**Mastering LangGraph Basics through Documentation and Implementation**

**What is LangGraph?**

LangGraph is like a powerful toolbox to build smart AI assistants (called "agents") that can talk to people, remember things, ask for help from humans, and do their tasks step-by-step safely and clearly.

We use it when you want your AI to be more:

* Reliable (not go off track)
* Controllable (we can guide what it does)
* Customizable (we can change how it behaves)
* Interactive in real-time (we can watch and interact as it thinks)

**Why LangGraph is preferred?**

**1. Reliability and Control**

* We can watch over what the AI is doing.
* If needed, we can even pause it and ask a human to approve or help (this is called human-in-the-loop).
* It can remember conversations, so our agent doesn’t forget what it talked about 5 minutes ago.

**2. Flexible and Customizable**

* We can build exactly what you want.
* We are not stuck with pre-made bots , we can design your own agents with unique roles and behaviors.
* We can even make multi-agent systems where different agents do different tasks and talk to each other like a team.

**3. Streaming Support**

* We can watch the AI think in real-time, word by word.
* We can also see its reasoning steps as they happen, like seeing how it solves a puzzle.

**Overview**

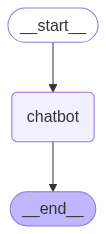
Chatbot will be able to:

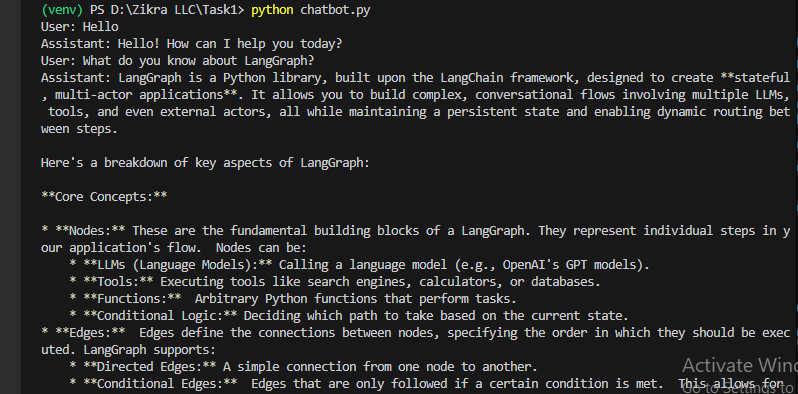
* Answer common questions using tools like search (basic chatbot and with tools)
* Remember the whole conversation so it doesn’t forget what you said before(Memory)
* Ask a human to step in for tough questions (human\_in\_the\_loop)
* Change how it behaves based on what’s happening (customization)
* Go back in time and explore different responses (time travel)

1. **Build a basic chatbot:**

In this tutorial, following objectives achieved:

* Built a basic chatbot using a StateGraph (LangGraph's central structure).
* Learned how to create nodes, manage state, and invoke LLMs.
* Set the foundation for more advanced features in later tutorials (memory, tool use, human-in-the-loop, etc.).





1. **Add tools:**

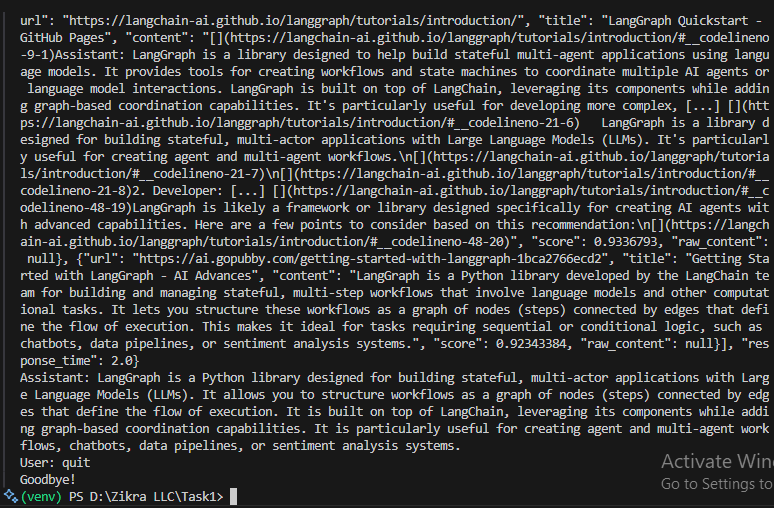
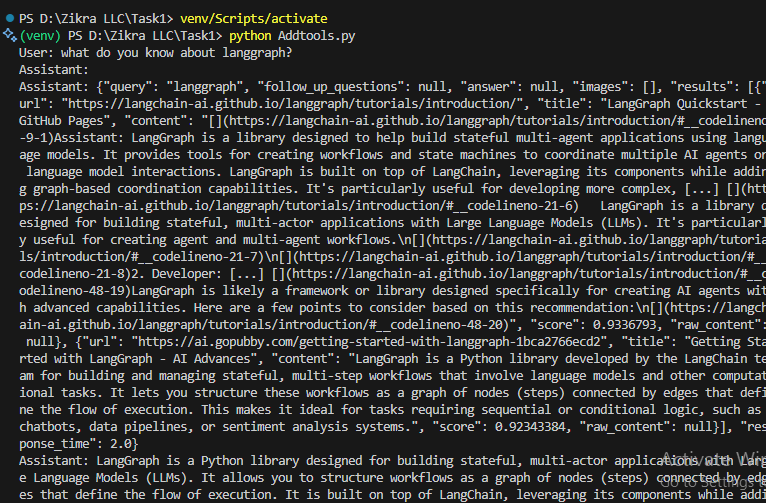
Key things implemented in this section:

* Handle unknown queries by invoking a search tool (Tavily).
* Route messages conditionally:

If the LLM wants to use a tool, it goes to the tools node.

Otherwise, it returns a direct answer.

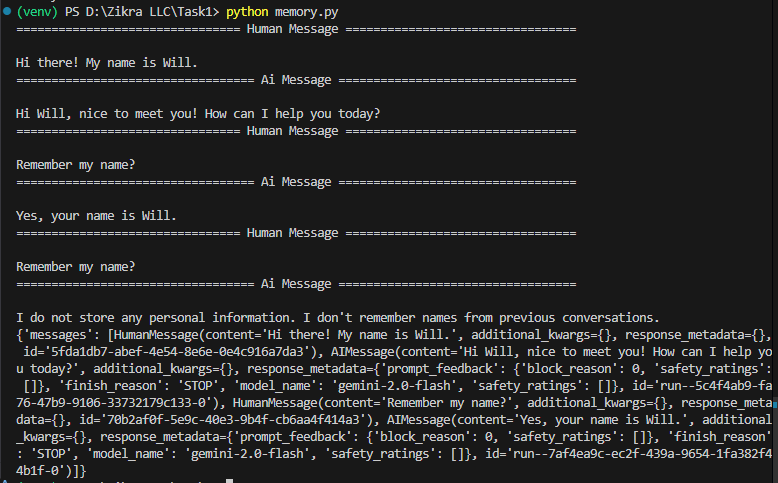
* Return search results and continue the conversation with updated context.



1. **Memory:**

Key objectives achieved in this section:

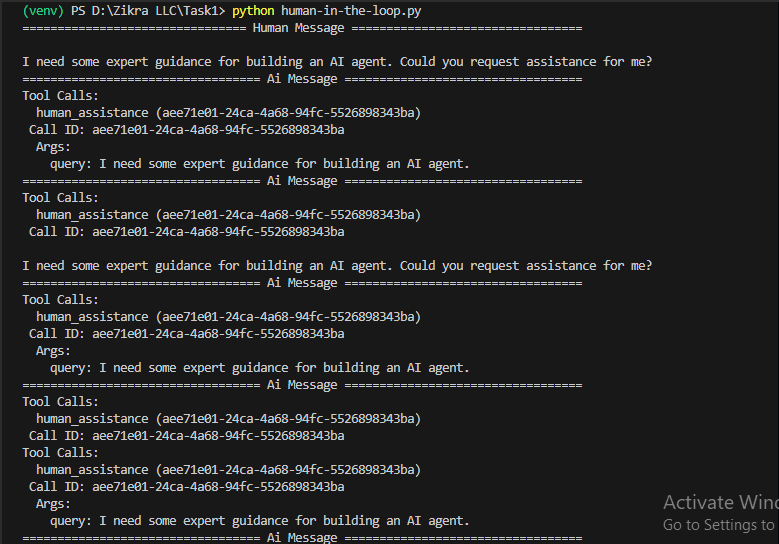
* Enabled memory in LangGraph using a checkpointer (MemorySaver).
* Maintained conversation history across multiple user inputs using thread\_id.
* Configured and compiled the graph with persistent state tracking.
* Demonstrated multi-turn interactions with memory recall.
* Inspected saved state using graph.get\_state() to verify memory.
* Handled independent sessions via different thread\_ids for parallel chats.

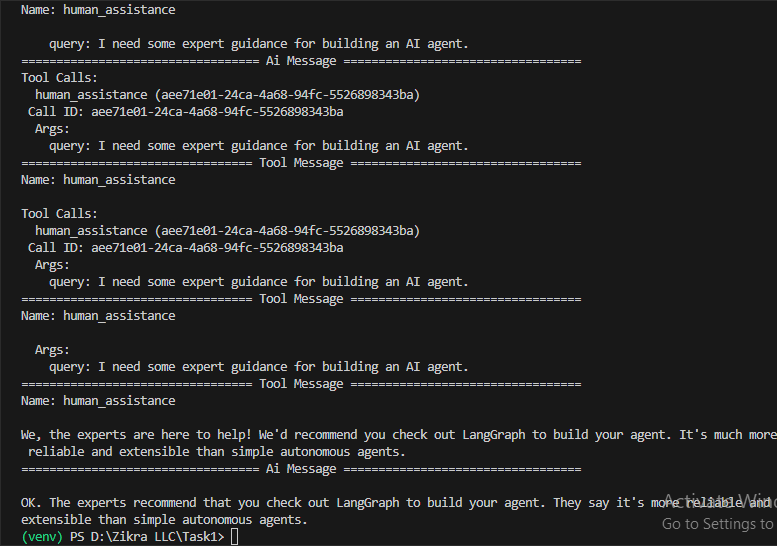


1. **Human-in-the loop controls:**

Objectives achieved in this section:

* **Integrated Human-in-the-Loop Control:** Enabled pausing of agent execution to await human input using the interrupt() function.
* **Added Human Assistance Tool:** Created a human\_assistance tool that simulates requesting help from a human during agent execution.
* **Tool Execution Handling:** Modified the chatbot flow to allow tool execution (with ToolNode) and conditionally route based on tool calls.
* **Stateful Execution with Checkpointing:** Used MemorySaver for persisting and resuming execution state during the pause.
* **Resumed Execution with Command:** Demonstrated how to resume the interrupted flow using Command(resume={"data": ...}) to inject human responses back into the flow.

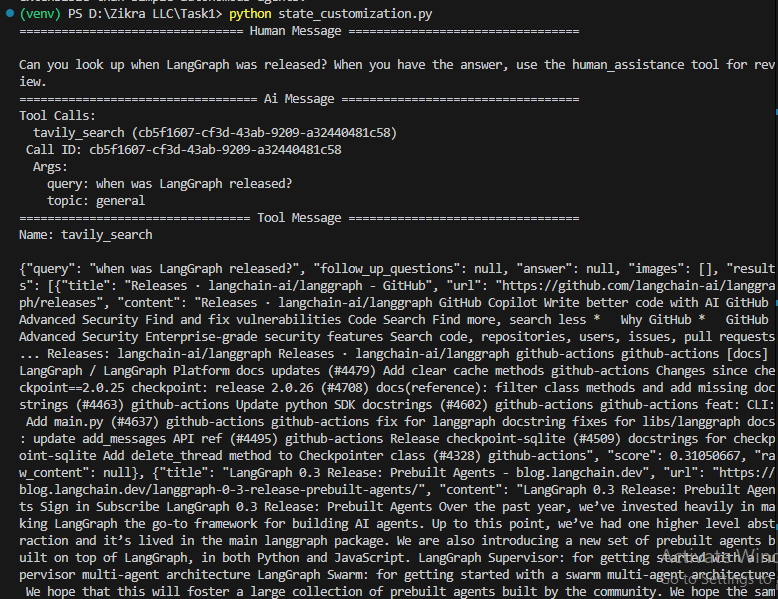


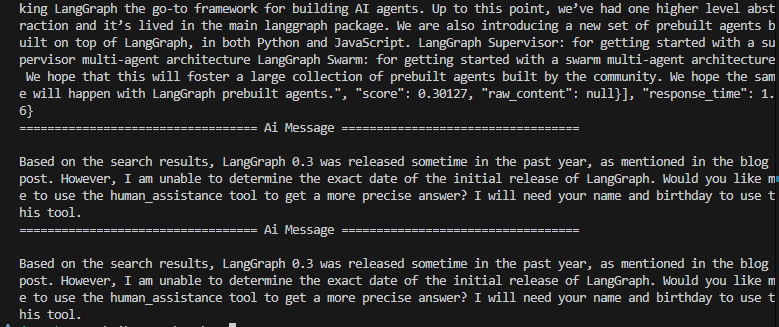


1. **State customization:**

Objectives achieved:

* Uses tools (like search or human feedback).
* Maintains state across turns.
* Interacts conditionally based on tool use.
* Supports interruptions and resumptions (important for human-in-the-loop review).
* Demonstrates how to store state with a memory backend.

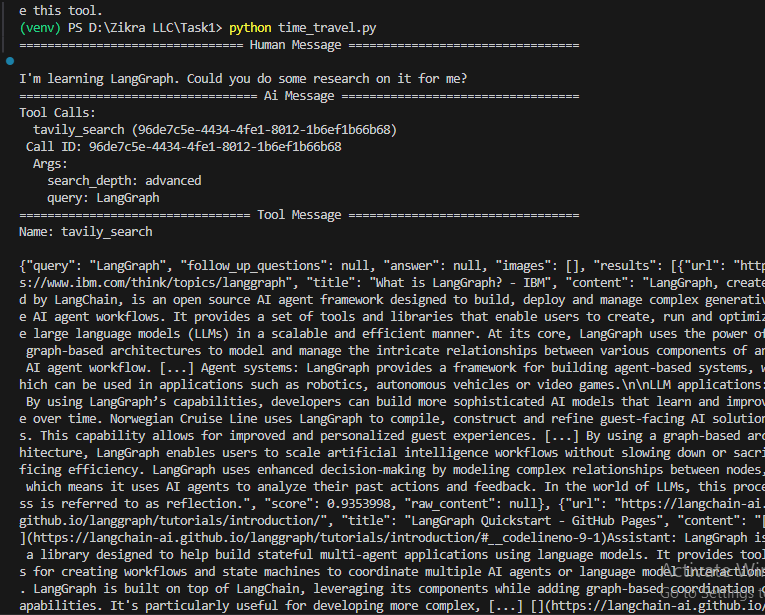


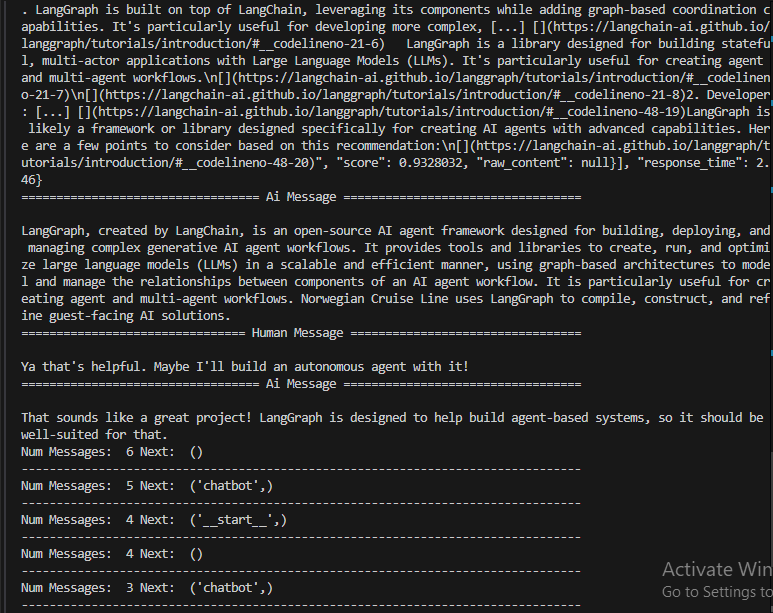


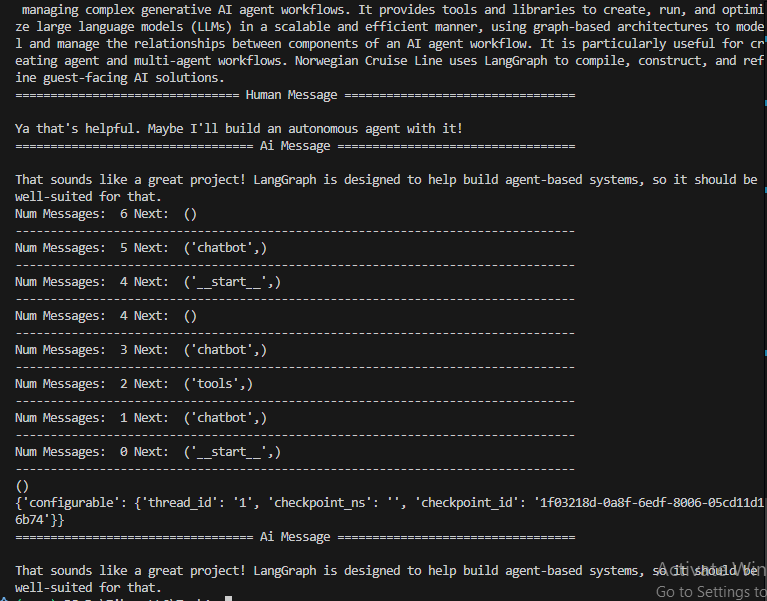
1. **Time Travel:**

Objectives achieved:

* State Persistence using MemorySaver checkpointer.
* Time Travel (State Replay) from saved checkpoints.
* State Inspection via get\_state\_history().
* Dynamic Tool Use triggered by LLM reasoning.
* Conditional Graph Flow using tools\_condition.
* Streaming Execution for real-time feedback and debugging.







**Summary:**

In order to summarize, we first initialized the Gemini-2.0 language model and set up the necessary environment and type definitions to manage conversation state. We integrated the Tavily search tool with the model to enable the agent to perform live research and enhance its responses with external information. Using LangGraph, we constructed a state graph comprising chatbot and tool nodes, connected by conditional edges to manage interaction flow based on user inputs. We implemented persistent memory storage with the MemorySaver checkpointing system, allowing the conversation state to be saved and retrieved later for continuity. The tutorial also demonstrated streaming responses for multiple user inputs, enabling real-time interactive conversations. Finally, we showcased time travel functionality by accessing historical conversation states and replaying them, which supports debugging, experimentation, and seamless session resumption from any checkpointed point.

**Note:**

LangChain = Our AI's brain. It handles understanding what people say, thinking about it, and coming up with answers or actions. It’s where the AI processes language and figures out what to do next.  
LangGraph = Our AI's *nervous system* that controls memory, tools, logic, and teamwork between agents. It is like the nervous system that connects everything together. It controls how the AI remembers past conversations (memory), uses different tools (like searching the internet), follows rules and logic to decide what to do, and helps multiple AI parts (agents) work as a team smoothly. It manages the flow and coordination so the AI behaves smartly and consistently.