

LangGraph

<https://www.langchain.com/langgraph>

Gain control with LangGraph to design agents that reliably handle complex tasks.

If LangChain can build AI Agents, why do we need LangGraph?

LangGraph is different from LangChain. Basically you can create Agents in LangGraph without using LangChain. Ofcourse you can use LangChain inside LangGraph if you want.

Why LangGraph?

If you are a conventional developer and you are building everything by yourself. When you write code, you know exactly what input you want and what output you are getting and how much time it will take. But as we move towards AI, where you are creating functions then if you want to generate some data you will get it from LLM. The next step is, what if you already have certain functions and AI is asking you call a particular function (while loop). We are giving some control to LLM which function to call. Say if you go to the next step, where you want to book a movie ticket or plan your itinerary and you want AI Agents to do everything for you, that's more of an Autonomous work you want AI to do. If you want to reach that level of autonomy, LangChain is giving you the flexibility to do that but not the control.

In this case, we have one Agent and two Tools

```
search_tool = DuckDuckGoSearchRun()
llm = ChatOpenAI(model="gpt-4o")
tools = [get_current_date_time, search_tool]
llm_with_tools = llm.bind_tools(tools=tools)
```

This Agent is using one of the tools that's search_tool and also there is a flow in which sequence it is going to execute the two tools: based on the prompt we pass-in, Agent will decide. Do we even need a tool? If yes, which tool we want. Agent decides at run-time which tool to call for the given request query. Sequence of tool calling is not fixed, it is based on what LLM is thinking during prompt. On one hand, you are giving a lot of flexibility to LLM, on the other hand, LLM could hallucinate and skip calling 'search_tool' multiple times and that could cost us a lot of tokens. *What it means is that we have to give flexibility to LLM to determine the sequence of tool calling but also we need to have some degree of control.* How can we achieve that where LLM can call tools independently but also we get some control? How do you do that trade-off where LLM can call any tools but you also want that control?

workflow or agent. LangGraph does not abstract prompts or architecture, and provides the following central benefits:

- **Durable execution**: Build agents that persist through failures and can run for extended periods, resuming from where they left off.
- **Human-in-the-loop**: Incorporate human oversight by inspecting and modifying agent state at any point.
- **Comprehensive memory**: Create stateful agents with both short-term working memory for ongoing reasoning and long-term memory across sessions.
- **Debugging with LangSmith**: Gain deep visibility into complex agent behavior with visualization tools that trace execution paths, capture state transitions, and provide detailed runtime metrics.
- **Production-ready deployment**: Deploy sophisticated agent systems confidently with scalable infrastructure designed to handle the unique challenges of stateful, long-running workflows.

https://docs.langchain.com/oss/python/langgraph/overview?_gl=1*1w7fs1j*_gcl_au*Njc4OTU5NjYxLjE3NjM3NzAxNzM.*_ga*MjAxNjYyMjMuMTc2Mzc3MDE3Mw..*_ga_47WX3HKKY2*cze3Njc5ODU1OTMkbzckZzEkdDE3Njc5ODU2OTlkajI4JGwwJGgw

‘Human-in-loop’: How do we do it? *We want LLM to ask you some permissions or validation in between the loop.* Validation like should I go ahead or continue? LangChain could do it but the lines of code you are going to write is huge. The more code you write, the more bugs you will put in the system. It is better to delegate that kind of work to someone else. *And that someone else is LangGraph.* Whenever you want to create multiple agents, let's use LangGraph, which will take care of a lot of things from you. LangGraph will help gain control of LLMs without writing a lot of code in a nutshell.

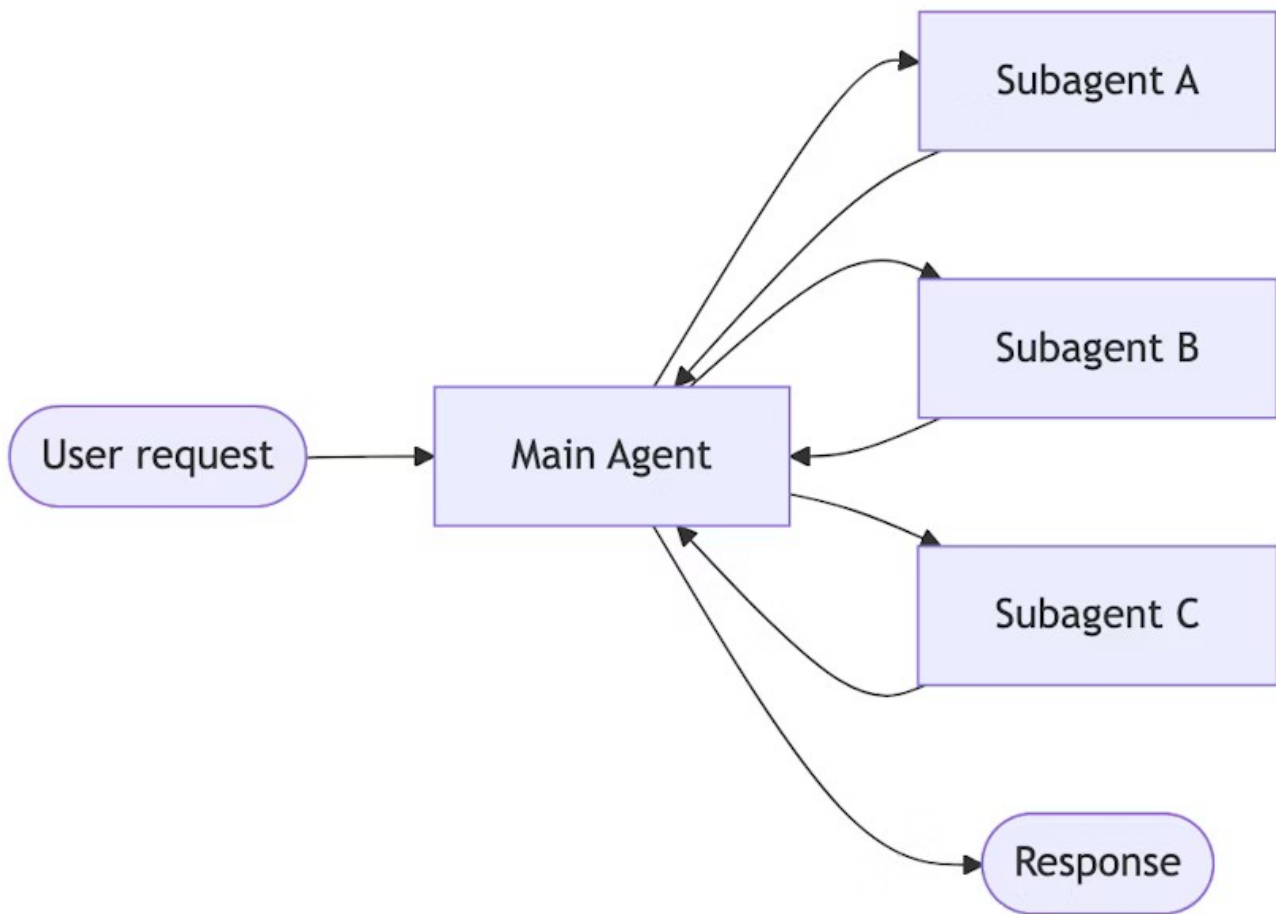
multi_agent.py

In this case, we want agent to take care of the entire ReAct while loop

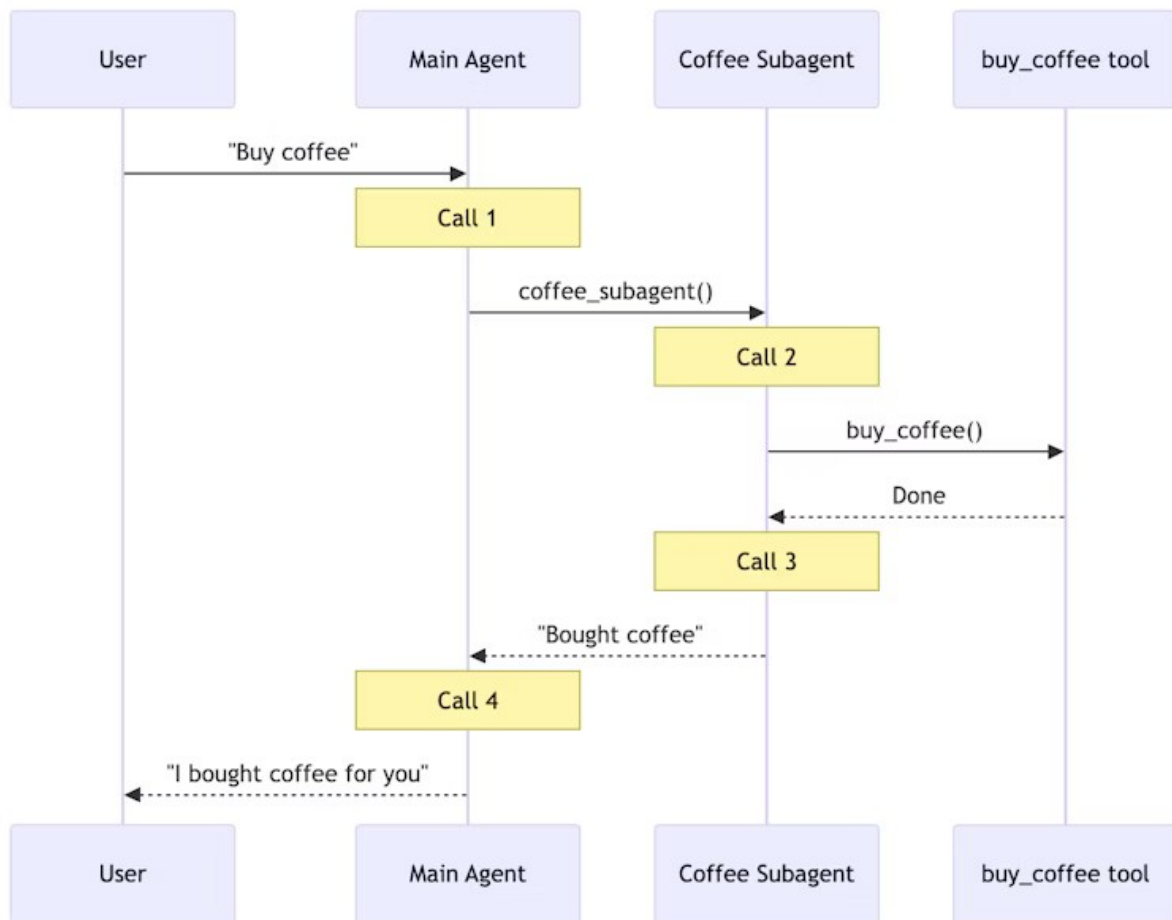
```
agent = create_agent(model=llm, tools=tools, system_prompt="You are a helpful AI agent that uses tools to answer user queries.")
```

Instead of using one Agent, we are going to create two agents.

<https://docs.langchain.com/oss/python/langchain/multi-agent>



In our case, Main Agent is the 'supervisor_agent', supervisor_agent can use multiple sub-agents. Say you have given a big task, it can break the task into multiple tasks, first agent will do this, second agent will do that and so on. All subagents will send responses to the main agent then main agent will give the final response. The idea of the sub-agent is to help the main agent.



User sends query to Main agent, main agent calls the subagents, subagent can call another subagent. Then the response goes back to Main agent then Main agent gives response to the user.

We got a `research_agent`, which is a subagent. Main agent is `'supervisor_agent'`. In the first agent, if you observe, we are passing only `search_tool`. `supervisor_agent` uses only the tool not the agent. You have to wrap your agent into a tool. Since in this program “multi-agent.py”, we are not using `LangGraph`, we are wrapping that Agent into a tool. We are passing that tool to the main agent. `supervisor_agent` is accepting main tools that's `get_current_local_time` and `research_web`.

`supervisor_executor.invoke({"input": query}) =>` first calls the `supervisor_agent`
`supervisor_agent` knows it needs the help of `researcher_agent`

There is a lot of loop holes in the code without `LangGraph`, that's what `LangGraph` is going to solve.

In short, User request (current local time in which country there was a FIFA worldcup in 2022) first goes into Main agent (`supervisor_agent`), Main agent knows there is a tool available called “search”, but this tool “search” is available with the subagent (`research_agent`), request goes there, gets the response (it's in Qatar). Once that's done, it executes another tool (not agent) available called “`get_current_date_time()`” and based on that response, LLM will give you the final response.

We are wrapping another agent “research_agent” inside a tool (“research_web”) so the Main agent will be using a tool

```
main_tools = [get_current_local_time, research_web]
```

Basic idea is we can create multiple agents. If you want that hierarchy, we have to create a main agent and main agent is going to call subagents.

Next class we will start with LangGraph.