and Samples (with sub-sections Quickstart (Python), Introduction, Setup, Agent Skills & Agent Card, Agent Executor, Start Server, Interact with Server, Streaming & Multiturn, Next Steps, Specification, SDK Reference, Community, Partners, and Roadmap). The main content area starts with a welcome message: "Welcome to the Agent2Agent (A2A) Python Quickstart Tutorial!". It then describes the tutorial's purpose: "In this tutorial, you will explore a simple 'echo' A2A server using the Python SDK. This will introduce you to the fundamental concepts and components of an A2A server. You will look at a more advanced example that integrates a Large Language Model (LLM)". Below this, it lists what the hands-on guide will help understand: basic concepts behind the A2A protocol, setting up a Python environment for development, how Agent Skills and Agent Cards describe an agent, how an A2A server handles tasks, interacting with a server using a client, streaming capabilities, multi-turn interactions, and integrating an LLM into an A2A agent. To the right, there is a sidebar with a "Table of contents" and "Tutorial Sections" section, which lists the same navigation menu items. Below the sidebar, there are two main sections: "2. Setup Your Environment" and "Clone the Repository". The "Setup Your Environment" section lists prerequisites: Python 3.10 or higher, access to a terminal or command prompt, Git for cloning the repository, and a code editor like Visual Studio Code. The "Clone the Repository" section provides the command: `git clone https://github.com/a2aproject/a2a-samples.git -b main --depth 1 cd a2a-samples`.

2) A2A

 A2A(Agent-to-Agent) hands on 1 - quickstart

<https://a2aproject.github.io/A2A/latest/tutorials/python/1-introduction/>

```
PS D:\GITLAB\text2SQL> git clone https://github.com/google-a2a/a2a-samples.git -b main --depth 1
PS D:\GITLAB\text2SQL> cd a2a-samples
PS D:\GITLAB\text2SQL\aa2a-samples> python -m venv .venv
PS D:\GITLAB\text2SQL\aa2a-samples> .\venv\Scripts\activate
(.venv) PS D:\GITLAB\text2SQL\aa2a-samples> pip install -r samples/python/requirements.txt

# install error 발생시
(.venv) PS D:\GITLAB\text2SQL\aa2a-samples> pip install uv
(.venv) PS D:\GITLAB\text2SQL\aa2a-samples> uv pip install -r samples/python/requirements.txt
(.venv) PS D:\GITLAB\text2SQL\aa2a-samples> pip install -r samples/python/requirements.txt

# 셋팅 테스트
(.venv) PS D:\GITLAB\text2SQL\aa2a-samples> python -c "import a2a; print('A2A SDK imported successfully')"
A2A SDK imported successfully

# 서버 구동
(.venv) PS D:\GITLAB\text2SQL\aa2a-samples> python samples/python/agents/helloworld/__main__.py
INFO:     Started server process [25076]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     Uvicorn running on http://0.0.0.0:9999 (Press CTRL+C to quit)

# client 구동
PS D:\GITLAB> cd .\text2SQL\aa2a-samples
PS D:\GITLAB\text2SQL\aa2a-samples> .venv\Scripts\activate
(.venv) PS D:\GITLAB\text2SQL\aa2a-samples> python samples/python/agents/helloworld/test_client.py
INFO:__main__:Attempting to fetch public agent card from: http://localhost:9999/.well-known/agent.json
INFO:htpx:HTTP Request: GET http://localhost:9999/.well-known/agent.json "HTTP/1.1 200 OK"
INFO:a2a.client.client:Successfully fetched agent card data from http://localhost:9999/.well-known/agent.json: {'capabilities': {'streaming': True}, 'defaultInputModes': ['text'], 'defaultOutputModes': ['text'], 'description': 'Just a hello world agent', 'name': 'Hello World Agent', 'skills': [{'description': 'just returns hello world', 'examples': ['hi', 'hello world'], 'id': 'hello_world', 'name': 'Returns hello world', 'tags': ['hello world']}]}
```

```
'supportsAuthenticatedExtendedCard': True, 'url': 'http://localhost:9999/', 'version': '1.0.0'
INFO:__main__:Successfully fetched public agent card:
INFO:__main__:
  "capabilities": {
    "streaming": true
  },
  "defaultInputModes": [
    "text"
  ],
  "defaultOutputModes": [
    "text"
  ],
  "description": "Just a hello world agent",
  "name": "Hello World Agent",
  "skills": [
    {
      "description": "just returns hello world",
      "examples": [
        "hi",
        "hello world"
      ],
      "id": "hello_world",
      "name": "Returns hello world",
      "tags": [
        "hello world"
      ]
    }
  ],
  "supportsAuthenticatedExtendedCard": true,
  "url": "http://localhost:9999/",
  "version": "1.0.0"
}
INFO:__main__:
Using PUBLIC agent card for client initialization (default).
INFO:__main__:
Public card supports authenticated extended card. Attempting to fetch from:
http://localhost:9999/agent/authenticatedExtendedCard
INFO:htpx:HTTP Request: GET http://localhost:9999/agent/authenticatedExtendedCard "HTTP/1.1 200 OK"
INFO:a2a.client.client:Successfully fetched agent card data from
http://localhost:9999/agent/authenticatedExtendedCard: {'capabilities': {'streaming': True}, 'defaultInputModes': ['text'], 'defaultOutputModes': ['text'], 'description': 'The full-feature hello world agent for authenticated users.', 'name': 'Hello World Agent - Extended Edition', 'skills': [{"description": "just returns hello world", "examples": ["hi", "hello world"], "id": "hello_world", "name": "Returns hello world", "tags": ["hello world"]}], {"description": "A more enthusiastic response for authenticated users.", "examples": ["super hi", "give me a super hello"], "id": "super_hello_world", "name": "Returns a SUPER Hello World", "tags": ["hello world", "super"]}], "version": "1.0.0"}
```

2

MCP, A2A, Langfuse

2) A2A

A2A(Agent-to-Agent) hands on 2 – 환전, 비용처리 Agent 1/5

어떤 통화에서 한국 돈으로 환전하고 싶으신가요? 미국 달러(USD)를 예로 들어 설명해 주시면 됩니다. 또한, 특정 날짜의 환율을 원하시면 날짜를 알려주세요. 그렇지 않으면 최신 환율을 사용하겠습니다.

한달전, USD

정확한 날짜를 알려주시겠어요? 한 달 전의 정확한 날짜를 알아야 환율을 검색할 수 있습니다.

2025-06-01

2025년은 아직 오지 않았습니다. 다른 날짜를 입력해 주시겠어요?

2024-06-01

2024-06-01 기준으로 20 USD는 27665.2 KRW입니다. 이제 Reimbursement Agent를 사용하여 27665.2 KRW 환불을 진행하시겠습니까?

네.

Reimbursement Agent가 환불을 진행하기 전에 거래 날짜와 사업상 정당성/목적이 필요하다고 합니다. 해당 정보를 제공해 주시겠습니까?

2024-05-01, 중고노트북을 판 정당한 거래.

Date
2024-05-01

Amount
27665.2

Purpose
중고노트북을 판 정당한 거래

Request ID
request_id_1495697

Cancel Submit

How can I help you?

<https://github.com/a2aproject/a2a-samples/tree/main/samples/python/agents/langgraph>

Conversation

\$20로 결제한 1개월 구독권을 한국 돈으로 환전해서 환불받고 싶어

환불 처리를 위해 먼저 20달러를 한국 돈으로 환전해야 합니다. "Currency Agent"를 사용하여 환전을 진행한 다음, "Reimbursement Agent"를 사용하여 환불을 요청할 수 있습니다. 먼저 환전을 진행하시겠습니까?

우선 환전을 해줘.

Could you please specify the currency you want to convert from? I need the three-letter currency code (e.g., USD, EUR, GBP).

USD

이제 27216.4 KRW로 환전되었습니다. 환불 처리를 위해 "Reimbursement Agent"에게 환불 요청을 보내시겠습니까?

네.

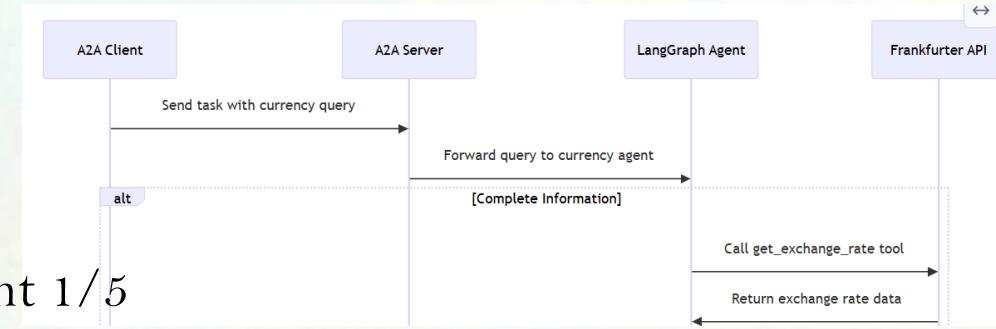
Working...

How can I help you?

date: 2024-05-01
amount: 27665.2
purpose: 중고노트북을 판 정당한 거래
request_id: request_id_1495697

Form submitted

환불 요청이 승인되었습니다. 요청 ID는 request_id_1495697입니다.



2) A2A

A2A(Agent-to-Agent) hands on 2 – 환전, 비용처리 Agent 2/5

```

import logging
import os
import sys

import click
import httpx
import uvicorn

# A2A 프레임워크 관련 핵심 컴포넌트 임포트
from a2a.server.apps import A2AStarletteApplication
from a2a.server.request_handlers import DefaultRequestHandler
from a2a.server.tasks import InMemoryPushNotifier, InMemoryTaskStore
from a2a.types import (
    AgentCapabilities,
    AgentCard,
    AgentSkill,
)
from dotenv import load_dotenv

# 사용자 정의 애이전트 로직 및 실행기 임포트
from app.agent import CurrencyAgent
from app.agent_executor import CurrencyAgentExecutor

load_dotenv()

logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

```

```

class MissingAPICallError(Exception):
    """Exception for missing API key."""

@click.command()
@click.option('--host', 'host', default='localhost')
@click.option('--port', 'port', default=10000)
def main(host, port):
    """환율 정보 애이전트(Currency Agent) 서버를 시작하는 메인 함수"""
    try:
        # [1]. LLM 소스 및 API 키 유효성 검사
        # 모델 소스가 google인 경우와 커스텀(TOOL_LLM)인 경우를 구분하여 체크
        if os.getenv('model_source', 'google') == 'google':
            if not os.getenv('GOOGLE_API_KEY'):
                raise MissingAPICallError(
                    'GOOGLE_API_KEY environment variable not set.'
                )
        else:
            if not os.getenv('TOOL_LLM_URL'):
                raise MissingAPICallError(
                    'TOOL_LLM_URL environment variable not set.'
                )
            if not os.getenv('TOOL_LLM_NAME'):
                raise MissingAPICallError(
                    'TOOL_LLM_NAME environment variable not set.'
                )

        # [2]. 애이전트 역량(Capabilities) 설정
        # 스트리밍 응답 지원 여부와 푸시 알림 지원 여부를 정의
        capabilities = AgentCapabilities(streaming=True, pushNotifications=True)
    
```

2) A2A

 A2A(Agent-to-Agent) hands on 2 – 환전, 비용처리 Agent 2/5

```

# [3]. 에이전트 스킬(skill) 정의
# 이 에이전트가 어떤 구체적인 작업을 수행할 수 있는지 다른 에이전트나 시스템에 알림
skill = AgentSkill(
    id='convert_currency',
    name='Currency Exchange Rates Tool',
    description='Helps with exchange values between various currencies',
    tags=['currency conversion', 'currency exchange'],
    examples=['What is exchange rate between USD and GBP?'],
)

# [4]. 에이전트 카드(AgentCard) 작성
# 에이전트의 메타데이터(이름, 버전, URL, 입출력 모드, 스킬 등)를 포함하는 명세서
agent_card = AgentCard(
    name='Currency Agent',
    description='Helps with exchange rates for currencies',
    url=f'http://{host}:{port}/',
    version='1.0.0',
    # 에이전트가 지원하는 데이터 타입 (예: text, json 등)을 설정
    defaultInputModes=CurrencyAgent.SUPPORTED_CONTENT_TYPES,
    defaultOutputModes=CurrencyAgent.SUPPORTED_CONTENT_TYPES,
    capabilities=capabilities,
    skills=[skill],
)

# [5]. 실행 환경 및 핸들러 설정
# 비동기 통신을 위한 HTTPX 클라이언트 초기화
httpx_client = httpx.AsyncClient()

# DefaultRequestHandler: 요청을 받아 에이전트 실행기로 전달하고 상태를 관리함
request_handler = DefaultRequestHandler(
    agent_executor=CurrencyAgentExecutor(),      # 실제 비즈니스 로직(환율 계산) 수행
    task_store=InMemoryTaskStore(),                # 작업 상태를 메모리에 임시 저장
    push_notifier=InMemoryPushNotifier(httpx_client), # 비동기 알림 처리
)

```

```

# [6]. A2A 서버 애플리케이션 구축
# Starlette 기반의 웹 애플리케이션으로 에이전트 명세와 핸들러를 결합
server = A2AStarletteApplication(
    agent_card=agent_card, http_handler=request_handler
)

# [7]. Uvicorn 서버 실행
uvicorn.run(server.build(), host=host, port=port)

except MissingAPIKeyError as e:
    logger.error(f'Error: {e}')
    sys.exit(1)
except Exception as e:
    logger.error(f'An error occurred during server startup: {e}')
    sys.exit(1)

if __name__ == '__main__':
    main()

```



2) A2A

 A2A(Agent-to-Agent) hands on 2 – 환전, 비용처리 Agent 4/5

1. 환전 Agent

```
(a2a-sample-agent-langgraph) PS D:\GITLAB\text2SQL\a2a-samples\samples\python\agents\langgraph> code .env
(a2a-sample-agent-langgraph) PS D:\GITLAB\text2SQL\a2a-samples\samples\python\agents\langgraph> uv run app
```

<https://console.cloud.google.com/apis>

```
GOOGLE_API_KEY=A1zaSyBnX5GmuWo2u2hXA8c_GGGGGGGGGG
TOOL_LLM_URL="http://192.168.1.203:11434"
TOOL_LLM_NAME="mistral:latest"
```

2. 비용처리 Agent

2.1 소스수정필요

```
(adk_expense_reimbursement) PS D:\GITLAB\text2SQL\a2a-samples\samples\python\agents\adk_expense_reimbursement> code agents.py
```

str | None으로 되어 있는 모든 매개변수를 Optional[str]로 변경

```
def create_request_form(
    date: Optional[str] = None,
    amount: Optional[str] = None,
    purpose: Optional[str] = None,
) -> dict[str, Any]:
    ...
def return_form(
    form_request: dict[str, Any],
    tool_context: ToolContext,
    instructions: Optional[str] = None,
) -> dict[str, Any]:
```

```
(adk_expense_reimbursement) PS D:\GITLAB\text2SQL\a2a-samples\samples\python\agents\adk_expense_reimbursement> uv run .
```

3. agent 관리 ui

```
(a2a-python-example-ui) PS D:\GITLAB\text2SQL\a2a-samples\demo\ui> uv run main.py
```

2) A2A

 A2A(Agent-to-Agent) hands on 2 – 환전, 비용처리 Agent 5/5

localhost:12000/agents

Remote Agents

| Address | Name | Description | Organization | Input Modes | Output Modes | Streaming |
|-------------------------|---------------------|---|--------------|------------------|------------------|-----------|
| http://localhost:10000/ | Currency Agent | Helps with exchange rates for currencies | | text, text/plain | text, text/plain | True |
| http://localhost:10002/ | Reimbursement Agent | This agent handles the reimbursement process for the employees given the amount and purpose of the reimbursement. | | text, text/plain | text, text/plain | True |

localhost:10000/well-known/agent.json

```
{
  "capabilities": {
    "pushNotifications": true,
    "streaming": true
  },
  "defaultInputModes": [
    "text",
    "text/plain"
  ],
  "defaultOutputModes": [
    "text",
    "text/plain"
  ],
  "description": "Helps with exchange rates for currencies",
  "name": "Currency Agent",
  "skills": [
    {
      "description": "Helps with exchange values between various currencies",
      "examples": [
        "What is exchange rate between USD and GBP?"
      ],
      "id": "convert_currency",
      "name": "Currency Exchange Rates Tool",
      "tags": [
        "currency conversion",
        "currency exchange"
      ]
    }
  ],
  "url": "http://localhost:10000/",
  "version": "1.0.0"
}
```

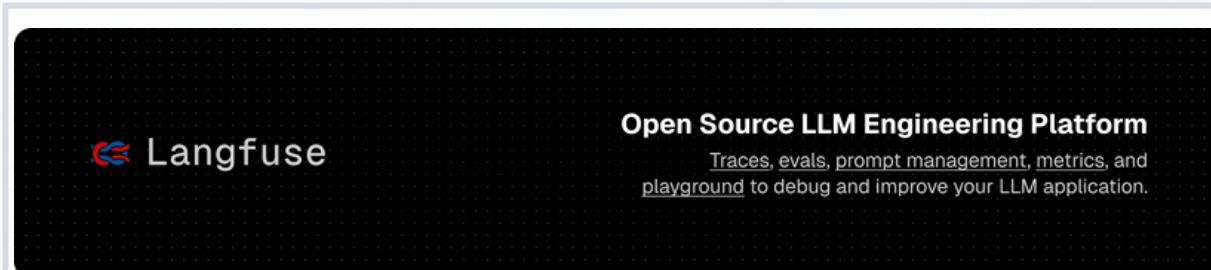
localhost:10000/agents

Remote Agents

| Address | Name | Description | Organization | Input Modes | Output Modes | Streaming |
|-------------------------|---------------------|---|--------------|------------------|------------------|-----------|
| http://localhost:10000/ | Currency Agent | Helps with exchange rates for currencies | | text, text/plain | text, text/plain | True |
| http://localhost:10002/ | Reimbursement Agent | This agent handles the reimbursement process for the employees given the amount and purpose of the reimbursement. | | text, text/plain | text, text/plain | True |

3) Langfuse

Langfuse란



traces, evals, metrics,
prompt management,
and playground...

...to debug and
improve your LLM
application together.

Langfuse

Langfuse는 대규모 언어 모델(LLM)로 구동되는 애플리케이션을 위해 특별히 설계된 **오픈 소스** 통합 가시성 및 분석 플랫폼입니다.

Langfuse는 시장에서 [가장 인기 있는 오픈소스 LLMOps 도구로](#)

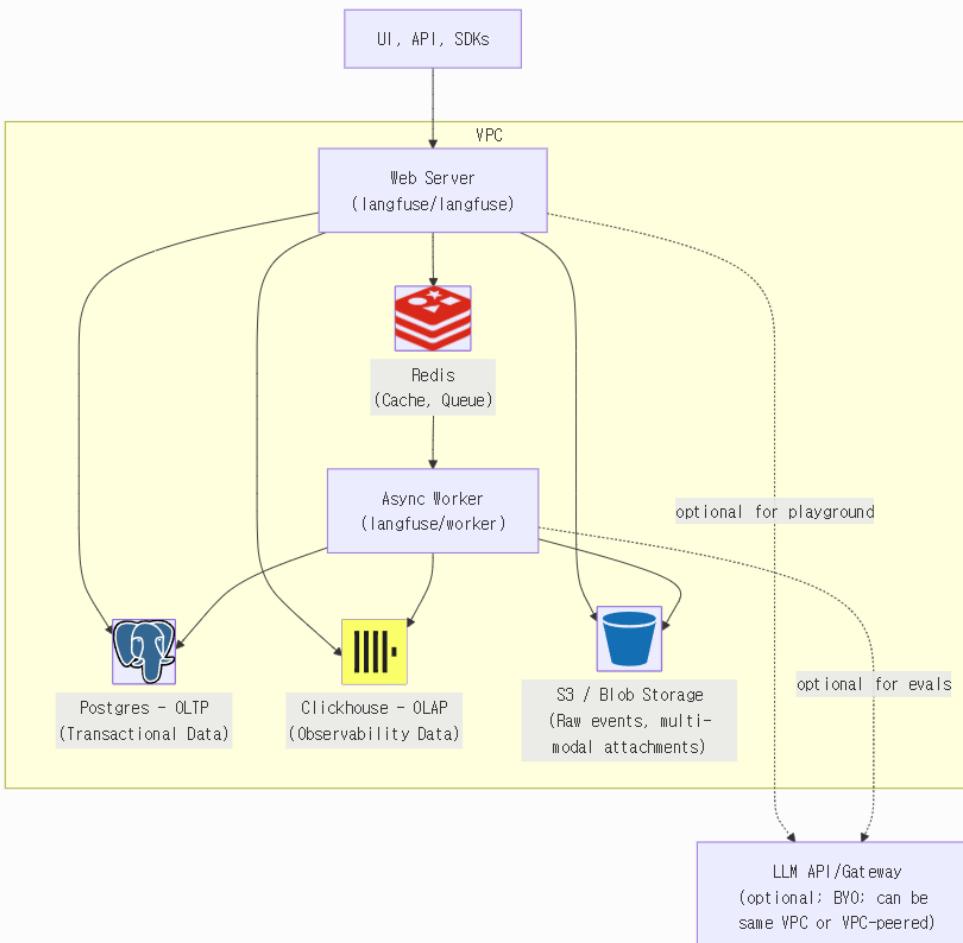
Langfuse는 [셀프 호스팅이 쉬우며](#)

Langfuse는 제품을 개발하고 개선하는 데 도움이 되는 **동급 최고의 추적 기능**을 제공

3) Langfuse

Langfuse Architecture

Langfuse only depends on open source components and can be deployed locally, on cloud infrastructure, or on-premises.



•LLM 애플리케이션 관측

앱에 계측(instrumentation)을 추가하여 Langfuse로 trace 데이터를 수집함으로써, 검색, 임베딩, 또는 에이전트 동작과 같은 LLM 호출 및 기타 관련 로직을 추적할 수 있습니다. 복잡한 로그와 사용자 세션을 확인 및 디버깅 해보세요. 인터랙티브 데모를 통해 실제 작동 예를 확인할 수 있습니다.

•프롬프트 관리

프롬프트를 중앙에서 관리하고 버전 관리하며 협업으로 수정할 수 있도록 도와줍니다. 서버와 클라이언트 측의 강력한 캐싱 덕분에 애플리케이션에 지연(latency)을 추가하지 않고도 프롬프트를 반복 개선할 수 있습니다.

•평가

LLM 애플리케이션 개발 워크플로우에서 핵심적인 역할을 하며, Langfuse는 여러분의 필요에 맞게 유연하게 대응합니다. LLM을 심사자로 활용하는 기능, 사용자 피드백 수집, 수동 라벨링 및 API/SDK를 통한 맞춤 평가 파이프라인을 지원합니다.

•데이터셋

LLM 애플리케이션 평가를 위한 테스트 세트와 벤치마크를 제공하여, 지속적인 개선, 배포 전 테스트, 구조화된 실험, 유연한 평가 및 LangChain과 LlamaIndex와 같은 프레임워크와의 원활한 통합을 지원합니다.

•LLM 플레이그라운드

프롬프트와 모델 구성에 대해 테스트 및 반복 개선할 수 있는 도구로, 피드백 루프를 단축하여 개발 속도를 높여줍니다. trace에서 이상한 결과가 발생하면 플레이그라운드로 바로 이동해 개선할 수 있습니다.

•종합 API

Langfuse는 API를 통해 제공되는 구성 요소들을 활용하여 맞춤형 LLMOps 워크플로우를 강화하는데 자주 사용됩니다. OpenAPI 명세, Postman 컬렉션, Python 및 JS/TS용 타입드 SDK가 제공됩니다.

3) Langfuse

Langfuse quickstart 1/2

<https://us.cloud.langfuse.com/>

Setup

1. Create Organization > 2. Invite Members > 3. Create Project > **4. Setup Tracing** >

API Keys

These keys are used to authenticate your API requests. You can create more keys later in the project settings.

You need to create an API key to start tracing your application.

[Create API Key](#)

Setup Tracing

pending

Tracing is used to track and analyze your LLM calls. You can always skip this step and setup tracing later.

Python JS/TS OpenAI Langchain Langchain JS Other

```
pip install langfuse
```

```
from langfuse import Langfuse

langfuse = Langfuse(
    secret_key="secret key",
    public_key="public key",
    host="https://us.cloud.langfuse.com"
)
```

See [Quickstart](#) and [Python docs](#) for more details and an end-to-end example.

Do you have questions or issues? Check out this [FAQ post](#) for common resolutions, [Ask AI](#) or [get support](#).

[Skip for now](#)

```
from langfuse import Langfuse

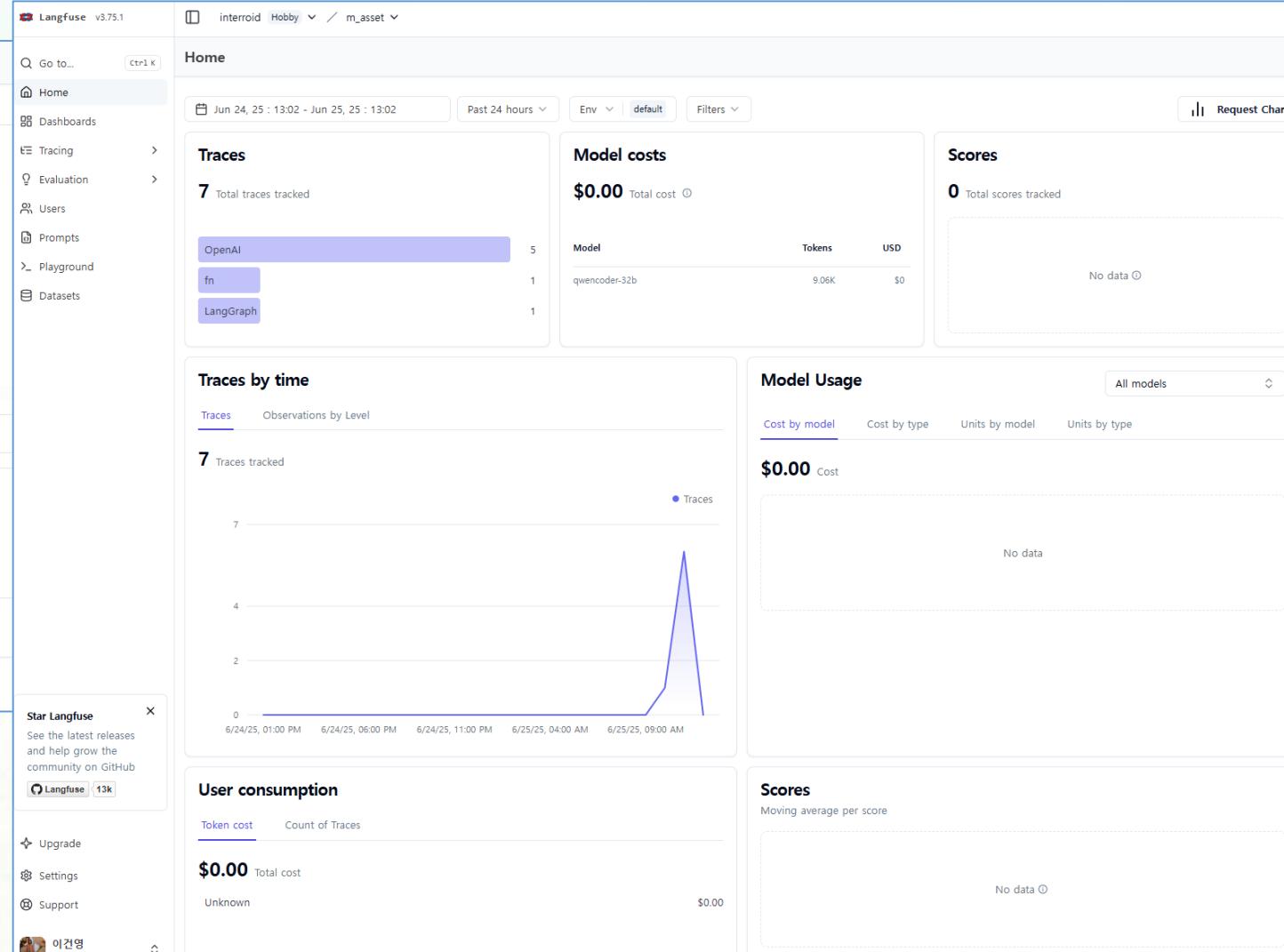
langfuse = Langfuse(
    secret_key="sk-1f-14dc9542-fc0c-48c9-a742-ggggggggggggg",
    public_key="pk-1f-f8bcdf8f-21e4-4cb6-87bf-ggggggggggggg",
    host="https://us.cloud.langfuse.com"
)
```

```
from langfuse.langchain import CallbackHandler

# Initialize Langfuse CallbackHandler for Langchain (tracing)
langfuse_handler = CallbackHandler()

...
graph.invoke(inputs, config={"callbacks": [langfuse_handler]})

response = llm_sql_gen.invoke(prompt, config={"callbacks": [langfuse_handler]})
```



2

MCP, A2A, Langfuse



3) Langfu

The screenshot displays the Langfuse interface version 3.75.1. The top navigation bar shows the project name "interroid Hobby" and the dataset "m_asset". The left sidebar includes links for Home, Dashboards, Tracing, Evaluation, Users, Prompts, Playground, and Datasets. A "Star Langfuse" button is present, along with Upgrade, Settings, and Support options.

The main area is divided into two sections: "Traces" and "LangGraph".

Traces: This section lists traces from June 25, 2025, showing interactions with OpenAI and various internal components like "get_schema_from_vector_db", "generate_sql_query_with_schema", "check_sql_query", "sql_db_query_checker", "LLMChain", and "execute_sql_query". Each trace entry includes a timestamp, name, input, and output.

LangGraph: This section provides a visual representation of the workflow. It starts with a blue box labeled "start_". An arrow points from "start_" to a blue box labeled "get_schema_from_vector_db". Another arrow points from "get_schema_from_vector_db" to a blue box labeled "generate_sql_query_with_schema". From "generate_sql_query_with_schema", an arrow points to a blue box labeled "check_sql_query". Finally, an arrow points from "check_sql_query" to a blue box labeled "execute_sql_query", which ends with a red box labeled "end_".

Trace Detail: A detailed view of a specific trace is shown on the right. The trace ID is 79f3e46803f06f8a4933c9daa0761521, with a timestamp of 2025-06-25 11:25:20.545. The latency is 0.11s and the environment is default. The preview shows the schema for "m_asset.securities_stock_info_kospi" and "m_asset.exchange_kosdaq_stock_master_01".



- ❶ 출처 1 : https://modulabs.co.kr/blog/langgraph_multiagent
- ❷ 출처 2 : <https://www.blog.langchain.com/langgraph-multi-agent-workflows/>
- ❸ 출처 3 : <https://github.com/langfuse/langfuse>
- ❹ 출처 4 : <https://langfuse.com/kr>
- ❺ 출처 5 : <https://langfuse.com/blog/2024-12-langfuse-v3-infrastructure-evolution>
- ❻ 출처 6 : <https://github.com/langfuse/langfuse/blob/main/README.kr.md>

- ✿ LLM Agent는 LLM을 활용하므로 응답에 대한 신뢰성을 평가하고, 필요시 외부 도구를 호출하여 정보를 보완하는 기능이 중요합니다. LangGraph는 이러한 기능을 체계적으로 구현할 수 있는 프레임워크.
- ✿ LangGraph는 역할 별 Agent에 한 Plan, 단계별 역할 수행을 설계, 수행할 수 있는 Supervisor, Planner, Evaluator를 사용하고 각 역할 별로 세부 Task를 정의하여 체계적으로 관리.
- ✿ MCP(Model Context Protocol)는 AI 모델이 외부 도구, 데이터, 애플리케이션과 표준화된 방식으로 소통하고 연결할 수 있도록 돋는 개방형 통신 규약.
- ✿ A2A(Agent-to-Agent)는 AI 에이전트들이 서로 소통하고 협력하기 위한 구조 및 플랫폼
- ✿ Langfuse는 LLM 애플리케이션을 위해 특별히 설계된 오픈 소스 통합 분석 플랫폼