Analytics Vidhya

The Command construct in LangGraph - Dynamic Navigation

Dipanjan Sarkar

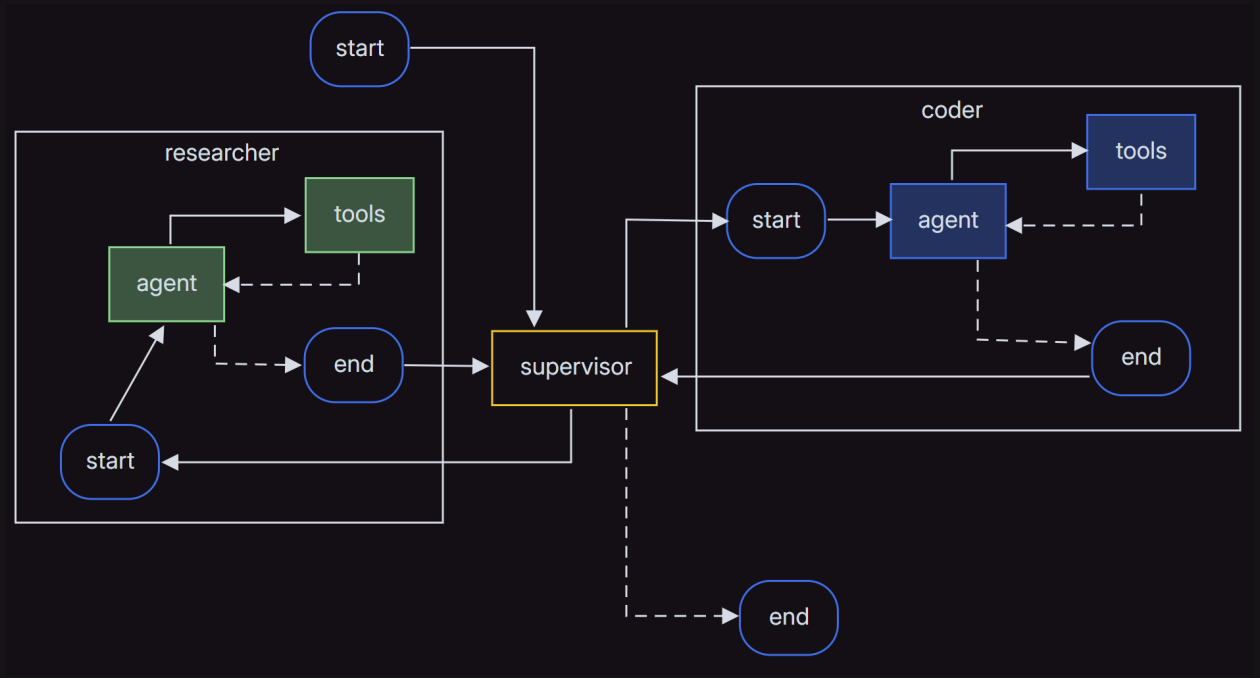
Head of Community & Principal AI Scientist at Analytics Vidhya

Google Developer Expert - ML & Cloud Champion Innovator

Published Author

Dynamic Agent Navigation with Command

In LangGraph, the Command object is pivotal for orchestrating complex workflows involving multi-agent systems. It facilitates dynamic state updates and directs the flow of execution among agents with definite ‘commands’.

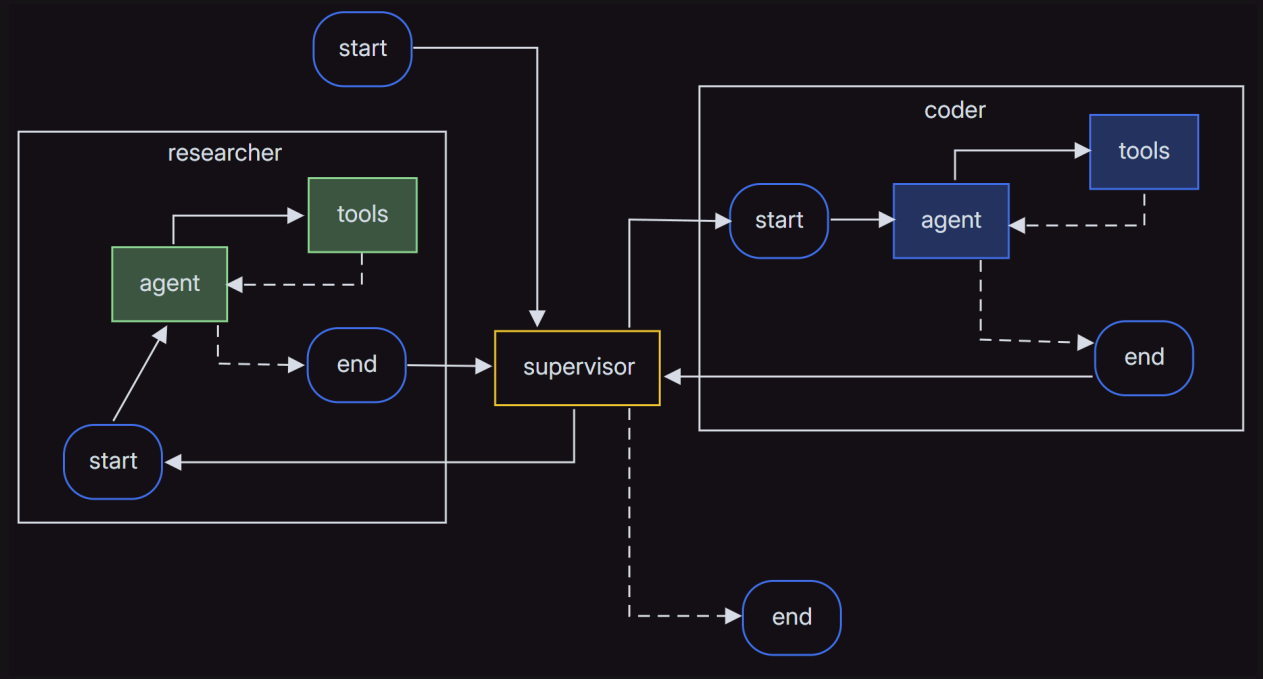


Dynamic Agent Navigation with Command

Key Points

Dynamic Workflow Control

Sub-agents or even single nodes can return a Command object to specify the next agent or node to goto after doing the necessary computations and state updates, enabling flexible and dynamic routing within the graph.

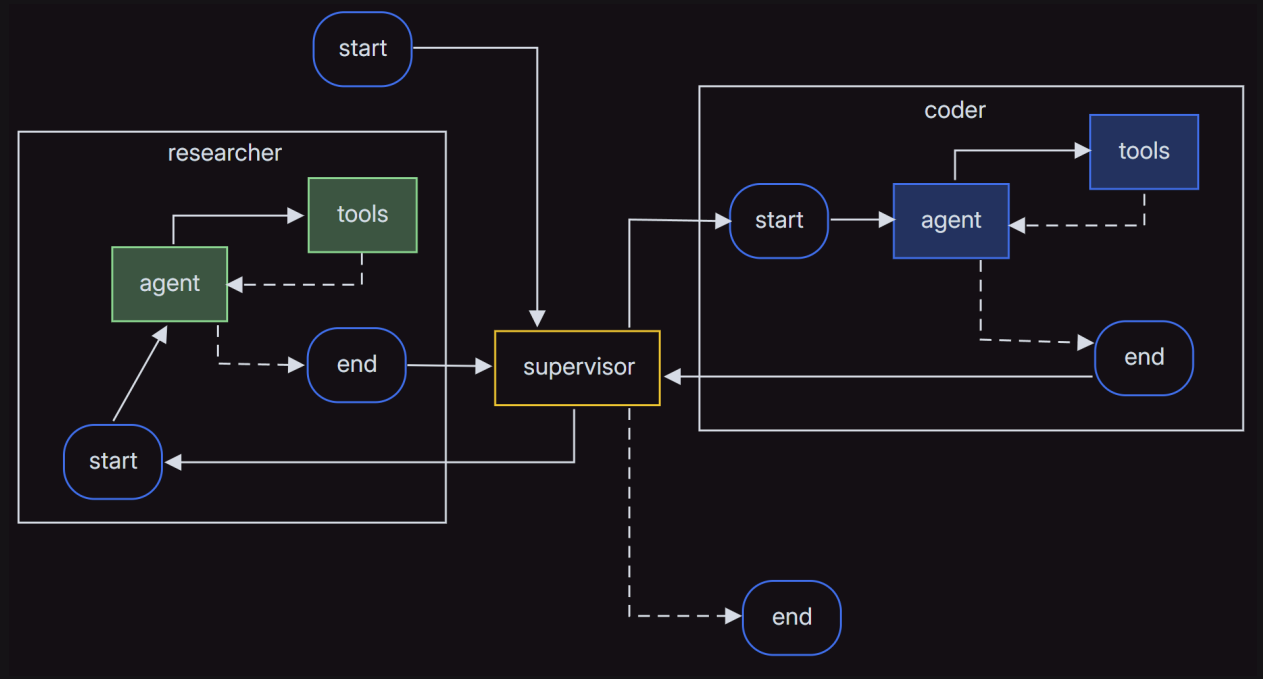


Dynamic Agent Navigation with Command

Key Points

State Management

The Command object allows sub-agents to update the shared state schema, ensuring that all agents have access to the latest information necessary for task execution.

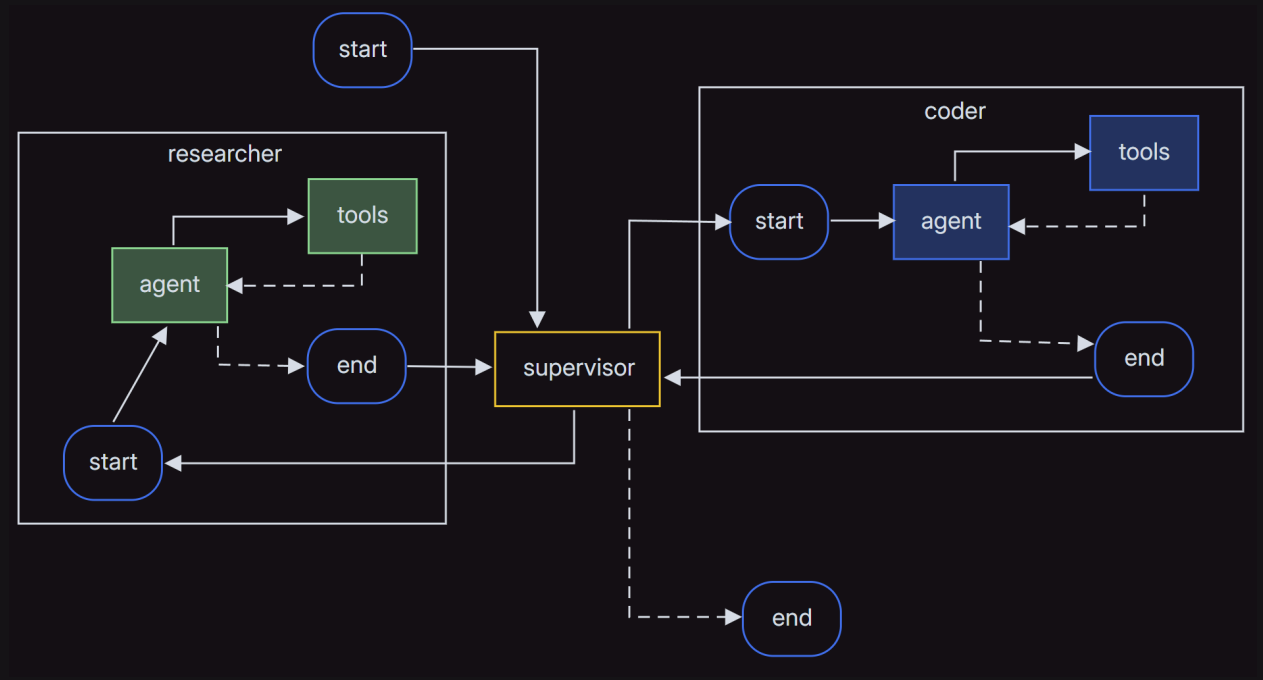


Dynamic Agent Navigation with Command

Key Points

Supervisor Agent Coordination

A supervisor agent can use the Command **pattern** to delegate tasks to sub-agents based on requirements. Sub-agents, upon completion, can redirect the workflow back to the supervisor using the same pattern, enabling efficient task distribution and streamlined supervision.

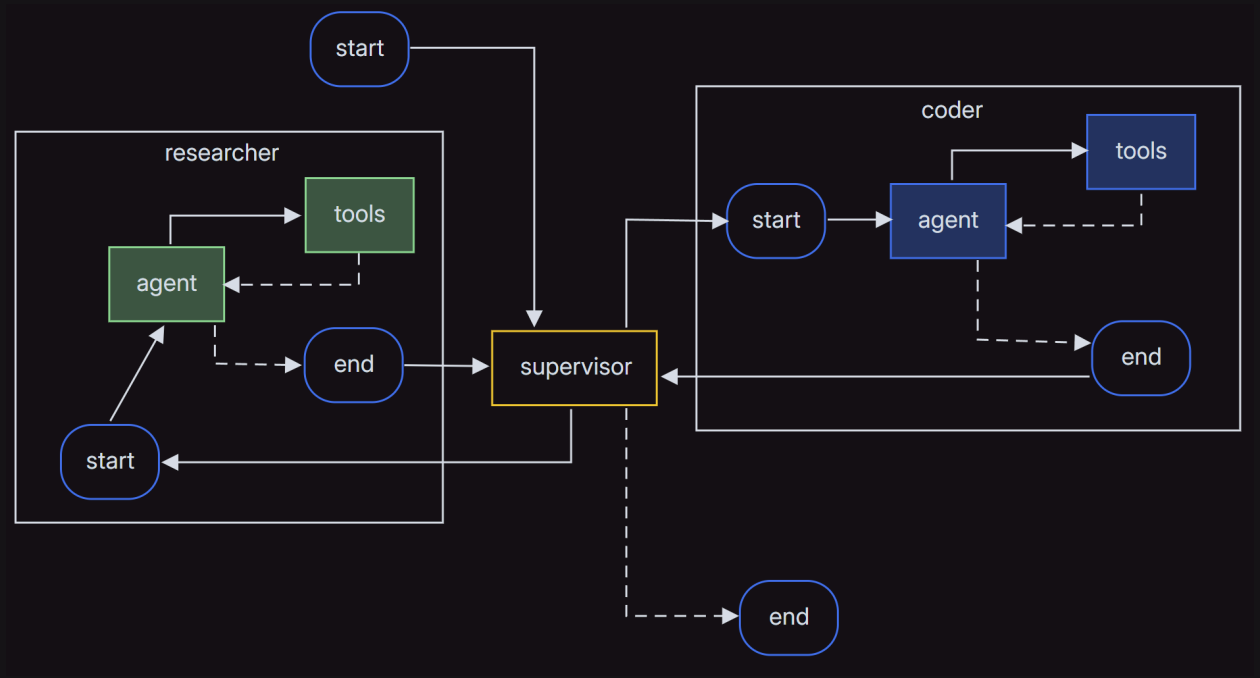


Dynamic Agent Navigation with Command

Key Points

Implementation Example

In a multi-agent system, a supervisor agent can delegate tasks to a researcher agent for web searches and a coder agent for plotting or computational analysis. Using the Command construct, sub-agents execute tasks independently and report back to the supervisor seamlessly.



Dynamic Agent Navigation with Command

The Command object allows agents to dynamically delegate tasks and manage workflow transitions in a multi-agent system.

Supervisor Node

- Acts as the central decision-maker.
- Evaluates the task context and routes it to the appropriate sub-agent (e.g., researcher or coder) using the goto property in the Command construct, ensuring precise task allocation and execution.

```python
# uses Command to delegate tasks to researcher or coder
def supervisor_node(state: State) -> Command[Literal["researcher", "coder", "__end__"]]:
    messages = [{"role": "system",
                 "content": SUPERVISOR_AGENT_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})

# searches the web, gets information and uses Command to report back to supervisor
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",
    )

# creates and runs code and uses Command to report back to supervisor
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",
    )
```

Dynamic Agent Navigation with Command

The Command object allows agents to dynamically delegate tasks and manage workflow transitions in a multi-agent system.

Researcher Node

- Handles information gathering (e.g., web search or querying data sources).
- Uses Command to send task results back to the supervisor by updating the state with the gathered content (messages) and directing the flow to the supervisor.

```python
# uses Command to delegate tasks to researcher or coder
def supervisor_node(state: State) -> Command[Literal["researcher", "coder", "__end__"]]:
    messages = [{"role": "system",
                 "content": SUPERVISOR_AGENT_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})

# searches the web, gets information and uses Command to report back to supervisor
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",
    )

# creates and runs code and uses Command to report back to supervisor
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",
    )
```

Dynamic Agent Navigation with Command

The Command object allows agents to dynamically delegate tasks and manage workflow transitions in a multi-agent system.

Coder Node

- Executes programming or code-based tasks.
- Similar to the researcher, it updates the shared state with the output (messages) and routes the control back to the supervisor.

```python
# uses Command to delegate tasks to researcher or coder
def supervisor_node(state: State) -> Command[Literal["researcher", "coder", "__end__"]]:
    messages = [{"role": "system",
                 "content": SUPERVISOR_AGENT_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})

# searches the web, gets information and uses Command to report back to supervisor
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",
    )

# creates and runs code and uses Command to report back to supervisor
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",
    )
```

Thanks