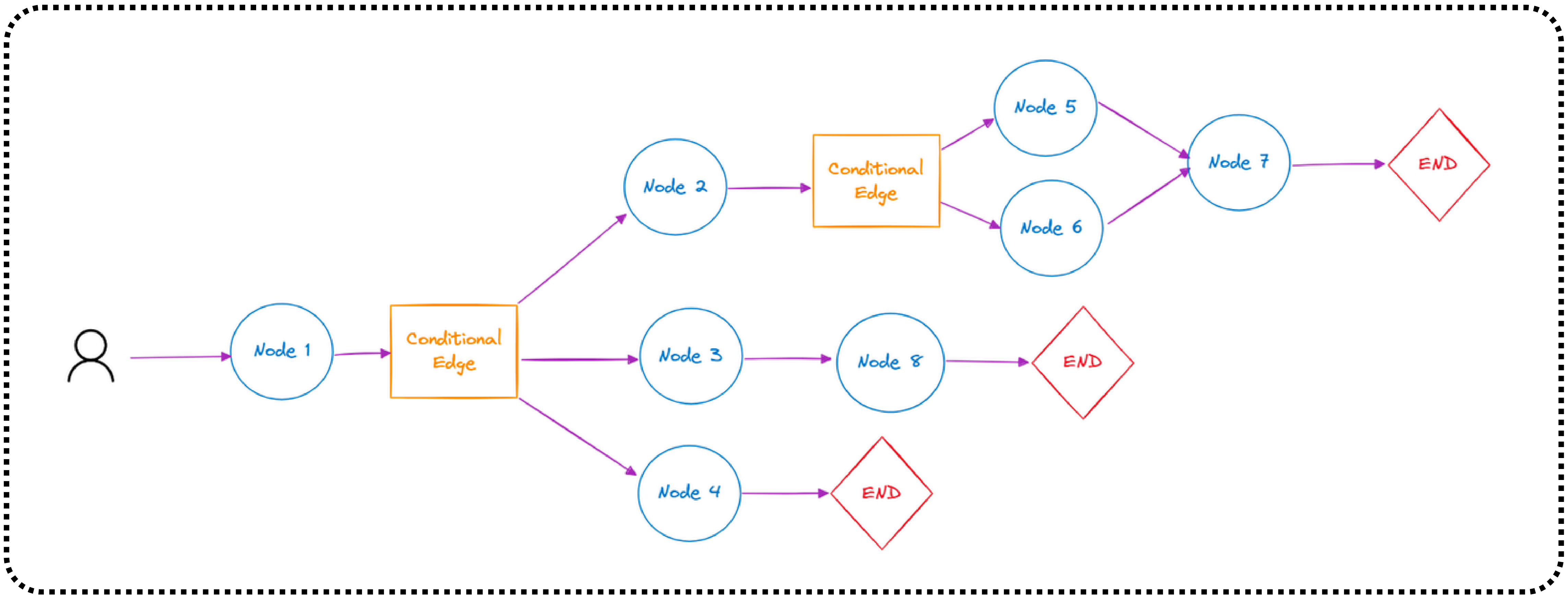
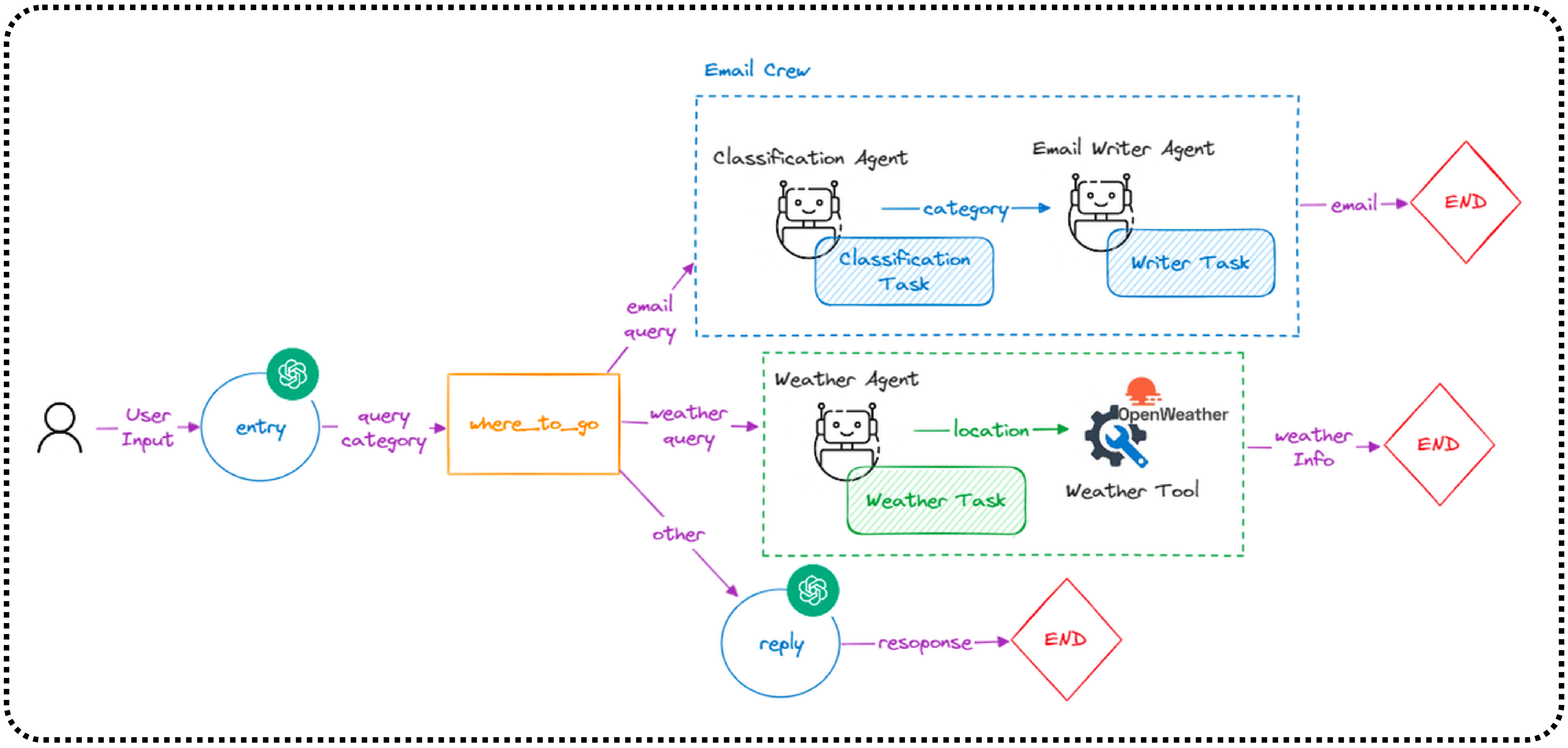
Day 16 of Mastering AI Agents

Deep Dive into LangGraph





What is LangGraph?

LangGraph presents a novel paradigm for agent building and runtime construction. Large Language Models (LLMs) are the foundation for designing sophisticated AI agents, and LangGraph, built on top of Langchain, is intended to make the process of creating cyclic graphs easier.

LangGraph views agent workflows as cyclic graph topologies at their foundation. This method enables more variable and nuanced behaviors from agents, surpassing its predecessors' linear execution model. Using graph theory, LangGraph provides new avenues for developing intricate, networked agent systems.

At its core, LangGraph is a stateful, multi-agent orchestration framework built on top of LangChain. It allows developers to:

- Design conversational agents as directed graphs (nodes and edges).
- Control flow between nodes dynamically, based on context or output.
- Support state persistence and asynchronous thinking.
- Easily integrate tools, APIs, memory, and other agents.

Instead of the classic chain or pipeline approach, LangGraph embraces graphs as the fundamental building block. This gives devs fine-grained control over how conversations, logic branches, and decisions unfold.

Why use LangGraph?

- **Flexibility**: As AI agents evolved, developers required more control over the agent runtime to enable personalized action plans and decision-making procedures.
- **The Cyclical Nature of AI Reasoning**: Many intricate LLM applications depend on cyclical execution when employing strategies like chain-of-thought reasoning. LangGraph offers a natural framework for modeling these cyclical processes.
- **Multi-Agent Systems**: As multi-agent workflows became more common, there was an increasing demand for a system that could efficiently manage and coordinate several autonomous agents.

State Management: As agents became more sophisticated, tracking and updating state data as the agent was being executed became necessary. LangGraph's stateful graph methodology satisfies this need.

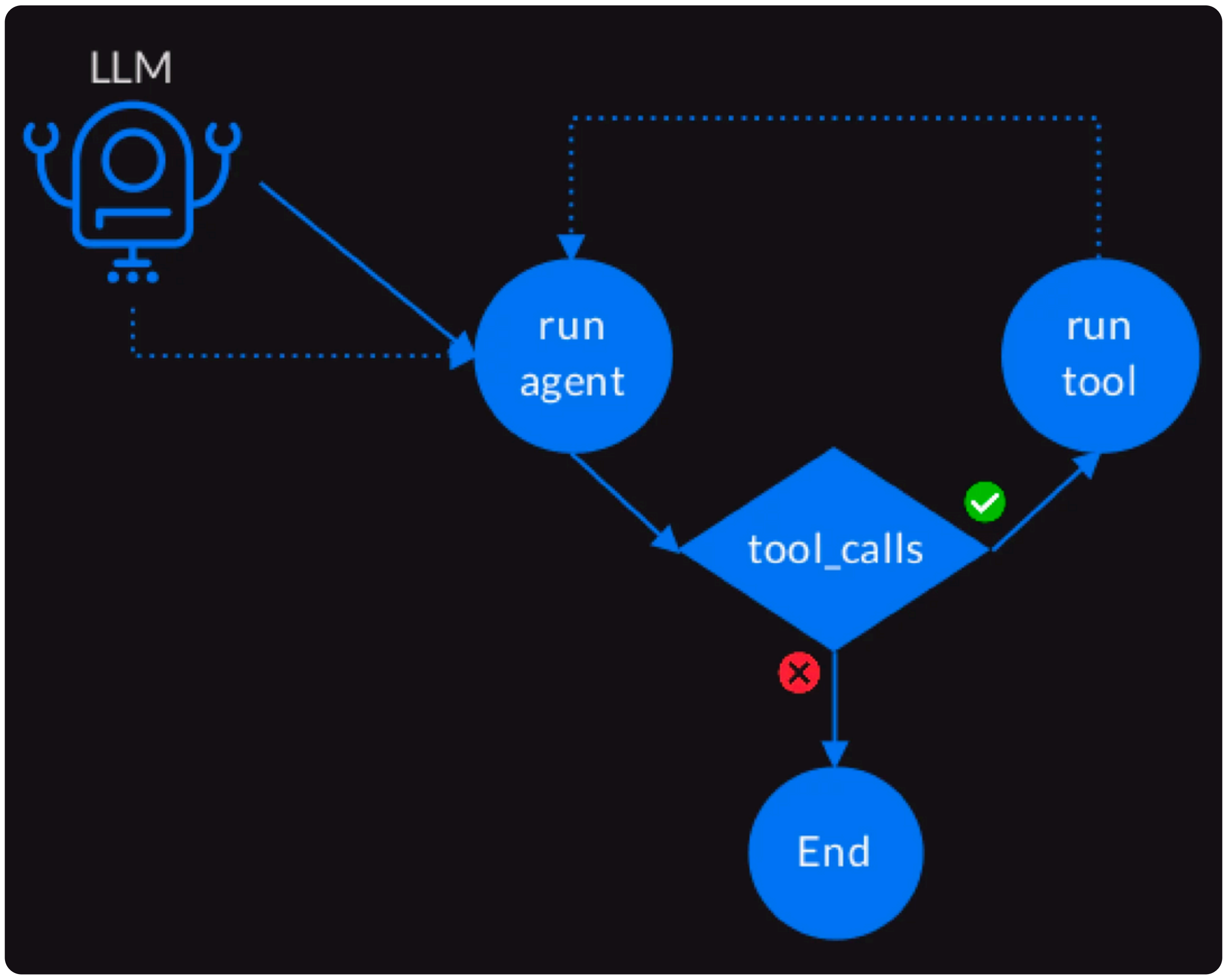
Analytics Vidhya

How LangGraph Works?

The functionality of LangGraph is based on several essential elements:

- **Nodes**: These are functions or Langchain runnable items, like the agent's tools.
- **Edges**: Paths that define the direction of execution and data flow within the agent system, connecting nodes.
- **Stateful Graphs**: LangGraph allows for persistent data across execution cycles by managing and updating state objects as data flows through the nodes.

The following diagram can be used to illustrate the working:



As shown in the image, the nodes include LLM, tools, etc. which are represented by circles or rhombus which. Flow of information between various nodes is represented by arrows.

The popular NetworkX library served as the model for the library's interface, which makes it user-friendly for developers with prior experience with graph-based programming.

LangGraph's approach to agent runtime differs significantly from that of its forerunners. Instead of a basic loop, it enables the construction of intricate, networked systems of nodes and edges. With this structure, developers can design more complex decision-making procedures and action sequences.

Now, let us build an agent using LangGraph to understand them better. First, we will implement tool calling, then using a pre-built agent, and then building an agent ourselves in LangGraph.

Tool Calling in LangGraph

Pre-requisites

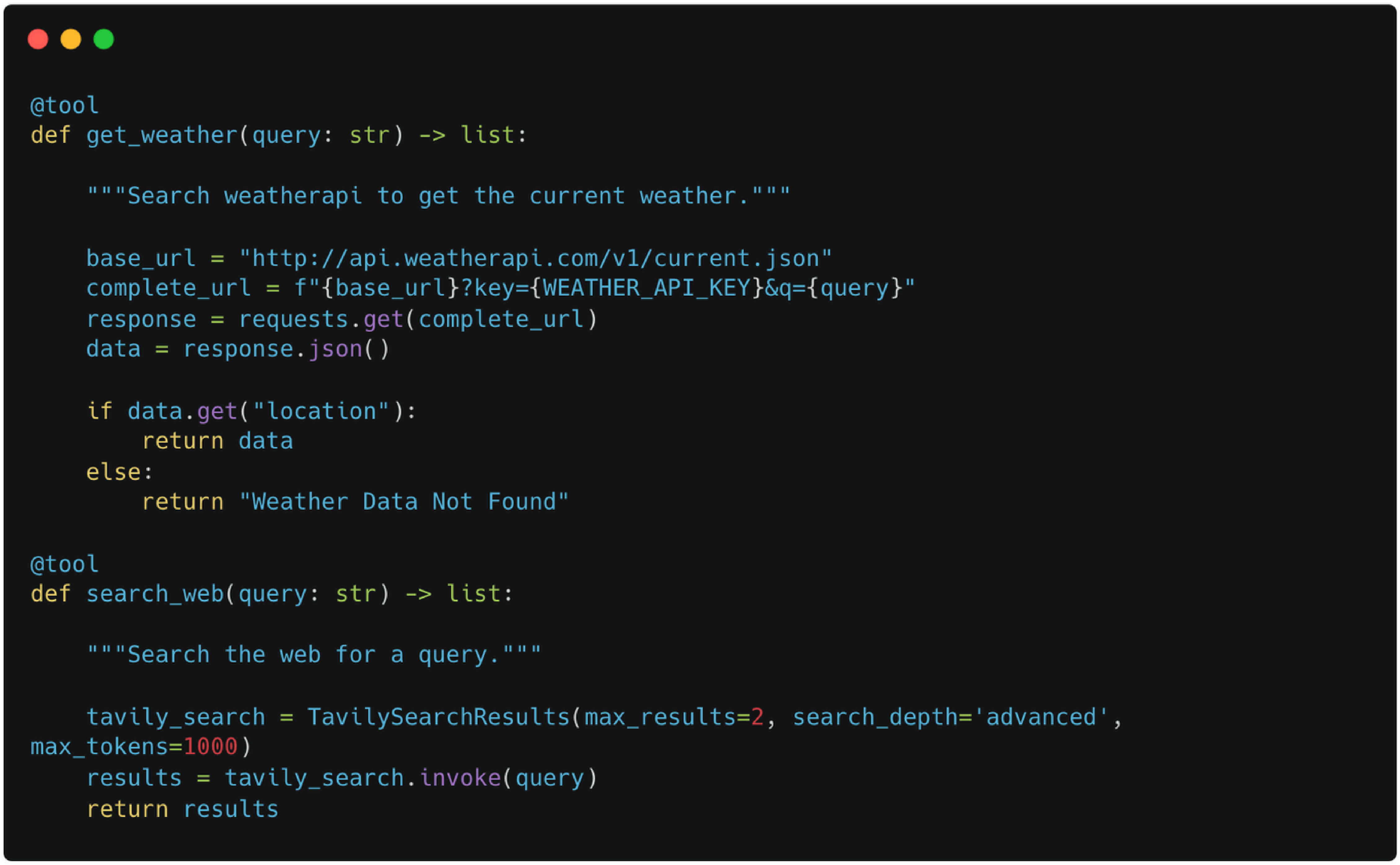
Create an OpenAI API key to access the LLMs and Weather API key (**get here**) to access the weather information. Store these keys in a '.env' file:

Load and Import the keys as follows:

```
import os

from dotenv import load_dotenv

load_dotenv('/.env')

WEATHER_API_KEY = os.environ['WEATHER_API_KEY']

# Import the required libraries and methods

import json

import requests

import rich

from typing import List, Literal

from IPython.display import Image, display

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

from langchain_openai import ChatOpenAI
```

Define Tools

We will define two tools. One is to to get weather information when the query is specific to weather and another is to search the internet when the LLM doesn't know the answer to the given query:



```python
@tool
def get_weather(query: str) -> list:

    """Search weatherapi to get the current weather."""

    base_url = "http://api.weatherapi.com/v1/current.json"
    complete_url = f"{base_url}?key={WEATHER_API_KEY}&q={query}"
    response = requests.get(complete_url)
    data = response.json()

    if data.get("location"):
        return data
    else:
        return "Weather Data Not Found"

@tool
def search_web(query: str) -> list:

    """Search the web for a query."""

    tavily_search = TavilySearchResults(max_results=2, search_depth='advanced',
max_tokens=1000)
    results = tavily_search.invoke(query)
    return results
```

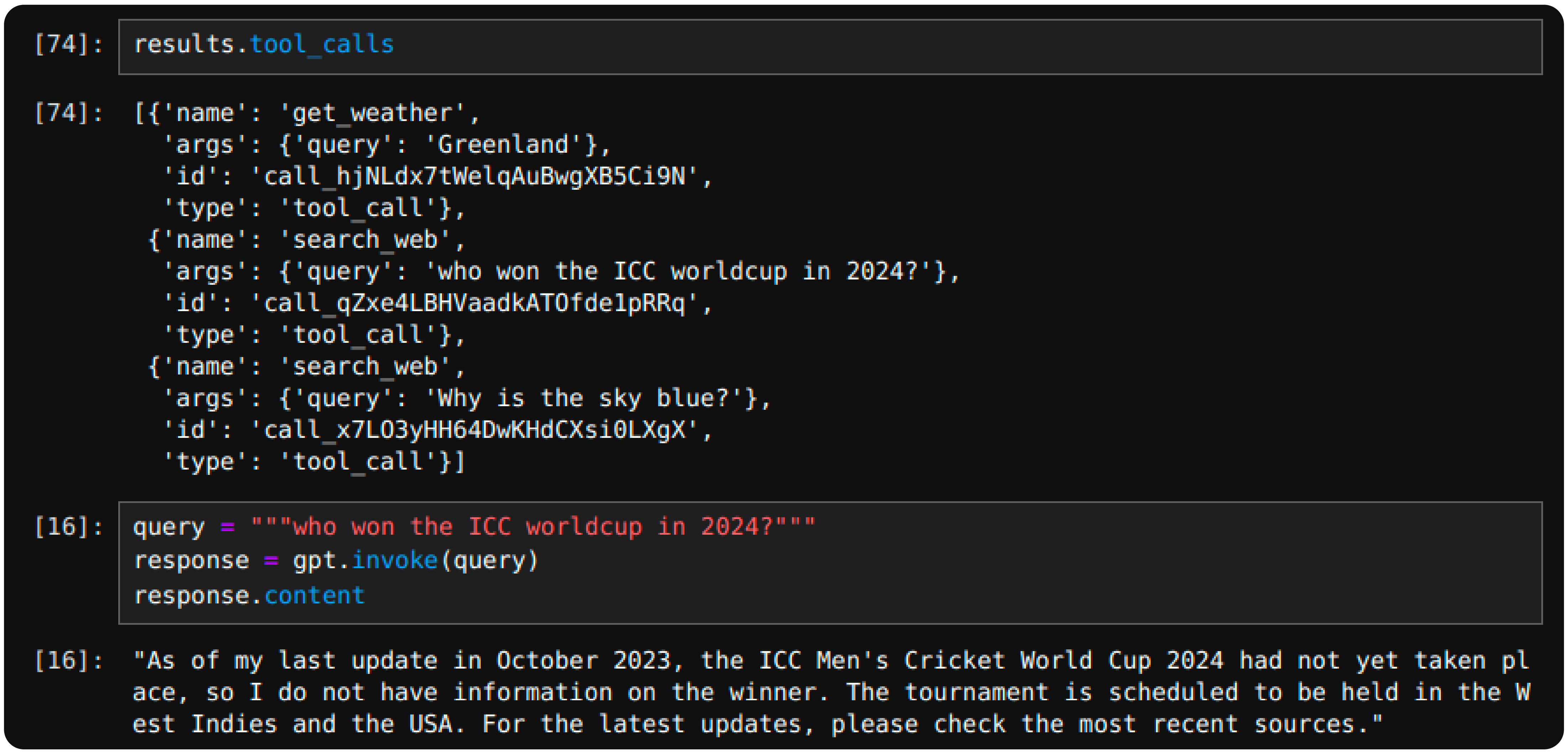
To make these tools available for the LLM, we can bind these tools to the LLM as follows:

```
gpt = ChatOpenAI(model="gpt-4o-mini", temperature=0)

tools = [search_web, get_weather]

gpt_with_tools = gpt.bind_tools(tools)
```

Now, let’s invoke the LLM to with a prompt to see the results:

```
prompt = """
         Given only the tools at your disposal, mention tool calls for the following tasks:

         Do not change the query given for any search tasks

         1. What is the current weather in Greenland today

         2. Can you tell me about Greenland and its capital

         3. Why is the sky blue?
      """

results = gpt_with_tools.invoke(prompt)

results.tool_calls
```

The results will be the following:

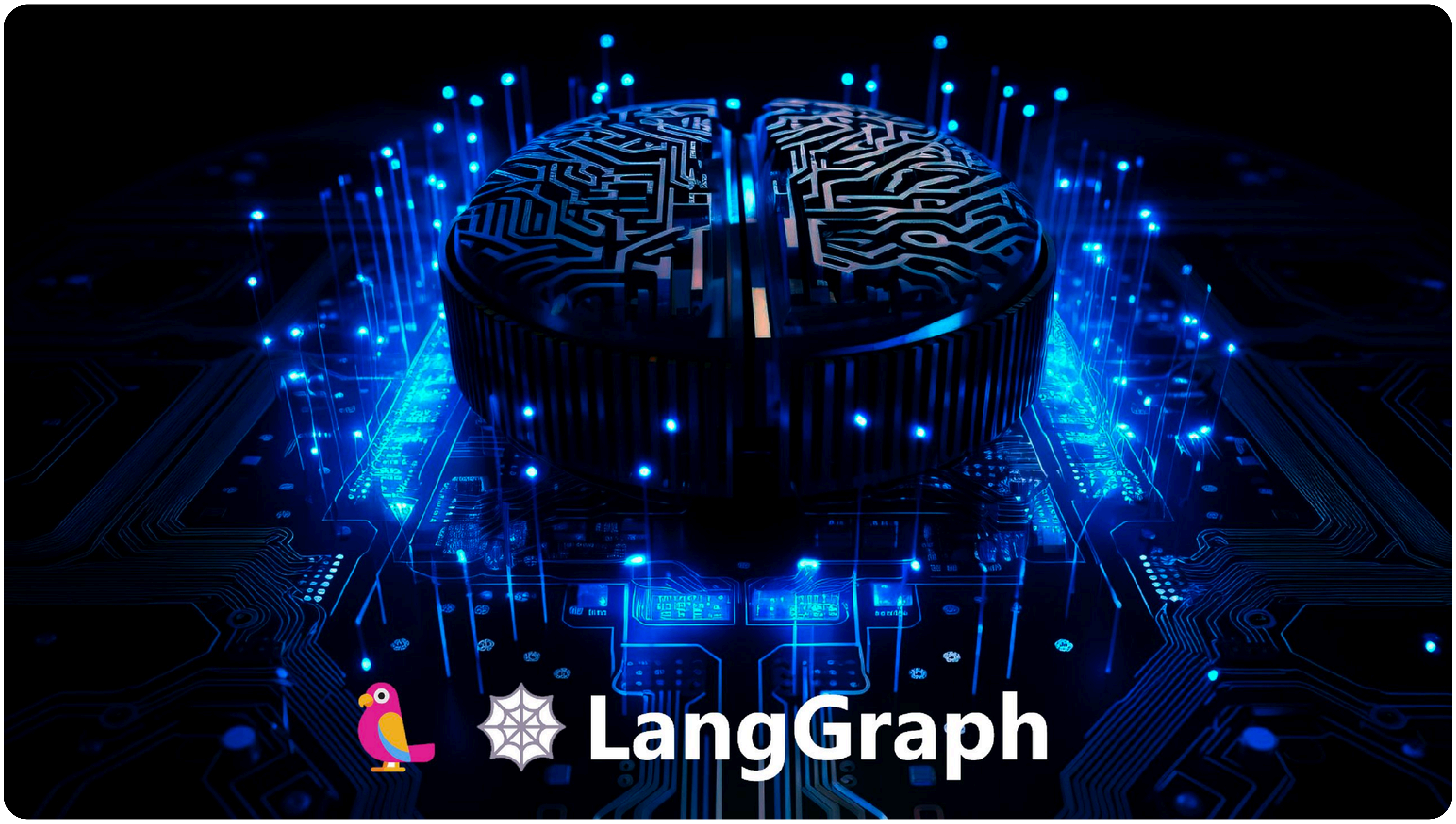


```
[74]: results.tool_calls

[74]: [{'name': 'get_weather',
        'args': {'query': 'Greenland'},
        'id': 'call_hjNLdx7tWelqAuBwgXB5Ci9N',
        'type': 'tool_call'},
       {'name': 'search_web',
        'args': {'query': 'who won the ICC worldcup in 2024?'},
        'id': 'call_qZxe4LBHVaadkATOfde1pRRq',
        'type': 'tool_call'},
       {'name': 'search_web',
        'args': {'query': 'Why is the sky blue?'},
        'id': 'call_x7LO3yHH64DwKHdCXsi0LXgX',
        'type': 'tool_call'}]

[16]: query = """who won the ICC worldcup in 2024?"""
      response = gpt.invoke(query)
      response.content

[16]: "As of my last update in October 2023, the ICC Men's Cricket World Cup 2024 had not yet taken pl
      ace, so I do not have information on the winner. The tournament is scheduled to be held in the W
      est Indies and the USA. For the latest updates, please check the most recent sources."
```

As we see, when we ask about the weather, get_weather tool is called.

The GPT model doesn’t know the who won ICC worldcup in 2024, as it is updated with information upto October 2023 only. So, when we ask about this query, it is calling search_web tool.

For more information, kindly visit this article



Advanced Artificial Intelligence Generative AI Large Language Models LLMs

What is LangGraph?

LangGraph, built on Langchain, enables the creation of cyclic graphs for AI agents, supporting flexible multi-agent coordination and more.

Sahitya Arya 04 Sep, 2024