

Streaming and Persistence

viernes, noviembre 01, 2024 1:53 PM

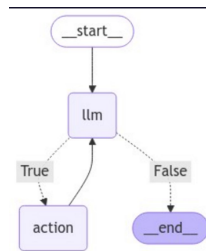
```
Now, instead of calling invoke() to run our model, we just stream over the graph, with the selected thread_id=1 we selected before
```

- In this way, we'll be getting all the intermediate steps and messages going on in our agent and tools until it delivers the end answer

```
for event in abot.graph.stream({"messages": messages}, thread_id=1):  
    for v in event.values():  
        print(v["messages"])
```

```
[AIMessage(content='', additional_kwargs={'tool_calls': [{'id': 'call_1', 'type': 'function', 'function': {'name': 'tavily_search_results_json', 'args': {'query': 'Los Angeles'}}]}),  
Calling: {'name': 'tavily_search_results_json', 'args': {'query': 'Los Angeles'}},  
Back to the model!  
[ToolMessage(content='[{\"url\": \"https://www.weatherapi.com/\", \"text\": \"The current weather in San Francisco is 13.\"}']', additional_kwargs={'tool_call_id': 'call_1'}),  
[AIMessage(content='The current weather in San Francisco is 13.')]
```

1. Adding persistence to the model.
2. Adding streaming
 - a. Both can be **synchronous**
 - b. Or **asynchronous**



Appendix (in notebooks) HUMAN IN THE LOOP INTERACTIONS

We can add interrupts in different nodes like node="action" so whenever the agent needs to run a node (e.g action), it will stop there, and from there we decide whether resuming the graph or stopping the agent

```
for event in abot.graph.stream({"messages": messages}, thread):  
    for v in event.values():  
        print(v)  
    while abot.graph.get_state(thread).next():  
        print("\n", abot.graph.get_state(thread), "\n")  
        _input = input("proceed?")  
        if _input != "y":  
            print("aborting")  
            break  
    for event in abot.graph.stream(None, thread):  
        for v in event.values():  
            print(v)
```

1. Persistence

The **persistence** allows us to checkpoint the state of the agent after and between every node

- a. It can be synchronous
- b. Or asynchronous



```
thread = {"configurable": {"thread_id": "1"}}
```

!! We are using threads.
This allows us in real world apps to
Have multiple conversations in parallel

- Whereas the agents, can remember from the Context.
E.g if you use a different thread_id, it won't remember anything

2. Streaming

```
abot.graph.stream({"messages": messages}, thread):
```

Depending on the **persistence** configuration (agent graph memory saver) we selected: we'll see the intermediate steps of the agent and output

A) It can be **synchronous**

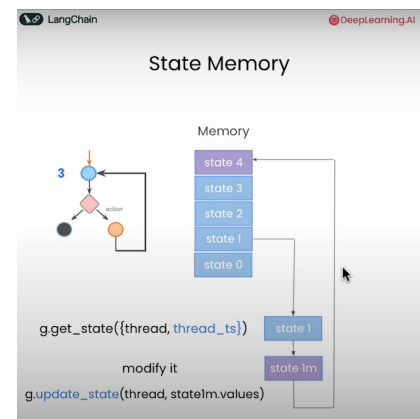
```
{'messages': [AIMessage(content='', additional_kwargs={'tool_calls': [{'id': 'call_1', 'type': 'function', 'function': {'name': 'tavily_search_results_json', 'args': {'query': 'Los Angeles'}}]}),  
Calling: {'name': 'tavily_search_results_json', 'args': {'query': 'Los Angeles'}},  
Back to the model!  
{'messages': [ToolMessage(content='[{\"url\": \"https://www.weatherapi.com/\", \"text\": \"The current weather in Los Angeles is 15.1\"}']', additional_kwargs={'tool_call_id': 'call_1'}),  
{'messages': [AIMessage(content='The current weather in Los Angeles is 15.1')]
```

B) Or it can be **asynchronous**

```
Calling: {'name': 'tavily_search_results_json', 'args': {'query': 'current price of tomatoes in San Francisco'}}  
Calling: {'name': 'tavily_search_results_json', 'args': {'query': 'current price of tomatoes in Barcelona'}}  
Back to the model!  
In| San| Francisco|,| the| average| price| for| 1| kilogram| (approximately|  
|In| Spain|,| specifically| in| Barcelona|,| the| price| for| 1| kilogram|  
|Compar|atively|,| tomatoes| are| significantly| more| expensive| in| San| Francisco|
```

The LLM now streams tokens too when running,
So we can see each generated token in real time

(this | was the separator
Between tokens)



Extra note: We can access to all the different "states" inside the llm, And change them

We can also "Mock out" responses for the llm instead of calling an Action node, so it directly goes to the llm...etc

(details in notebooks)