

Multi Agent Systems with LangGraph and CrewAI

Demo 1

Demo: Build a Customer Support Chatbot with LangGraph

Overview

This practical project leads you through creating a customer support chatbot with LangGraph. LangGraph enables us to coordinate various AI agents with specified workflows, resulting in more intricate and powerful systems than those using a single agent. This project emphasizes the hands-on execution of multi-agent systems, showcasing methods to devise, construct, and link various agents in a customer support environment. The chatbot is designed to assist customers with travel and hotel inquiries. By leveraging LangGraph, we'll create a system where multiple agents collaborate to provide a comprehensive customer experience.

Scenario

Envision a travel agency called "World-Hop Travelers" aiming to enhance its customer service. At present, customers frequently need to remain on hold or exchange emails with various departments to receive responses to their travel and lodging inquiries. This procedure is sluggish, unproductive, and may result in customer dissatisfaction. World-Hop Travelers imagines a chatbot capable of quickly responding to frequent customer questions, offering travel destination ideas, and suggesting hotels, all within a smooth and uninterrupted dialogue.

Problem Statement

World-Hop Travelers encounters these challenges:

- Isolated Information: Details about travel destinations and hotel suggestions are frequently managed by distinct teams, resulting in disjointed customer experiences.
- Scalability: Satisfying maximum demand with human agents is expensive and challenging.
- Response Time: Customers anticipate prompt responses, a challenge to deliver through conventional support methods.
- Personalization: Offering customized suggestions necessitates comprehending customer preferences and the situation.

- Complicated Queries: Managing intricate queries that necessitate collaboration among various fields of expertise is challenging.

Solution

To tackle these issues, their Chief Technical Officer has recommended and determined with the stakeholders to create a multi-agent customer service chatbot utilizing LangGraph. The chatbot will be made up of two agents:

- Travel Consultant: Tasked with suggesting travel locations tailored to client tastes.
- Hotel Consultant: Tasked with suggesting hotels according to the selected location and client preferences.

LangGraph will coordinate the interaction among these agents, facilitating a smooth transfer of the dialogue and a cohesive customer experience.



Stepwise Solution

Project Structure:

The project consists of a single Jupyter Notebook: Building Multi AI Agents Chatbots With LangGraph.ipynb contains all the code necessary to define the agents, their tools, the LangGraph graph, and the conversation flow.

NOTE: This Demonstration has been performed on [Google Colab Notebook](#).

Step 1: Installation and Setup



```
1 pip install langchain langgraph
2 pip install langchain-anthropic anthropic
3 import os
4
5 # Access the secret
6 ANTHROPIC_KEY = userdata.get("ANTHROPIC_KEY")
7
8 # Set it as environment variable (LangChain and Google SDK use this)
9 os.environ["ANTHROPIC_KEY"] = ANTHROPIC_KEY
10
11 from langchain_anthropic import ChatAnthropic
12 from langgraph.graph import MessagesState, StateGraph, START
13 from langgraph.prebuilt import create_react_agent, InjectedState
14 from langgraph.types import Command, interrupt
15 from langgraph.checkpoint.memory import MemorySaver
16 from langchain.tools import BaseTool
17 from typing import Any
18
19 model = ChatAnthropic(model="claude-3-5-sonnet-latest")
```

- **pip install langchain langgraph langchain-anthropic anthropic:** Installs the necessary libraries: LangChain, LangGraph, Anthropic's LangChain integration, and the Anthropic Python client.
- **import os:** Imports the os module for accessing environment variables.
- **ANTHROPIC_KEY = userdata.get("ANTHROPIC_KEY"):** Retrieves the Anthropic API key from a secure source (likely Google Colab's secrets manager).

- **os.environ["ANTHROPIC_KEY"] = ANTHROPIC_KEY:** Sets the Anthropic API key as an environment variable. LangChain and the Anthropic SDK rely on this environment variable for authentication.
- **from langchain_anthropic import ChatAnthropic ... from typing import Any:** Imports the necessary classes and functions from LangChain, LangGraph, and the typing module.
- **model = ChatAnthropic(model="claude-3-5-sonnet-latest"):** Creates an instance of the Anthropic Claude model, specifying "claude-3-5-sonnet-latest" as the model to use. This is the language model that powers the agents. Remember to check the latest model options.

Step 2: Define the MultiAgentState



```
1 class MultiAgentState(MessagesState):
2     last_active_agent: str
```

- **class MultiAgentState(MessagesState):** Defines a custom state class for the LangGraph graph. It inherits from MessagesState, which stores the conversation history (messages).
- **last_active_agent: str:** Adds a new attribute, last_active_agent, to the state. This attribute keeps track of which agent was the last one to respond. This is crucial for routing user input back to the correct agent after a handoff.

Step 3: Define the Travel Advisor's Functionality

```

● ● ●
1 def get_travel_recommendations(query: str) -> str:
2     """
3         This function takes a user query about travel recommendations and returns a string containing travel recommendations.
4
5     Args:
6         query: The user's query about travel recommendations.
7
8     Returns:
9         A string containing travel recommendations.
10    """
11    # Basic keyword-based recommendations
12    if "beach" in query.lower():
13        recommendations = "For a beach vacation, I recommend Hawaii, Bali, or the Maldives."
14    elif "mountain" in query.lower():
15        recommendations = "For a mountain getaway, consider the Swiss Alps, the Rockies, or the Himalayas."
16    elif "city" in query.lower():
17        recommendations = "For a city break, I suggest exploring New York, Paris, or Tokyo."
18    else:
19        recommendations = "I recommend visiting popular destinations like Hawaii, Bali, or Paris." # Default recommendations
20
21    return recommendations

```

- **def get_travel_recommendations(query: str) -> str:** Defines a function that takes a user query as input and returns travel recommendations.
- **if "beach" in query.lower(): ...:** This is a basic implementation that uses keyword matching to determine the type of travel recommendation to provide.

Important: This is a simplified example. In a real-world application, you would use more sophisticated techniques like natural language understanding (NLU) to better understand the user's intent and preferences.

Step 4: Create a Handoff Tool

```

● ● ●
1 def make_handoff_tool(agent_name: str) -> BaseTool:
2     class HandoffTool(BaseTool):
3         name: str = f"handoff_to_{agent_name}" # Type annotation added
4         description: str = f"Hand off the conversation to {agent_name}." # Type annotation added
5
6         def _run(self, tool_input: str, # type: ignore
7                 **kwargs: Any, # Type annotation modified
8                 ) -> str:
9             # Added an indented block here
10            return f"Switching to {agent_name}"
11
12         async def _arun(self, tool_input: str, # type: ignore
13                         **kwargs: Any, # Type annotation modified
14                         ) -> str:
15             # Added an indented block here
16             return f"Switching to {agent_name}"
17
18     return HandoffTool()

```

- **def make_handoff_tool(agent_name: str) -> BaseTool::** This function dynamically creates a BaseTool (from LangChain) that allows one agent to hand off the conversation to another agent.
- **class HandoffTool(BaseTool)::** Defines a custom tool class that inherits from BaseTool.
- **name: str = f"handoff_to_{agent_name}"**: Sets the name of the tool to handoff_to_{agent_name}, where {agent_name} is the name of the agent to hand off to (e.g., handoff_to_hotel_advisor).
- **description: str = f"Hand off the conversation to {agent_name}."**: Sets the description of the tool.
- **def _run(self, tool_input: str, **kwargs: Any) -> str::** Defines the _run method, which is executed when the tool is called. It simply returns a string indicating that the conversation is being switched to the specified agent. This message is crucial for guiding the conversation flow in the LangGraph.
- **async def _arun(self, tool_input: str, **kwargs: Any) -> str::** Defines the asynchronous version of the _run method.

Step 5: Define the Travel Advisor Agent

```
● ● ●  
1 travel_advisor_tools = [  
2     get_travel_recommendations,  
3     make_handoff_tool(agent_name="hotel_advisor"),  
4 ]  
5  
6 travel_advisor = create_react_agent(  
7     model,  
8     travel_advisor_tools,  
9     prompt=  
10        "You are a general travel expert that can recommend travel destinations "  
11        "(e.g. countries, cities, etc). If you need hotel recommendations, ask 'hotel_advisor' for help."  
12        "You MUST include a human-readable response before transferring to another agent."  
13    ),  
14 )
```

- **travel_advisor_tools = [...]**: Defines a list of tools that the travel advisor agent can use. It includes the get_travel_recommendations function and the handoff_to_hotel_advisor tool.
- **travel_advisor = create_react_agent(...)**: Creates a LangChain agent using the create_react_agent function. This agent uses the Claude model, the defined tools, and a specific prompt.

- **prompt = (...)**: This is the most important part of defining the agent. It instructs the agent on its role, capabilities, and how to interact with the user and other agents. Crucially, it tells the agent to use the handoff_to_hotel_advisor tool when the user asks about hotel recommendations, and it requires the agent to provide a human-readable response before handing off.

Step 6: Define the Travel Advisor Invocation Function

```
1 def call_travel_advisor(state: MultiAgentState) -> Command:  
2     response = travel_advisor.invoke(state)  
3     update = {**response, "last_active_agent": "travel_advisor"}  
4     return Command(update=update, goto="human")
```

- **def call_travel_advisor(state: MultiAgentState) -> Command**: This function is responsible for invoking the travel advisor agent and processing its response.
- **response = travel_advisor.invoke(state)**: Invokes the travel advisor agent with the current state of the conversation.
- **update = {**response, "last_active_agent": "travel_advisor"}**: Updates the state with the response from the travel advisor and sets the last_active_agent to "travel_advisor". This ensures that the next user input will be routed back to the travel advisor if necessary.
- **return Command(update=update, goto="human")**: Creates a Command object, which tells LangGraph how to update the state and which node to transition to next. In this case, it updates the state with the agent's response and transitions to the "human" node (representing the user).

Step 7: Define the Hotel Advisor's Functionality and Agent (Similar to Travel Advisor)

This code defines the functionality and agent for the hotel advisor. It's very similar to the travel advisor, but with hotel-related logic. Key differences are the get_hotel_recommendations function and the prompt, which instructs the hotel advisor to hand off to the travel_advisor if the user asks about travel destinations.

```

● ● ●

1 def get_hotel_recommendations(query: str) -> str:
2 """
3     This function takes a user query about hotel recommendations and returns a string containing hotel recommendations.
4
5     Args:
6         query: The user's query about hotel recommendations.
7
8     Returns:
9         A string containing hotel recommendations.
10    """
11    # Basic keyword-based recommendations (You'll need to improve this logic)
12    if "beach" in query.lower():
13        recommendations = "For beach hotels, I recommend The Ritz-Carlton, Bali or Four Seasons Maui."
14    elif "city" in query.lower():
15        recommendations = "For city hotels, consider The Peninsula, Hong Kong or The Savoy, London."
16    else:
17        recommendations = "I recommend checking out hotels like The Ritz-Carlton or Four Seasons."
18
19    return recommendations
20
21 hotel_advisor_tools = [
22     get_hotel_recommendations, # Now using the defined function
23     make_handoff_tool(agent_name="travel_advisor"),
24 ]
25
26 hotel_advisor = create_react_agent(
27     model,
28     hotel_advisor_tools,
29     prompt=(
30         "You are a hotel expert that can provide hotel recommendations for a given destination. "
31         "If you need help picking travel destinations, ask 'travel_advisor' for help. "
32         "You MUST include a human-readable response before transferring to another agent."
33     ),
34 )
35
36 def call_hotel_advisor(state: MultiAgentState) -> Command:
37     response = hotel_advisor.invoke(state)
38     update = {**response, "last_active_agent": "hotel_advisor"}
39     return Command(update=update, goto="human")

```

Step 8: Define the Human Node

```

● ● ●

1 def human_node(state: MultiAgentState, config) -> Command:
2     user_input = interrupt(value="Ready for user input.")
3     active_agent = state["last_active_agent"]
4
5     return Command(
6         update={
7             "messages": [{"role": "human", "content": user_input}]
8         },
9         goto=active_agent,
10    )

```

- **def human_node(state: MultiAgentState, config) -> Command::** This function represents the interaction with the user.

- **user_input = interrupt(value="Ready for user input.")**: Uses the interrupt function from LangGraph to pause the execution and wait for user input. The "Ready for user input." message is displayed to the user.
- **active_agent = state["last_active_agent"]**: Retrieves the name of the last active agent from the state.
- **return Command(...)**: Creates a Command object that updates the state with the user's input and transitions to the node corresponding to the last active agent. This ensures that the user's input is routed to the correct agent.

Step 9: Build the LangGraph Graph

```

● ● ●

1 builder = StateGraph(MultiAgentState)
2
3 builder.add_node("travel_advisor", call_travel_advisor)
4 builder.add_node("hotel_advisor", call_hotel_advisor)
5 builder.add_node("human", human_node)
6
7 builder.add_edge(START, "travel_advisor") # Initial entry point
8 checkpointer = MemorySaver()
9 graph = builder.compile(checkpointer=checkpointer)

```

- **builder = StateGraph(MultiAgentState)**: Creates a StateGraph object, specifying the MultiAgentState as the state class.
- **builder.add_node(...)**: Adds the travel advisor, hotel advisor, and human interaction as nodes in the graph. Each node is associated with its corresponding invocation function.
- **builder.add_edge(START, "travel_advisor")**: Defines the starting point of the graph. The conversation begins with the travel advisor.
- **checkpointer = MemorySaver()**: Creates a MemorySaver object for checkpointing the state of the graph. This allows you to resume conversations later.
- **graph = builder.compile(checkpointer=checkpointer)**: Compiles the LangGraph graph, creating an executable object.

Step 10: Run the Conversation

```
1 import uuid
2
3 thread_config = {"configurable": {"thread_id": str(uuid.uuid4())}}
4
5 inputs = [
6     {"messages": [{"role": "user", "content": "i wanna go somewhere warm in the caribbean"}]},  

7     Command(resume="could you recommend a nice hotel in one of the areas and tell me which area it is."),  

8     Command(resume="i like the first one. could you recommend something to do near the hotel?"),
9 ]
10
11 for idx, user_input in enumerate(inputs):
12     print(f"\n--- Conversation Turn {idx + 1} ---\n")
13     print(f"User: {user_input}\n")
14     for update in graph.stream(user_input, config=thread_config, stream_mode="updates"):
15         for node_id, value in update.items():
16             if isinstance(value, dict) and value.get("messages", []):
17                 last_message = value["messages"][-1]
18                 if isinstance(last_message, dict) or last_message.type != "ai":
19                     continue
20                 print(f"{node_id}: {last_message.content}"
```

- **import uuid:** Imports the uuid module for generating unique thread IDs.
- **thread_config = {"configurable": {"thread_id": str(uuid.uuid4())}}:** Creates a configuration dictionary for the graph, including a unique thread ID. This is important for managing multiple concurrent conversations.
- **inputs = [...]:** Defines a list of user inputs to simulate a conversation. It includes the initial user query and subsequent commands to resume the conversation.
- **for idx, user_input in enumerate(inputs)::** Iterates through the list of user inputs, simulating a conversation turn by turn.
- **graph.stream(user_input, config=thread_config, stream_mode="updates"):** Executes the LangGraph graph with the current user input and configuration. The stream_mode="updates" argument specifies that the graph should stream updates as they occur.
- **for update in ...:** Iterates through the updates streamed by the graph.
- **for node_id, value in update.items(): ...:** Processes each update, printing the messages generated by each node (agent). It ensures it only prints the content from the latest message generated by the AI agent.

Multi-turn Conversation Output:

```

--- Conversation Turn 1 ---
User: {'messages': [{'role': 'user', 'content': 'i wanna go somewhere warm in the caribbean'}]}
travel_advisor: Let me help you explore some wonderful Caribbean destinations! The Caribbean is perfect for warm-weather getaways, offering crystal clear waters and sun-filled days. Here are some top picks:
1. The Bahamas - Known for its stunning beaches, clear waters, and excellent water activities
2. Jamaica - Famous for its vibrant culture, reggae music, and beautiful resorts
3. Dominican Republic - Offers great all-inclusive resorts, beautiful beaches, and historical sites
4. Turks and Caicos - Perfect for luxury travelers and beach enthusiasts
5. St. Lucia - Known for its dramatic Piton mountains and romantic atmosphere

Would you like me to suggest some specific areas within any of these destinations, or would you like help finding hotels in any of these locations?
--- Conversation Turn 2 ---
User: Command(resume='could you recommend a nice hotel in one of the areas and tell me which area it is.')
travel_advisor: I've connected you with our hotel advisor who can provide specific hotel recommendations in the Caribbean. They'll be able to suggest some great options for you.
--- Conversation Turn 3 ---
User: Command(resume='i like the first one. could you recommend something to do near the hotel?')
travel_advisor: I notice there might be some confusion - I don't have access to the specific hotel recommendation that was provided by the hotel advisor. You can either share the name of the hotel and its location that was recommended, or let me connect you back with the hotel advisor to review the specific recommendation.

Once you provide these details, I can give you specific activity recommendations for that area of the Caribbean.

```

Map with Topics

This hands-on project demonstrates several key concepts from the "Multi Agent Systems with LangGraph and CrewAI" module:

1. **Multi Agent Systems:** The project implements a system with two interacting agents (travel advisor and hotel advisor).
2. **Multi Agent Workflows:** LangGraph defines the workflow and interaction between the agents. The handover mechanism is a key part of this workflow.
3. **Collaborative Multi Agents:** The agents collaborate to provide a complete solution to the customer's travel and hotel inquiries.
4. **Multi Agent Designs:** The code showcases a design pattern for creating specialized agents and orchestrating their interactions.
5. **Multi Agent Workflow with LangGraph:** The entire project is built using LangGraph to define the multi-agent workflow. The StateGraph and Command objects are central to this topic.
6. **CrewAI Introduction, CrewAI Components, Setting up CrewAI environment, Building Agents with CrewAI:** While this project uses LangGraph instead of CrewAI for agent orchestration, the agent design principles and concepts are similar. Understanding how to define agent roles, tools, and prompts is relevant to both LangGraph and CrewAI. The concept of defining agent "tasks" (what the

agent is supposed to do) is mirrored in the prompts we give to the LangGraph agents. You could re-implement this same chatbot using CrewAI as a comparative exercise.

This Demonstration offers a strong basis for comprehending multi-agent systems and constructing them with LangGraph. By enhancing this project with additional advanced features and agents, you can develop even more robust and flexible customer support solutions.

