**AI Agent Using Langchain**

**Key Agentic Workflow:**

1)Planning- It is the process of thinking through the steps to take the outline and what to do after that

2)The tool use like the search tool

3)Reflection- iteratively improve a result ,possibly with multiple LLM.

4)Multiagent communication

5)Memory -Tracking the progress .

**AI Agent using scratch**

It is based on ReACT Pattern ( Reasoning +Acting on LLM)

LLM first thinks what to do and then decides what action to take

Action is performed on the Environment.The Environment receives certain observation and then passes it to the LLM ,the whole process is repeated

**Process of Building AI Agent :**

1. Importing Libraries
2. Initializing the language model - Open AI
3. Create Agent:

* Create Class
* Parameterized system message and let user pass it and save it as an attribute
* Keep track of message
* In call method ,Message comes in form of string and append it to existing messages. Then Execute ( ) takes result and add to it .

**Execute( )**

* Call the openAI and initializes it . We keep the temperature as 0 in the attribute as it helps in giving a deterministic values

**Workflow of the ReACT Agent: Created Using LLM API and raw Python Code**

User

↓

Prompt 11111111\*r60\*t\*r60\*t

↓

LLM 11111111\*r60\*t\*r60\*t

↓

Thought

↓

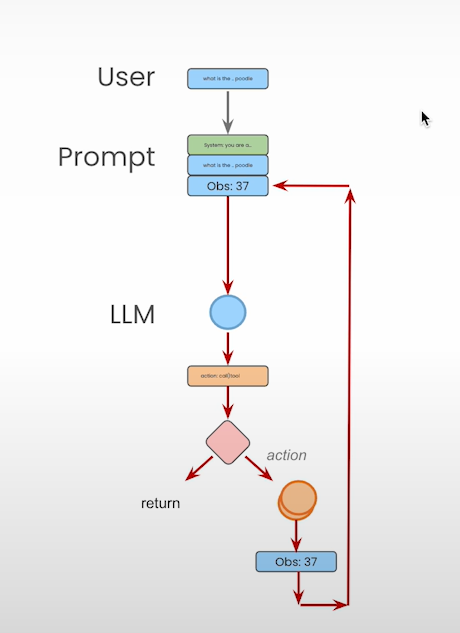
Action

↓

**Return Tool**

**Code :** [**ai-agents-/AI Agents using Langchain/Lesson\_1\_Student.ipynb at main · ibilji/ai-agents-**](https://github.com/ibilji/ai-agents-/blob/main/AI%20Agents%20using%20Langchain/Lesson_1_Student.ipynb)

**Agent using Lang Graph**



Langchain is used for single and multiagent flows.It provides controlled flow. We are building using raw LLM and raw python with tools and turn into real agent with complex questions.

API → Tavily Search API

**Features :**

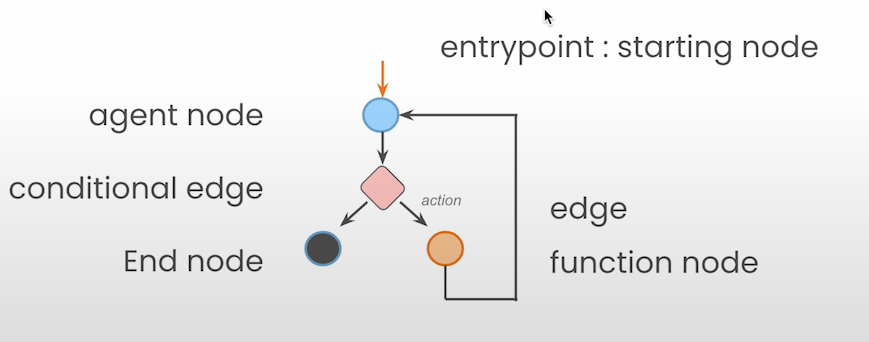
1. The Prompt is Reusable. We can have prompt templates
2. Tools→ Tavility Tool for search
3. Allows to create cyclic graph
4. Built in persistence which means we can have multiple conversations and have previous iteration fetched.
5. Human in loop features

**Components of the LangGraph**

OO - Nodes: Agents or Functions

→ Edges: Connect nodes

Conditional Edges: Decisions



**Agent State:** It is Accessible to all parts of the graph. Local to graph. Stored in persistence Layer meaning we can resume to that state at any point of time

* **Pseudo Code of the Simple State :**

Class Agent state (Typed Dict):

Messages: Annotated (Sequence (Base Message), Operator.add)

Here ,Base Message- is Lang Chain Type

Annotated Operator.add -doesn’t override but appends the new messages

* **In Complex State we have three functions**

Call OpenAI

Agent: 3 functions Check Action Present

Take Action

**LLM**: Call\_openai

**C\_edge**: exists\_action

**Action**: take\_action

**Code Information:**

* System is the attribute we take .
* The tool we use is the tifaly search tool
* We create a graph by Initializing using state graph
* **Graph.addnode→ Adds nodes (“llm”,llm\_node )**

(llm node ) (function we want to call)

* **graph.addconditional\_edges(Parameter1,Parameter2,Parameter3)**

1. node (llm)

2. function (After that)

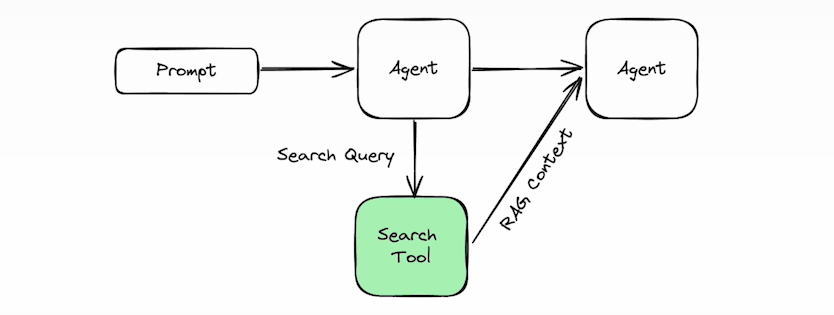
3. Dictionary that the response of function to the next node,if true it calls the action node if false it goes to end of the graph

* **the def call\_openai():** takes all the nodes and edges .Gets the messages from the states .Adds them in the system message .It calls the model and returns the message in the form of dictionary
* **the def take\_action():** uses tools to call on the last message .Putting it in a loop because we have it in the form of lists .Then we add it to the message
* **def exists\_action():**This is our conditional edge part . It takes the last message returned from state and returns length.tool . Retruns true if tool calls it .

Code :  [ai-agents-/AI Agents using Langchain/Lesson\_2\_Student.ipynb at main · ibilji/ai-agents-](https://github.com/ibilji/ai-agents-/blob/main/AI%20Agents%20using%20Langchain/Lesson_2_Student.ipynb)

**Agentic Search Tools**

* Agent wants a structured data



Code: [ai-agents-/AI Agents using Langchain/Lesson\_3\_Student.ipynb at main · ibilji/ai-agents-](https://github.com/ibilji/ai-agents-/blob/main/AI%20Agents%20using%20Langchain/Lesson_3_Student.ipynb)

**Persistence and Streaming**

Two important concepts when running on long running tasks:

1. Persistence: Lets you keep the state of an agent at a particular point in time. It lets you go back to the state and resume in that state in future interactions.
2. Streaming: You can emit list of signals of what is going on at that time and moment. So, in long running, we get to know what exactly they are doing.

**Steps to create Agent:**

1. Load the appropriate environment variable
2. Import libraries
3. Create Tavily Search Tool
4. Create Agent State
5. Create Agent
6. Now check persistence ,for persistence we add a check pointer in Lang Graph

* Check Pointer → checks the state, after and between each nodes
* Using Sqlite for in-memory database

1. Now we check the steaming .Two things to care of in streaming:

* Streaming of individual messages.
* Streaming the tokens. For each token of the LLM call, we want to stream the output.

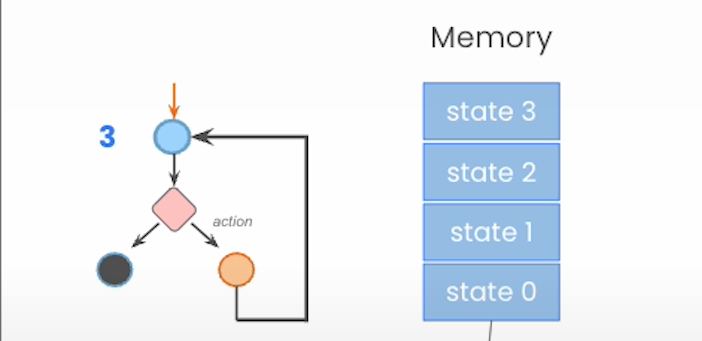
For streaming we use Thread config which keeps tracks of threads in check pointer. For individual, we use thread\_in and stream. For tokens, we use astream and async check pointer

Code: [ai-agents-/AI Agents using Langchain/Lesson\_4\_Student.ipynb at main · ibilji/ai-agents-](https://github.com/ibilji/ai-agents-/blob/main/AI%20Agents%20using%20Langchain/Lesson_4_Student.ipynb)

**Human In the Loop**

* For human in the loop, we don’t use operator.add, we replace the messages for that we add a custom reduced messages function.
* Custom reduced message looks for messages with same id and replace them. If different, it will append it. Then we create the Tavily Search and Agent.

State Memory : When graph is executed, snapshot of each state is stored in memory.

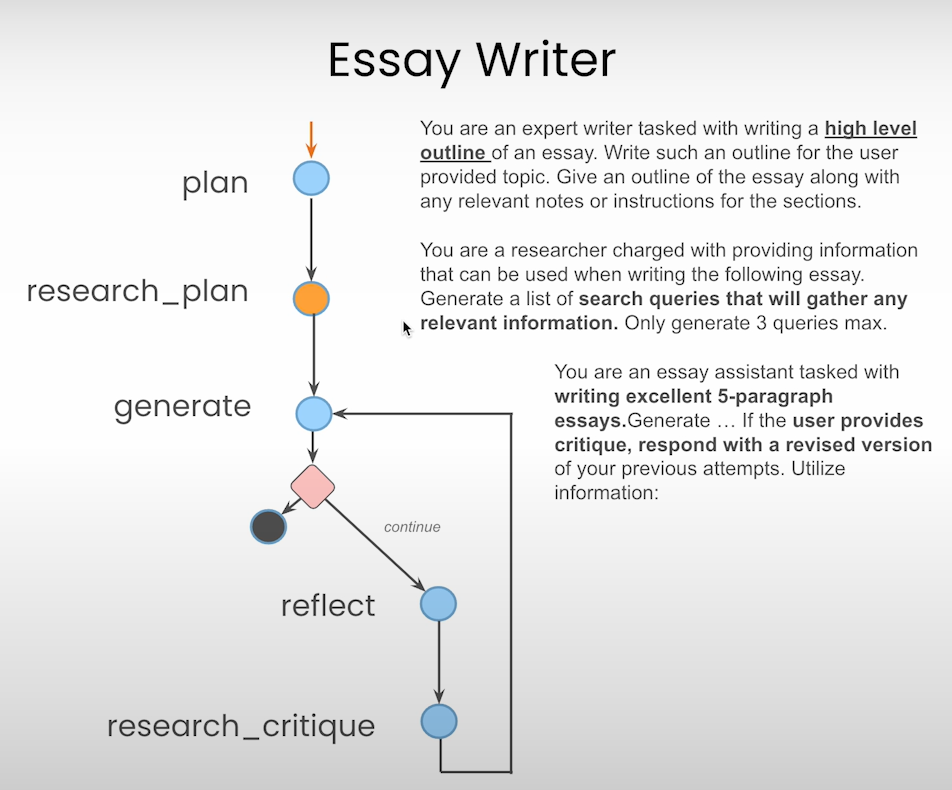


Form: **State Sanpshot: {Agent State, useful things}**

* Here ,useful things can be thread or unique identifier (thread\_ts)
* g.invoke(None, { ….., thread, thread\_ts}
* thread- to access a state 1 as current state .Without thread\_ts and only passing thread ID (thread) uses current state of thread as current point.
* We can get the state and modify it. The modified state is the new stat

Code : [ai-agents-/AI Agents using Langchain/Lesson\_5\_Student.ipynb at main · ibilji/ai-agents-](https://github.com/ibilji/ai-agents-/blob/main/AI%20Agents%20using%20Langchain/Lesson_5_Student.ipynb)

**Creating an Essay Writer**

****

Code: [ai-agents-/AI Agents using Langchain/Lesson\_6\_Student (1).ipynb at main · ibilji/ai-agents-](https://github.com/ibilji/ai-agents-/blob/main/AI%20Agents%20using%20Langchain/Lesson_6_Student%20(1).ipynb)