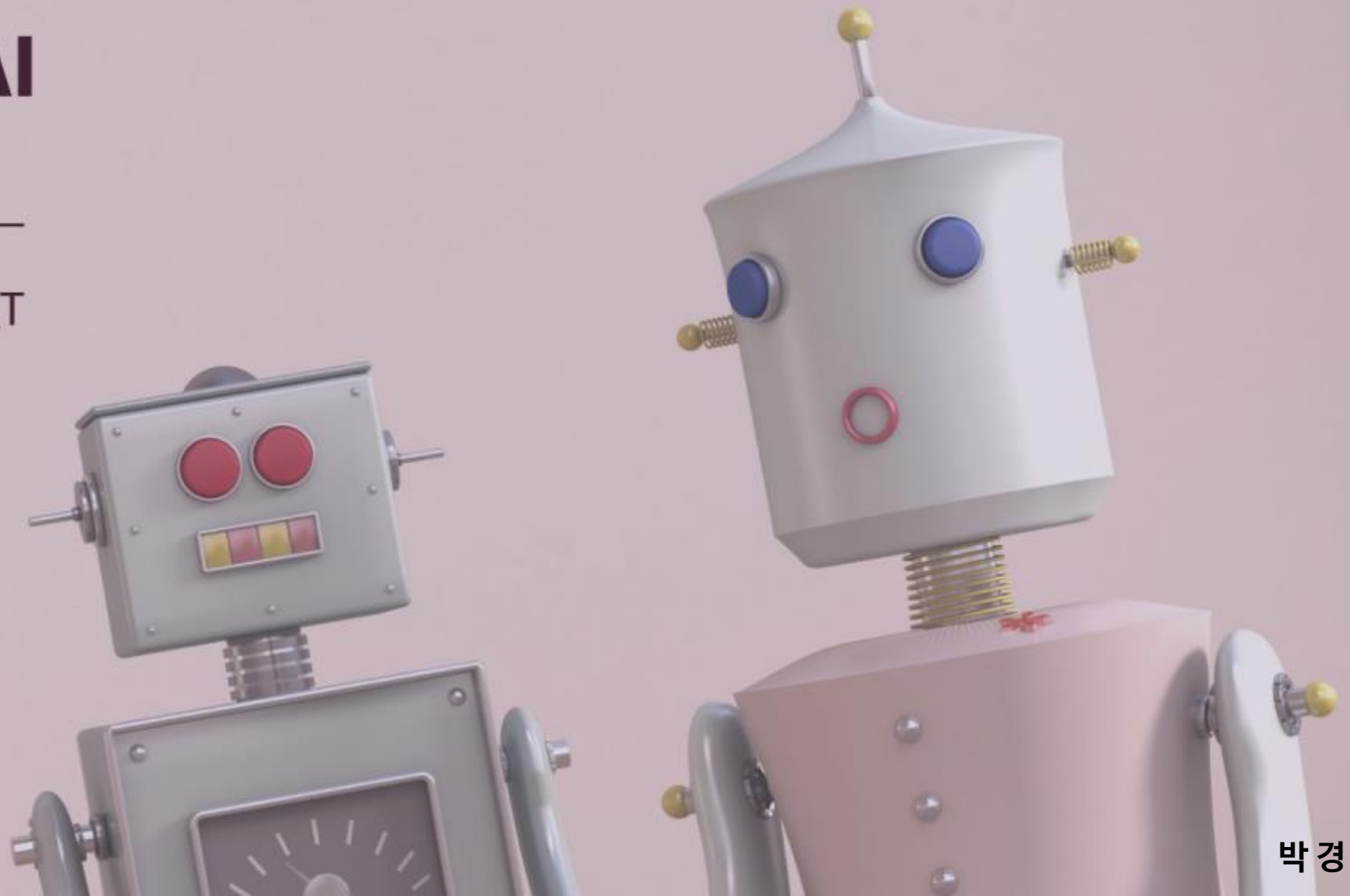
MULTIPLE AI AGENTS

CREATE MULTI-AGENT WORKFLOWS USING LANGGRAPH AND LANGCHAIN



박경규

LangGraph

https://github.com/langchain-ai/langgraph

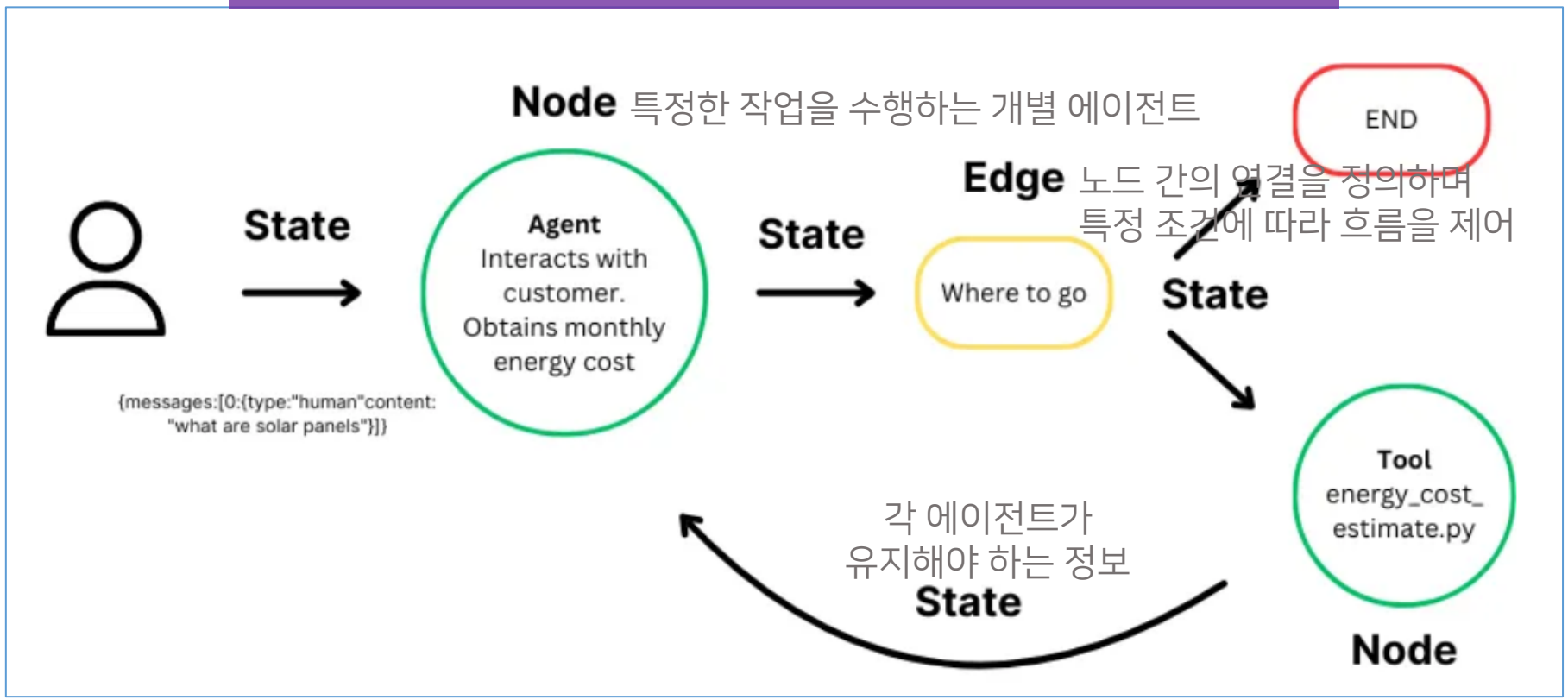
LangGraph는 제어 가능한 에이전트를 구축하기 위한 저수준 오케스트레이션 프레임워크입니다.

Langchain은 LLM 애플리케이션 개발을 간소화하기 위한 통합 및 구성 가능한 구성 요소를 제공하는 반면,

LangGraph 라이브러리는 에이전트 오케스트레이션을 가능하게 합니다.

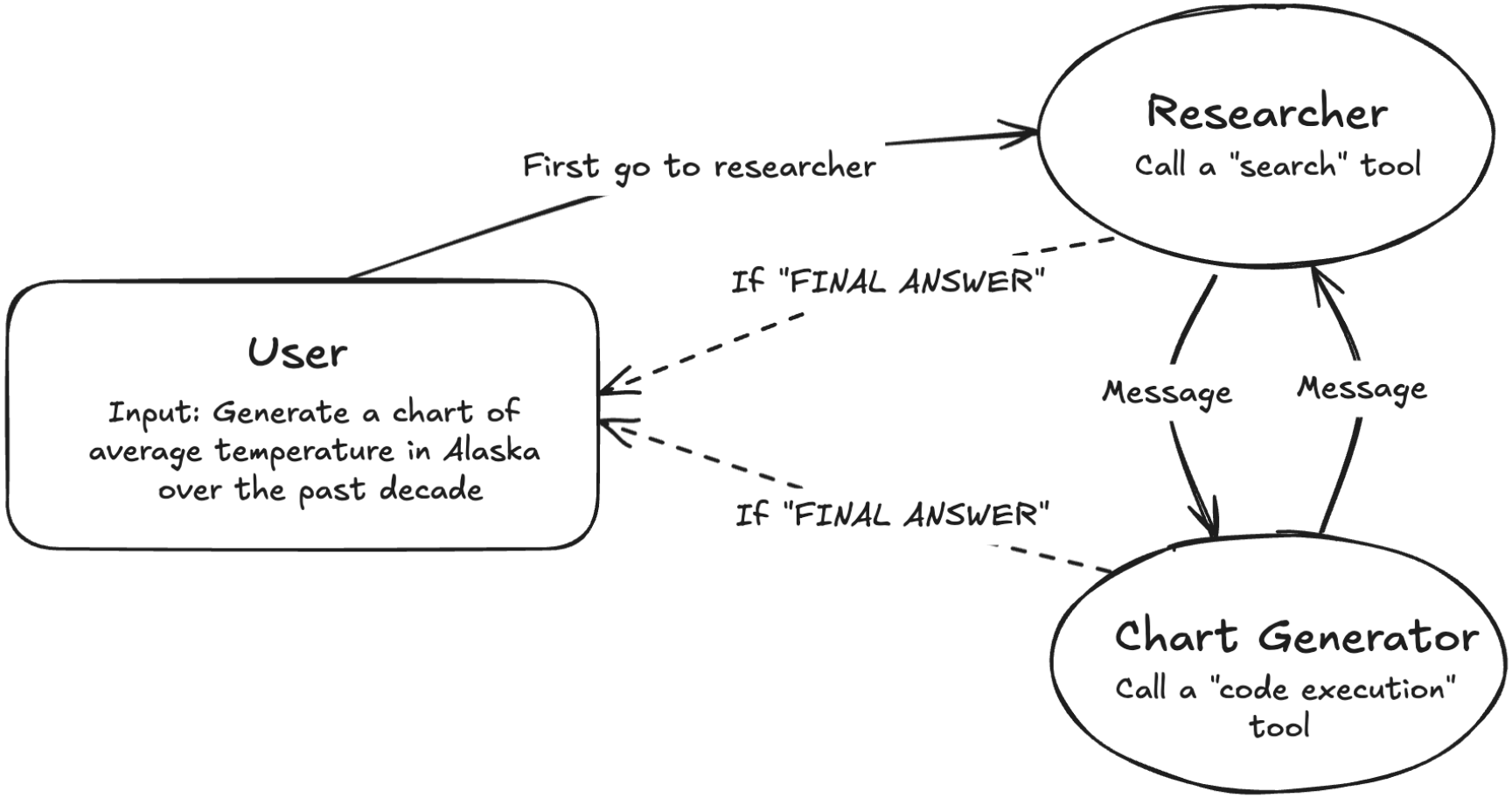
사용자 정의 가능한 아키텍처, 장기 메모리, human-in-the-loop 를 제공하여 복잡한 작업을 안정적으로 처리합니다.

그래프(Graph) : 노드와 엣지가 조합된 전체 워크플로우의 구조



멀티 에이전트 네트워크

멀티 AI 에이전트 시스템은 Divide And Conquer 방식처럼, 문제를 여러 서브 도메인에 특화된 AI 에이전트로 나누어 해결합니다.



멀티 AI 에이전트 협업 기능 구현 multi-agent-collaboration.ipynb

필수 패키지 설치

```
[1]  %%capture --no-stderr
     %pip install -U langchain_community langchain_openai langchain_anthropic
     %pip install -U langchain_experimental matplotlib langgraph
```

API 키 설정

o OPENAI_API_KEY : https://platform.openai.com/api-keys

o ANTHROPIC_API_KEY : https://console.anthropic.com/settings/keys

o TAVILY_API_KEY : https://app.tavily.com/home -> API Keys

멀티 AI 에이전트 협업 기능 구현

```
[2]  import getpass
     import os


     def _set_if_undefined(var: str):
         if not os.environ.get(var):
             os.environ[var] = getpass.getpass(f"Please provide your {var}")


     _set_if_undefined("OPENAI_API_KEY")
     _set_if_undefined("ANTHROPIC_API_KEY")
     _set_if_undefined("TAVILY_API_KEY")
```

```
Please provide your OPENAI_API_KEY · · · · · · · · · ·
Please provide your ANTHROPIC_API_KEY · · · · · · · · · ·
Please provide your TAVILY_API_KEY · · · · · · · · · ·
```

멀티 AI 에이전트 협업 기능 구현

도구(Tool) 정의

에이전트가 사용할 도구를 정의(Define)

```
[3]  from typing import Annotated

     from langchain_community.tools.tavily_search import TavilySearchResults
     from langchain_core.tools import tool
     from langchain_experimental.utilities import PythonREPL

     tavily_tool = TavilySearchResults(max_results=5)

     # Warning: This executes code locally, which can be unsafe when not sandboxed

     repl = PythonREPL()
```

멀티 AI 에이전트 협업 기능 구현

```python
@tool
def python_repl_tool(
    code: Annotated[str, "The python code to execute to generate your chart."],
):
    """Use this to execute python code. 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```python₩n{code}₩n```₩nStdout: {result}"
    return (
        result_str + "₩n₩nIf you have completed all tasks, respond with FINAL ANSWER."
    )
```

멀티 AI 에이전트 협업 기능 구현

그래프 만들기

이제 도구를 정의하고 일부 helper function을 만들었으므로 아래의
개별 에이전트를 만들고 LangGraph를 사용하여 서로 대화하는 방법을 알려줍니다.

에이전트 노드 정의

먼저 각 에이전트에 대한 시스템 프롬프트를 만드는 유틸리티를 만듭니다.

[4]
```python
def make_system_prompt(suffix: str) -> str:
    return (
        "You are a helpful AI assistant, collaborating with other assistants."
        " Use the provided tools to progress towards answering the question."
        " If you are unable to fully answer, that's OK, another assistant with different tools "
        " will help where you left off. Execute what you can to make progress."
        " If you or any of the other assistants have the final answer or deliverable,"
        " prefix your response with FINAL ANSWER so the team knows to stop."
        f"₩n{suffix}"
    )
```

멀티 AI 에이전트 협업 기능 구현

[5]
```python
from typing import Literal

from langchain_core.messages import BaseMessage, HumanMessage
from langchain_openai import ChatOpenAI
from langchain_anthropic import ChatAnthropic
from langgraph.prebuilt import create_react_agent
from langgraph.graph import MessagesState, END
from langgraph.types import Command

llm = ChatOpenAI(model="gpt-4-turbo")
# llm = ChatAnthropic(model="claude-3-5-sonnet-latest")

def get_next_node(last_message: BaseMessage, goto: str):
    if "FINAL ANSWER" in last_message.content:
        # Any agent decided the work is done
        return END
    return goto
```

멀티 AI 에이전트 협업 기능 구현

```
# Research agent and node
research_agent = create_react_agent(
    llm,
    tools=[tavily_tool],
    prompt=make_system_prompt(
        "You can only do research. You are working with a chart generator colleague."
    ),
)
```

멀티 AI 에이전트 협업 기능 구현

```python
def research_node(
    state: MessagesState,
) -> Command[Literal["chart_generator", END]]:
    result = research_agent.invoke(state)
    goto = get_next_node(result["messages"][-1], "chart_generator")
    # wrap in a human message, as not all providers allow
    # AI message at the last position of the input messages list
    result["messages"][-1] = HumanMessage(
        content=result["messages"][-1].content, name="researcher"
    )
    return Command(
        update={
            # share internal message history of research agent with other agents
            "messages": result["messages"],
        },
        goto=goto,
    )
```

멀티 AI 에이전트 협업 기능 구현

```
# Chart generator agent and node
# NOTE: THIS PERFORMS ARBITRARY CODE EXECUTION, WHICH CAN BE UNSAFE WHEN NOT SANDBOXED
chart_agent = create_react_agent(
    llm,
    [python_repl_tool],
    prompt=make_system_prompt(
        "You can only generate charts. You are working with a researcher colleague."
    ),
)
```

멀티 AI 에이전트 협업 기능 구현

```python
def chart_node(state: MessagesState) -> Command[Literal["researcher", END]]:
    result = chart_agent.invoke(state)
    goto = get_next_node(result["messages"][-1], "researcher")
    # wrap in a human message, as not all providers allow
    # AI message at the last position of the input messages list
    result["messages"][-1] = HumanMessage(
        content=result["messages"][-1].content, name="chart_generator"
    )
    return Command(
        update={
            # share internal message history of chart agent with other agents
            "messages": result["messages"],
        },
        goto=goto,
    )
```

멀티 AI 에이전트 협업 기능 구현

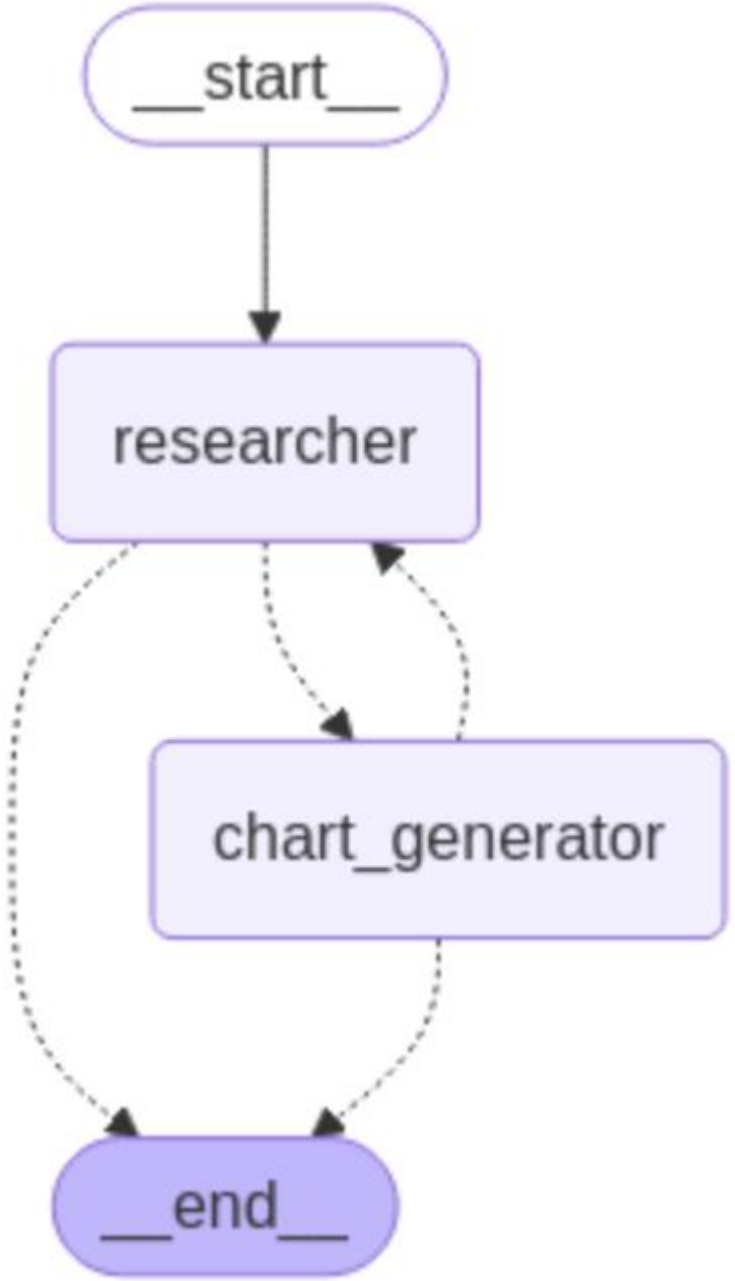
그래프 정의

이제 모든 것을 모아 그래프를 정의합니다.

```
[6]  from langgraph.graph import StateGraph, START

     workflow = StateGraph(MessagesState)
     workflow.add_node("researcher", research_node)
     workflow.add_node("chart_generator", chart_node)

     workflow.add_edge(START, "researcher")
     graph = workflow.compile()
```

멀티 AI 에이전트 협업 기능 구현

```
[7]  from IPython.display import Image, display

     try:
         display(Image(graph.get_graph().draw_mermaid_png()))
     except Exception:
         # This requires some extra dependencies and is optional
         pass
```



멀티 AI 에이전트 협업 기능 구현

호출(Invoke)

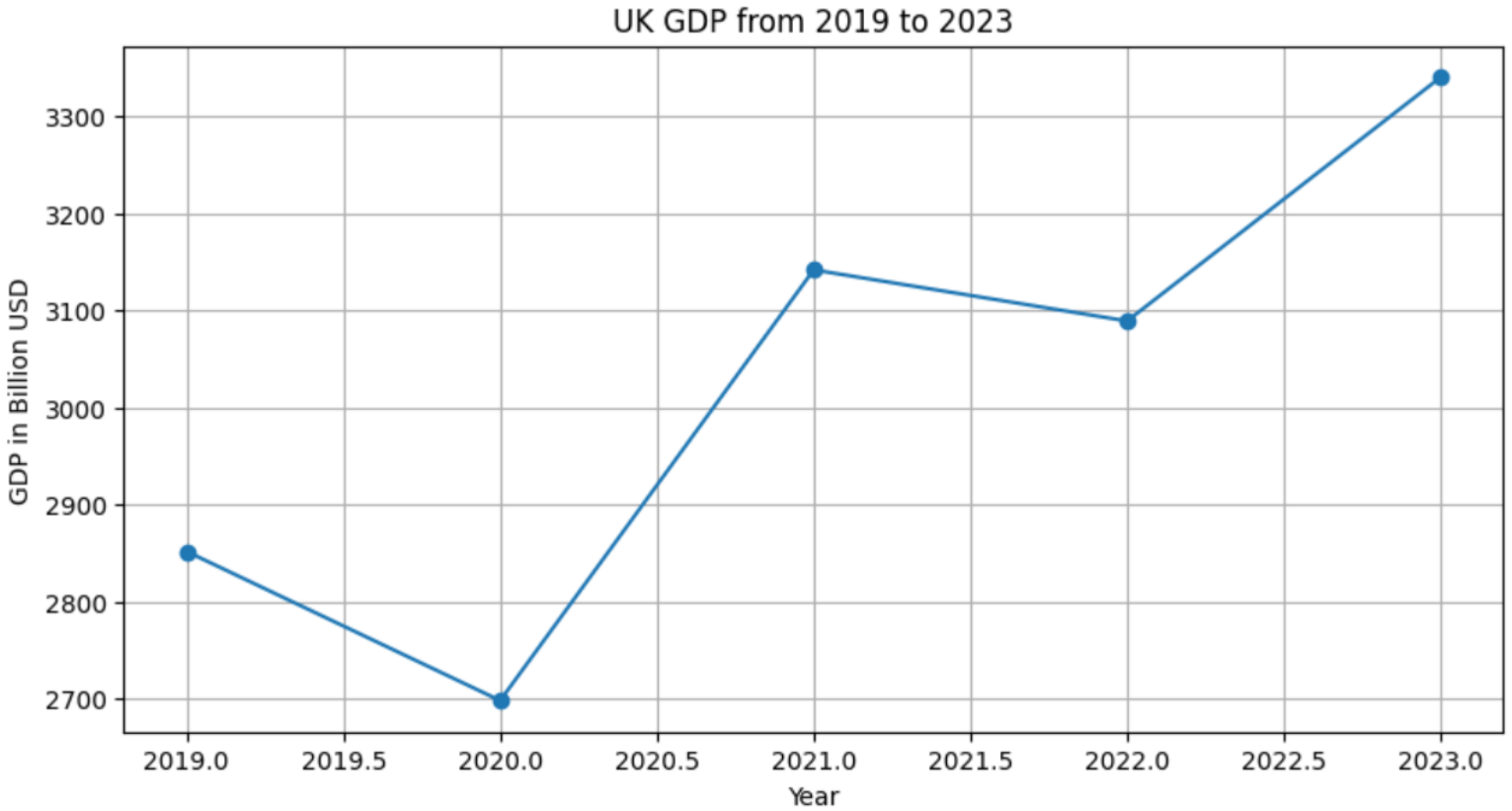
그래프가 생성되었으니 호출할 수 있습니다!

몇 가지 통계를 차트로 표시해 보겠습니다.

```
events = graph.stream(
    {
        "messages": [
            (
                "user",
                "First, get the UK's GDP over the past 5 years, then make a line chart of it. "
                "Once you make the chart, finish.",
            )
        ],
    },
```

멀티 AI 에이전트 협업 기능 구현

```
        # Maximum number of steps to take in the graph
        {"recursion_limit": 150},
    )
    for s in events:
        print(s)
        print("----")
```



Thank you