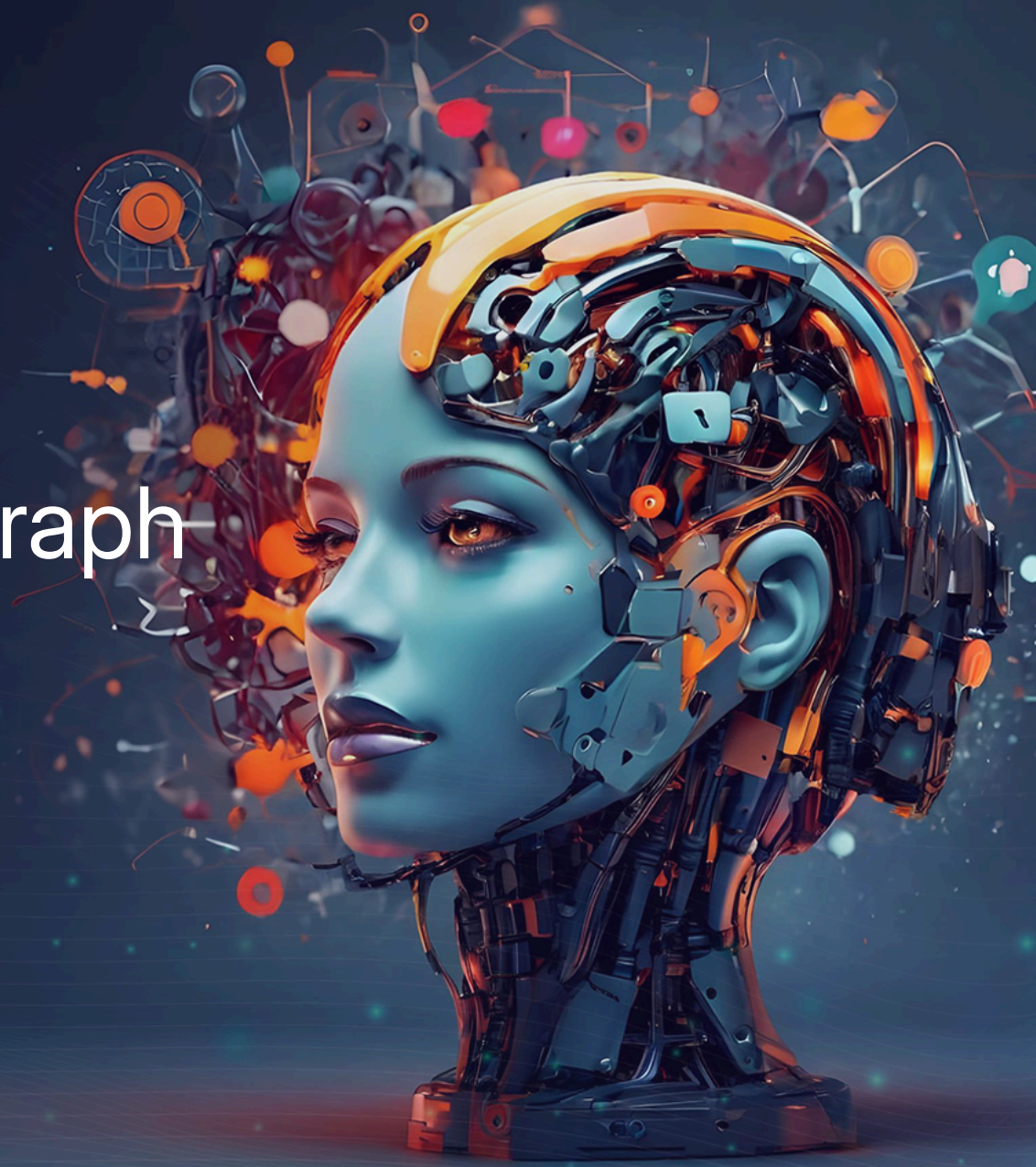
Analytics Vidhya

Memory & Threads in LangGraph for Multi-User Conversations



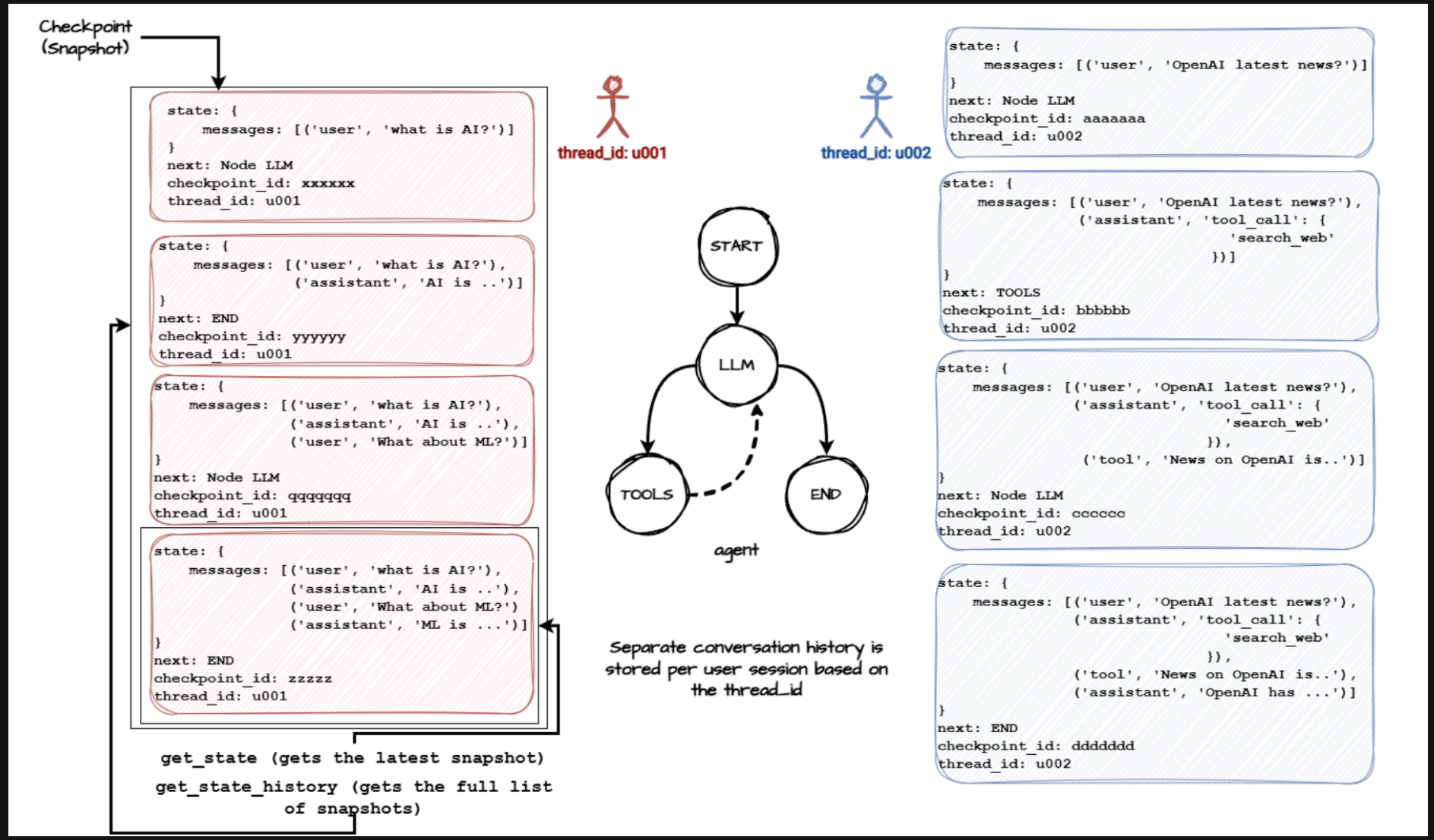
Dipanjan Sarkar

Head of Community & Principal AI Scientist at Analytics Vidhya

Google Developer Expert - ML & Cloud Champion Innovator

Published Author

Memory, Threads and Checkpointing in LangGraph

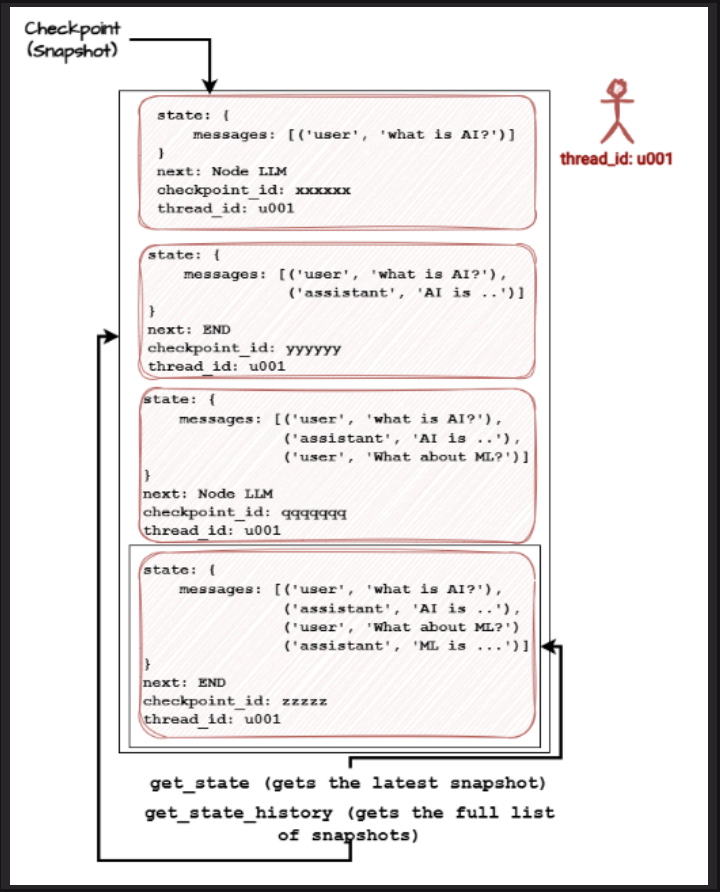


Analytics Vidhya

Memory, Threads and Checkpointing in LangGraph

Memory in LangGraph

- Utilizes a built-in persistence layer to maintain graph state across executions.
- Enables features like human-in-the-loop interactions, time travel, fault tolerance, and conversational capabilities

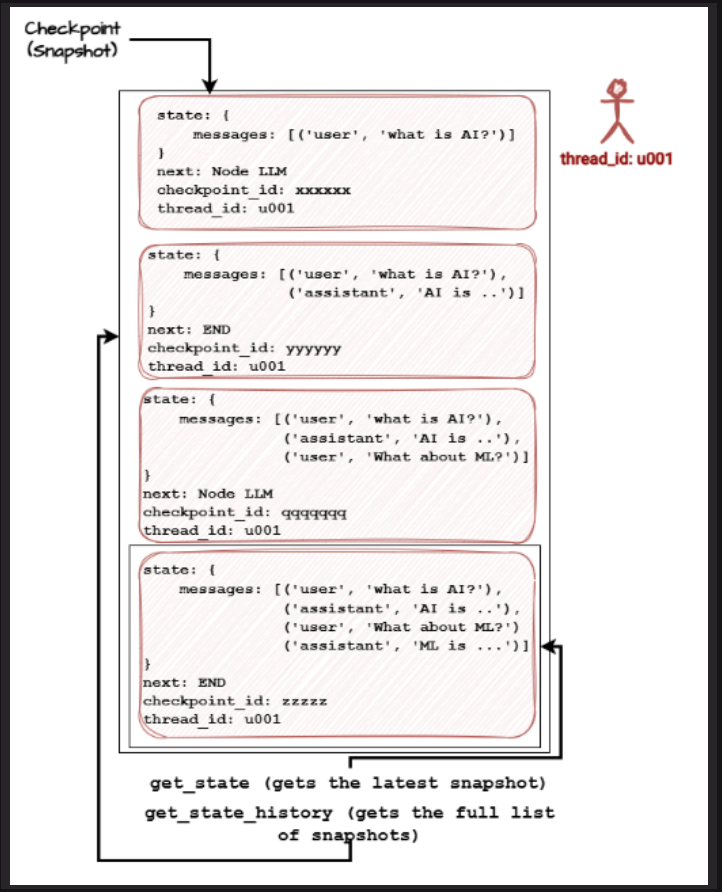


Analytics Vidhya

Memory, Threads and Checkpointing in LangGraph

Threads

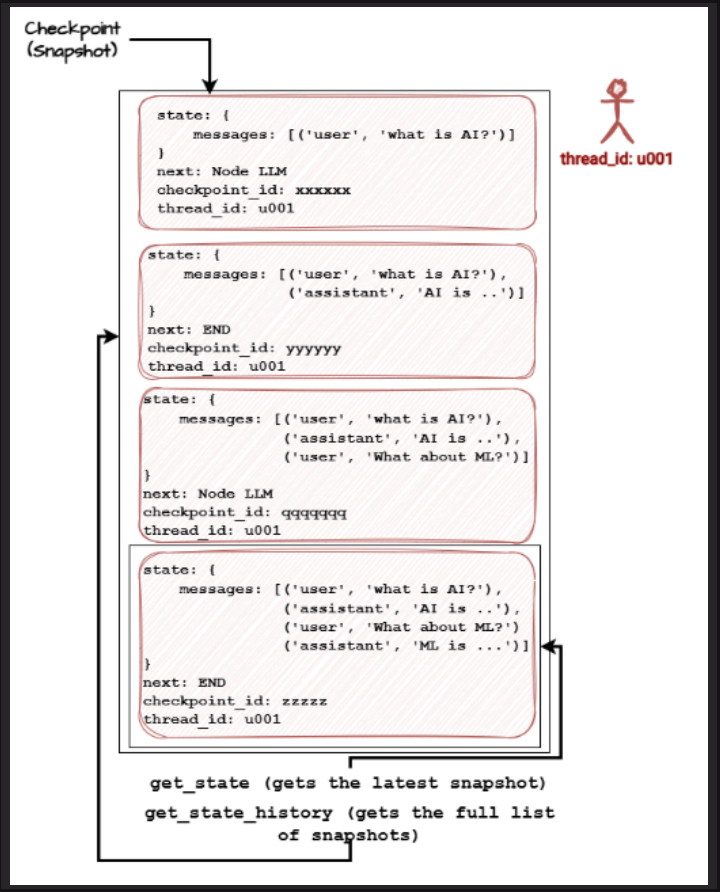
- Serve as unique identifiers for sequences of checkpoints (agent state snapshots)
- Allow retrieval and management of graph states post-execution
- Specified during graph invocation via {"configurable": {"thread_id": "user-session-id"}}
- User session ID can be generated and assigned per unique user or user session
- This is used by the agent to refer to past conversation and agent state history for any user session at any time
- Enables multi-user conversation for your agent



Memory, Threads and Checkpointing in LangGraph

Checkpoints

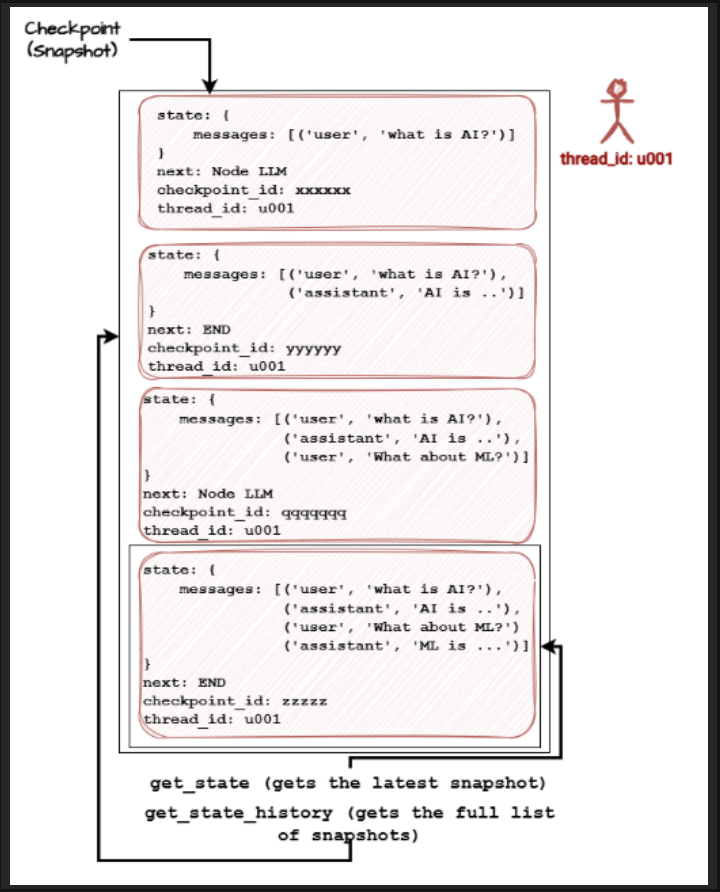
- Snapshots of the graph's state are saved after execution of each step (node) in your agent graph.
- Stored within threads, facilitating state recovery and inspection
- Contains configuration, metadata, current state schema values, and information on subsequent nodes



Memory, Threads and Checkpointing in LangGraph

Accessing State

- Retrieve Latest State
  - Use graph.get_state(config) with the appropriate thread_id
- View State History
  - Call graph.get_state_history(config) to obtain a chronological list of checkpoints

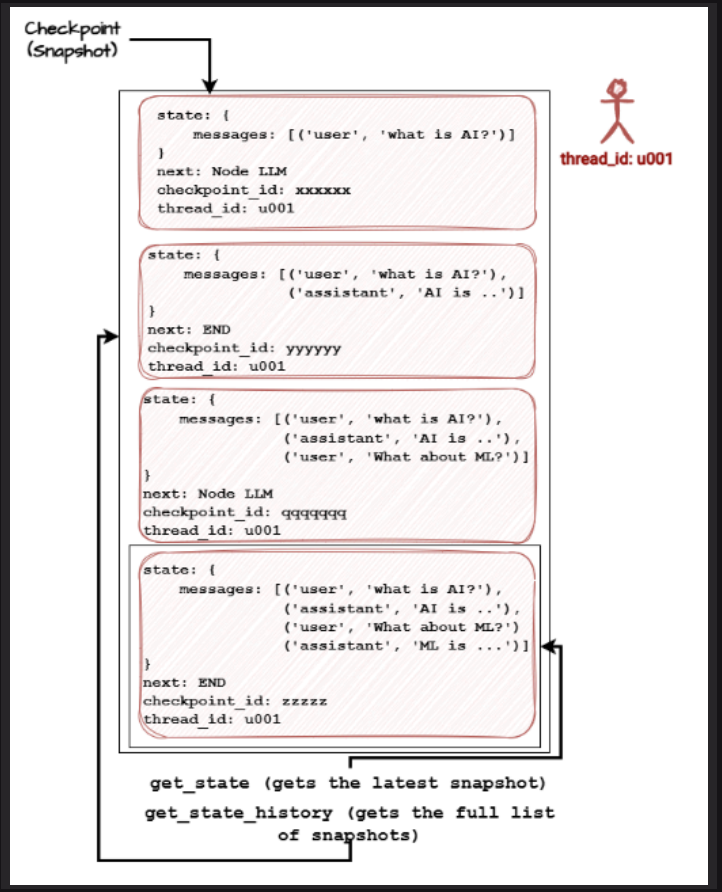


Analytics Vidhya

Memory, Threads and Checkpointing in LangGraph

Benefits

- Ensures continuity and context retention across multiple interactions.
- Facilitates debugging and enhances system reliability.



Analytics Vidhya

Memory, Threads and Checkpointing in LangGraph

Conversational Agent Graph with In-Memory Persistence

- Uses MemorySaver to enable transient, in-memory checkpointing for managing graph state during execution.
- Checkpoints are stored and accessed based on **unique thread IDs**, enabling separate sessions for multiple users.

```python
from langgraph.graph import StateGraph, START, END
from langgraph.prebuilt import ToolNode, tools_condition
from langgraph.checkpoint.memory import MemorySaver


# Build the graph
builder = StateGraph(State)
builder.add_node("tool_calling_llm", tool_calling_llm)
...
... # add nodes and edges
builder.add_edge("tools", END)

# add in-memory persistence (transient memory)
memory = MemorySaver()
agent = builder.compile(checkpointer=memory)

# separate conversation is stored per user session id (thread id)
uid = 'user001'
config = {"configurable": {"thread_id": uid}}
user_input = "Explain AI in 1 line"
for event in agent.stream(input={"messages": user_input},
                          config=config,
                          stream_mode='values'):
    event['messages'][-1].pretty_print()

# separate conversation is stored per user session id (thread id)
uid = 'user002'
config = {"configurable": {"thread_id": uid}}
user_input = "Tell me 3 latest OpenAI product releases"
for event in agent.stream(input={"messages": user_input},
                          config=config,
                          stream_mode='values'):
    event['messages'][-1].pretty_print()
```

Memory, Threads and Checkpointing in LangGraph

Separate Conversations by Thread ID

- Each user session is uniquely identified by a thread_id (e.g., user001, user002)
- Ensures state isolation for individual user conversations
- Conversation history can be controlled in terms of message length, number of messages, etc.

```python
from langgraph.graph import StateGraph, START, END
from langgraph.prebuilt import ToolNode, tools_condition
from langgraph.checkpoint.memory import MemorySaver


# Build the graph
builder = StateGraph(State)
builder.add_node("tool_calling_llm", tool_calling_llm)
...
... # add nodes and edges
builder.add_edge("tools", END)

# add in-memory persistence (transient memory)
memory = MemorySaver()
agent = builder.compile(checkpointer=memory)

# separate conversation is stored per user session id (thread id)
uid = 'user001'
config = {"configurable": {"thread_id": uid}}
user_input = "Explain AI in 1 line"
for event in agent.stream(input={"messages": user_input},
                          config=config,
                          stream_mode='values'):
    event['messages'][-1].pretty_print()

# separate conversation is stored per user session id (thread id)
uid = 'user002'
config = {"configurable": {"thread_id": uid}}
user_input = "Tell me 3 latest OpenAI product releases"
for event in agent.stream(input={"messages": user_input},
                          config=config,
                          stream_mode='values'):
    event['messages'][-1].pretty_print()
```

In-Memory Persistence vs. On Disk Persistence

```python
from typing import Annotated
from langchain_core.messages import BaseMessage
from typing_extensions import TypedDict
from langgraph.graph import StateGraph, START, END
from langgraph.graph.message import add_messages
from langgraph.checkpoint.memory import MemorySaver


class State(TypedDict):
    messages: Annotated[list, add_messages]

graph_builder = StateGraph(State)
tools = [...]
... # add nodes and edges

memory = MemorySaver()
agent = graph_builder.compile(checkpointer=memory)
... # call agent as usual
```

In-Memory Persistence (Transient Memory)

```python
from typing import Annotated
from langchain_core.messages import BaseMessage
from typing_extensions import TypedDict
from langgraph.graph import StateGraph, START, END
from langgraph.graph.message import add_messages


class State(TypedDict):
    messages: Annotated[list, add_messages]

graph_builder = StateGraph(State)
tools = [...]
... # add nodes and edges

agent = graph_builder.compile()

from langgraph.checkpoint.sqlite import SqliteSaver

with SqliteSaver.from_conn_string("memory.db") as memory:
        agent = agent_graph.compile(checkpointer=memory)
        ... # call agent in the memory scope
```

On-Disk Persistence (Permanent Memory)

Thanks