Getting Started with Agents Using LangChain

By Lucas Soares

04/14/2025

Lucas Soares

- AI Engineer

Lucas Soares

- AI Engineer
- Instructor at O'Reilly Media

Lucas Soares

- AI Engineer
- Instructor at O'Reilly Media
- Curious about all things intelligence

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Thought + Action

Thought + Action

- How do we do stuff? We **think** and we **act**

Thought + Action

- How do we do stuff? We **think** and we **act**
- Example: Decision-making process for attending a live-training

Thought + Action

- How do we do stuff? We **think** and we **act**
- Example: Decision-making process for attending a live-training

Thinking:

What to do + planning (order, priority..)

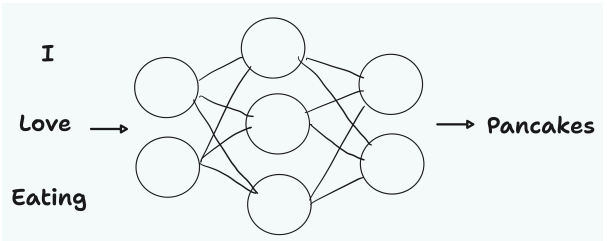
Acting:

used **tools: search, browser, etc...**

What is an Agent? (LLM + Tool)

LLM

Predicts next word/sentence



Tool

Performs actions in the real-world

LLMs can use tools!

- Toolformer

LLMs can use tools!

- Toolformer

> The New England Journal of Medicine is a registered trademark of **[QA("Who is the publisher of The New England Journal of Medicine?") → Massachusetts Medical Society]** the MMS.

> Out of 1400 participants, 400 (or **[Calculator(400 / 1400) → 0.29]** 29%) passed the test.

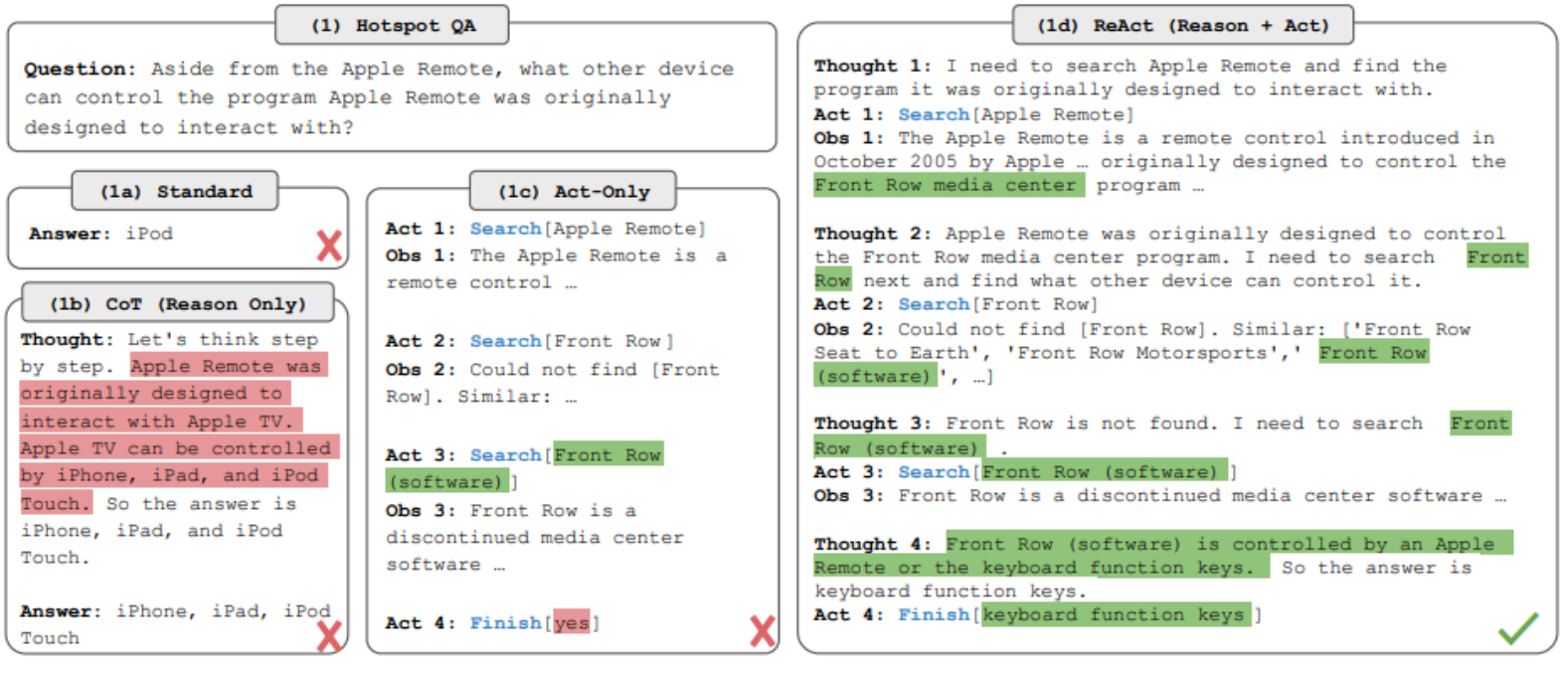
> The name derives from "la tortuga", the Spanish word for **[MT("tortuga") → turtle]** turtle.

> The Brown Act is California's law **[WikiSearch("Brown Act") → The Ralph M. Brown Act is an act of the California State Legislature that guarantees the public's right to attend and participate in meetings of local legislative bodies.]** that requires legislative bodies, like city councils, to hold their meetings open to the public.

[1] (Schick u. a., o. J., 2023)

Interleaving Thoughts and Actions

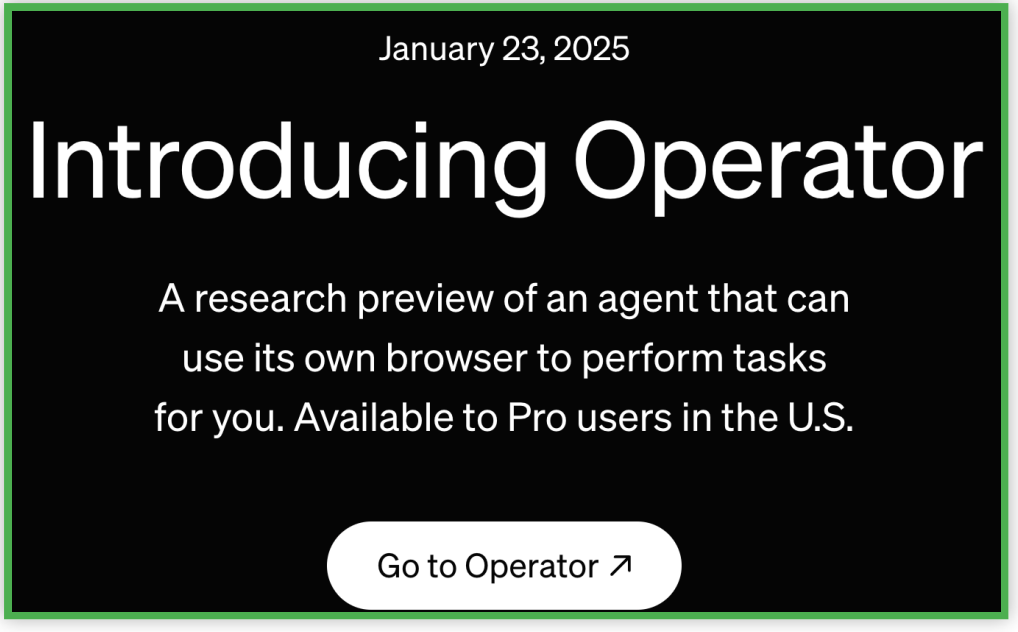
- ReACT: LLMs for **RE**asoning & **ACT**ion.



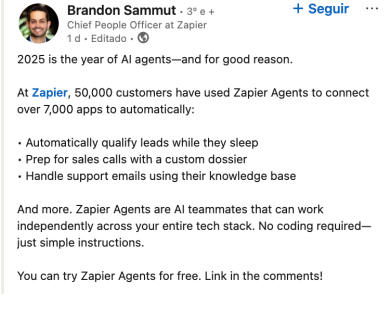
[2] Yao, X., et al. (2023)

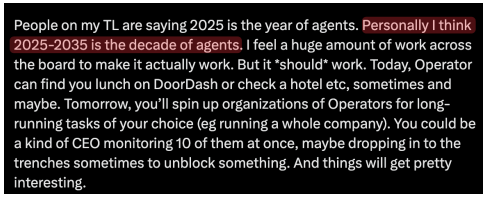
2025 Is the Year of Agents

2025 Is the Year of Agents



DAVOS, Switzerland – We are "just on the verge" of AI agents, OpenAI chief product officer Kevin Weil told Axios' Ina Fried on Jan.21.





Sources: Axios, Barrons, Karpathy, OpenAI

Agents in 3 Levels of Complexity

Level 1: LLM + functions inside the prompt

Level 1: LLM + functions inside the prompt

- Inspired by 'Toolformer'

The New England Journal of Medicine is a registered trademark of [QA("Who is the publisher of The New England Journal of Medicine?") → Massachusetts Medical Society] the MMS.

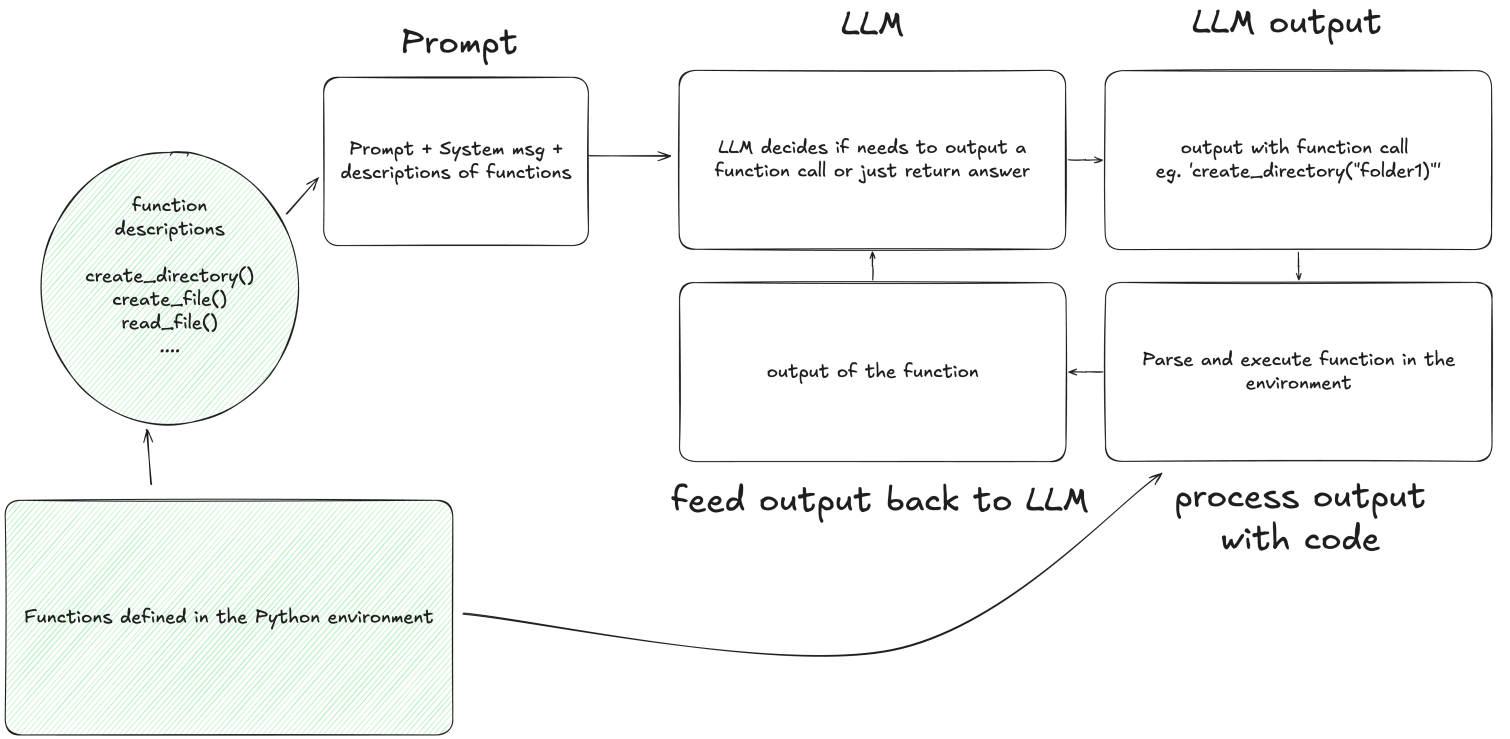
Out of 1400 participants, 400 (or [Calculator(400 / 1400) → 0.29] 29%) passed the test.

The name derives from "la tortuga", the Spanish word for [MT("tortuga") → turtle] turtle.

The Brown Act is California's law [WikiSearch("Brown Act") → The Ralph M. Brown Act is an act of the California State Legislature that guarantees the public's right to attend and participate in meetings of local legislative bodies.] that requires legislative bodies, like city councils, to hold their meetings open to the public.

Schick et al. 2023

Level 1: LLM + functions inside the prompt



Example from Toolformer Paper

```
Wikipedia Search We use the following prompt
for the Wikipedia search tool:
Your task is to complete a given piece
of text. You can use a Wikipedia Search
API to look up information. You can do
so by writing "[WikiSearch(term)]" where
"term" is the search term you want to
look up. Here are some examples of API
calls:
```

Example from Toolformer Paper

```
Wikipedia Search We use the following prompt
for the Wikipedia search tool:
Your task is to complete a given piece
of text. You can use a Wikipedia Search
API to look up information. You can do
so by writing "[WikiSearch(term)]" where
"term" is the search term you want to
look up. Here are some examples of API
calls:
Input: The colors on the flag of Ghana
have the following meanings: red is for
the blood of martyrs, green for forests,
and gold for mineral wealth.
Output: The colors on the flag of Ghana
have the following meanings: red is for
[WikiSearch("Ghana flag red meaning")]
the blood of martyrs, green for forests,
and gold for mineral wealth.
....(more input output examples)...
Input: x
Output:
```

Schick et al. 2023

Limitations

Limitations

- **Probabilistic outputs** make function calls unreliable

Limitations

- **Probabilistic outputs** make function calls unreliable
- Need for **structured ways to prepare the inputs** of the function calls

Limitations

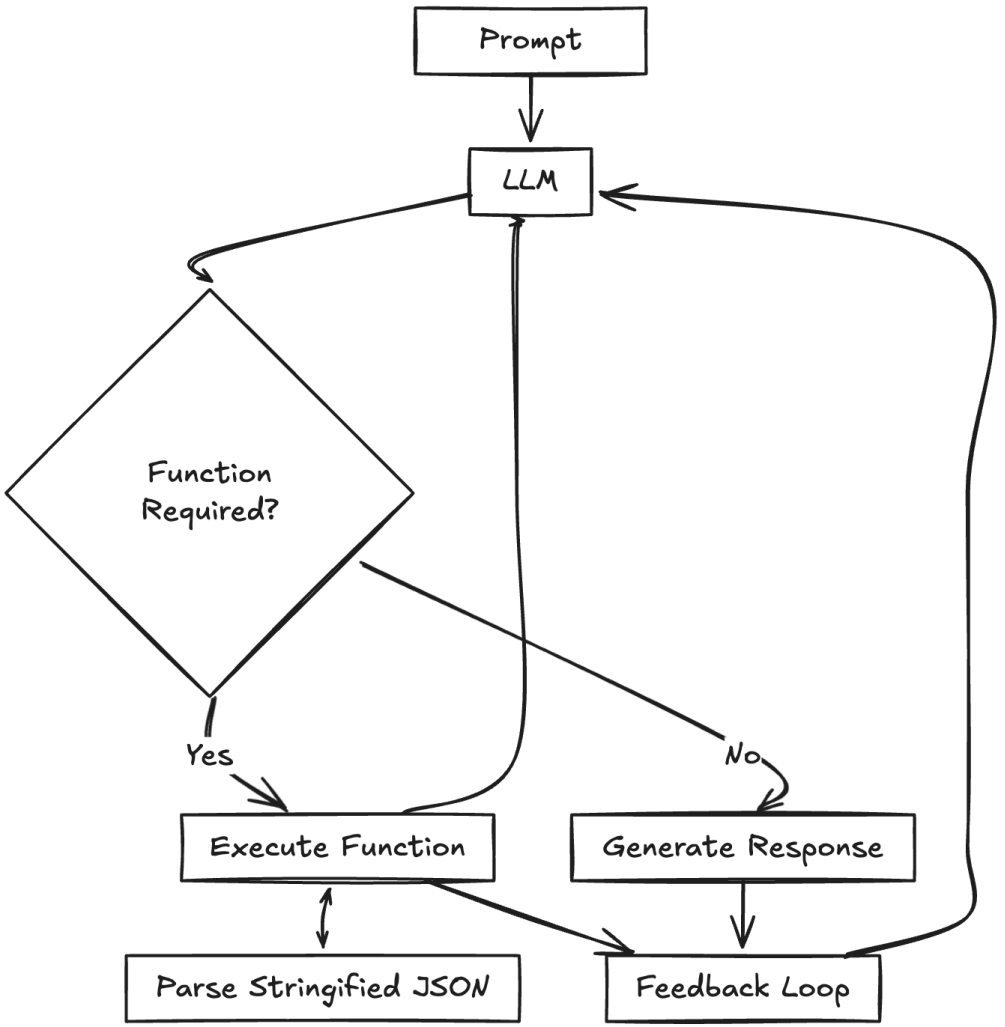
- **Probabilistic outputs** make function calls unreliable
- Need for **structured ways to prepare the inputs** of the function calls
- Putting entire functions inside text prompts is clunky and **non-scalable**

Limitations

- **Probabilistic outputs** make function calls unreliable
- Need for **structured ways to prepare the inputs** of the function calls
- Putting entire functions inside text prompts is clunky and **non-scalable**
- Solution? Function Calling!

Level 2: OpenAI Function Calling

Level 2: OpenAI Function Calling



Level 2: OpenAI Function Calling

1. Call model with functions defined – along with your system and user messages.

Level 2: OpenAI Function Calling

1. Call model with functions defined – along with your system and user messages.
2. Model decides to call function(s) – model returns the name and input arguments.

Level 2: OpenAI Function Calling

1. Call model with functions defined – along with your system and user messages.
2. Model decides to call function(s) – model returns the name and input arguments.
3. Execute function code – parse the model's response and handle function calls.

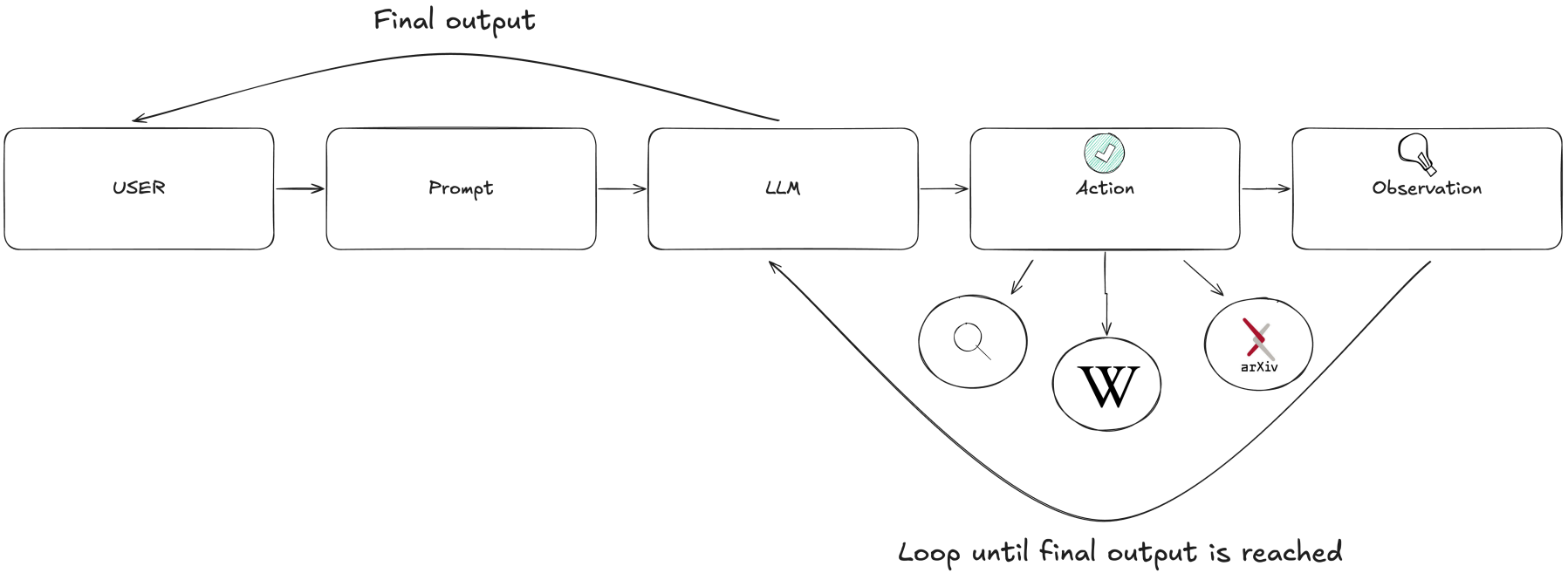
Level 2: OpenAI Function Calling

1. Call model with functions defined – along with your system and user messages.
2. Model decides to call function(s) – model returns the name and input arguments.
3. Execute function code – parse the model's response and handle function calls.
4. Supply model with results – so it can incorporate them into its final response.

OpenAI Function Calling Docs

Level 3: Autonomous Agents

Level 3: Autonomous Agents



Inspired by this article: 'OpenAI's Bet on a Cognitive Architecture'

Notebook Demo: Building a Simple React Agent with LangChain/LangGraph

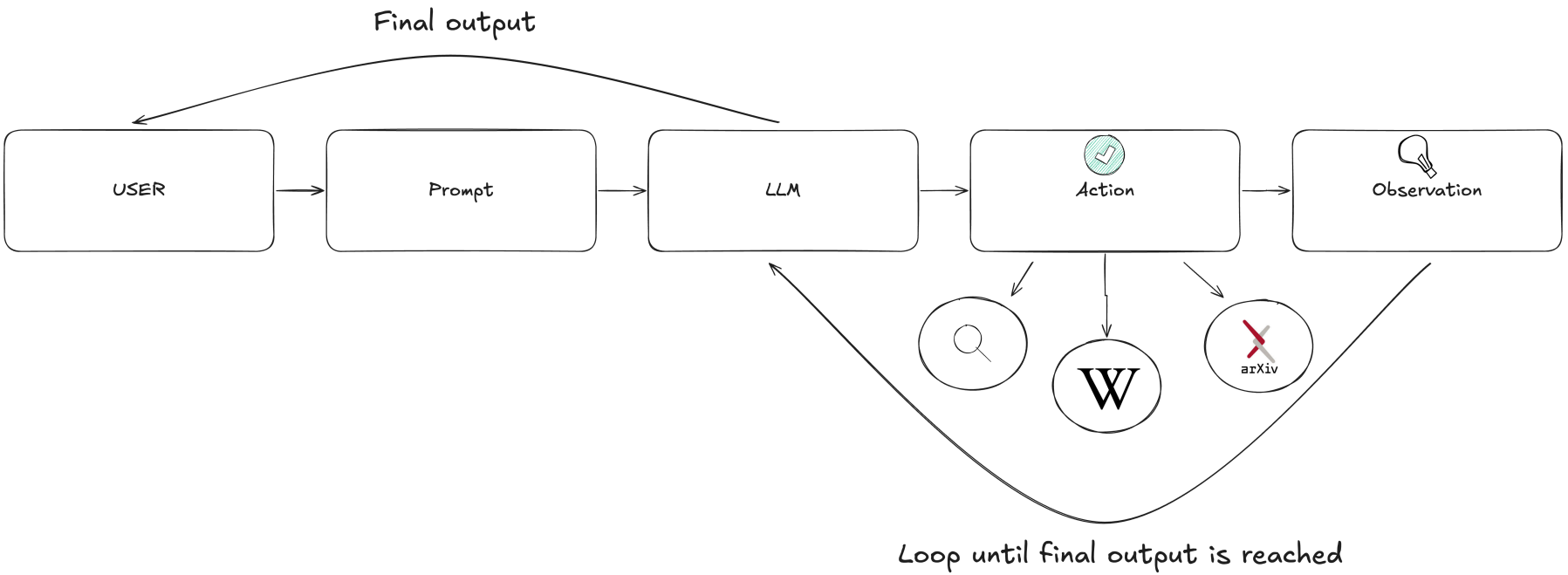
Q&A

Break

The Agent Loop

How Can We Effectively Perform Tasks with Agents?

How Can We Effectively Perform Tasks with Agents?



Inspired by this article: 'OpenAI's Bet on a Cognitive Architecture'

Good Agents are Routers

Good Agents are Routers

- Good examples of 'useful' agents (that implement a more like routing type of architecture than actual agent architecture)

Good Agents are Routers

- Good examples of 'useful' agents (that implement a more like routing type of architecture than actual agent architecture)
- LangChain is a framework to implement these types of routing procedures!

Inspired by this article: 'OpenAI's Bet on a Cognitive Architecture'

What is LangChain?

What is LangChain?

- LangChain is a framework for building context-aware reasoning applications

What is LangChain?

- LangChain is a framework for building context-aware reasoning applications
- Its main features

What is LangChain?

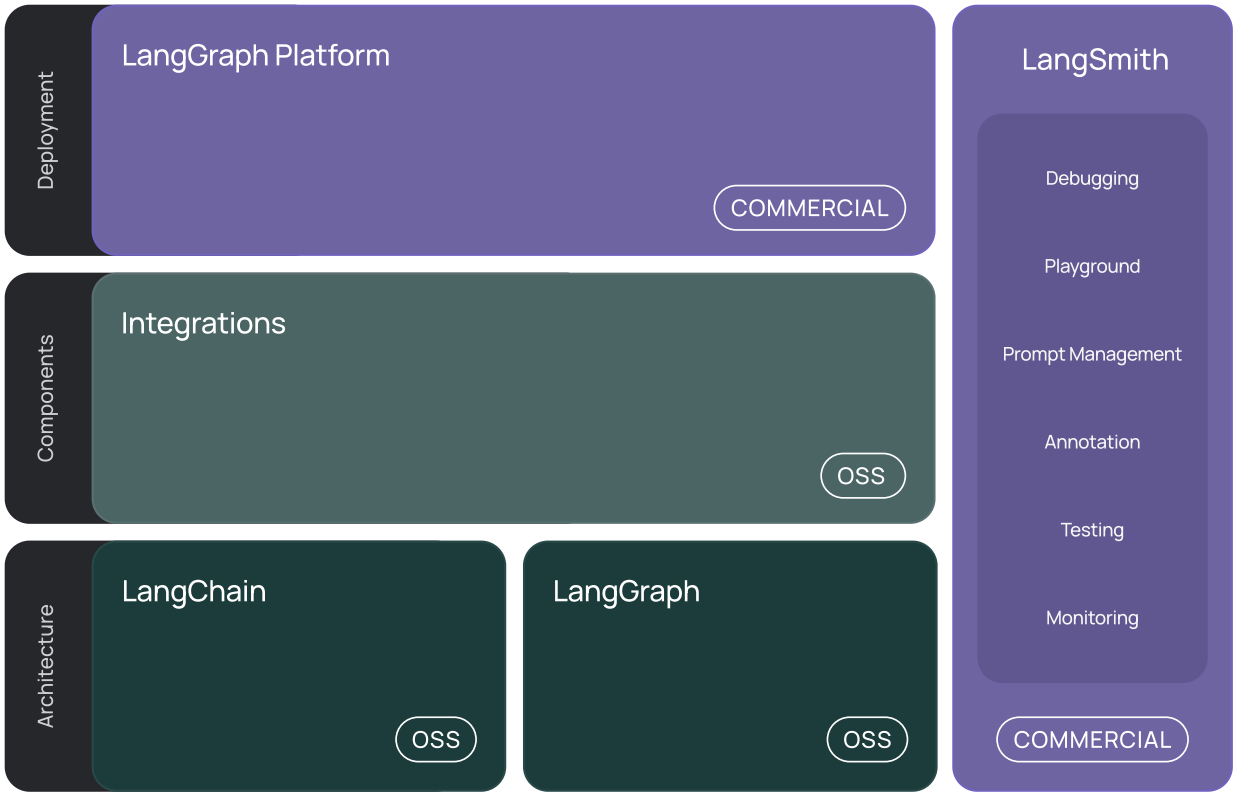
- LangChain is a framework for building context-aware reasoning applications
- Its main features
  - **Components**: composable tools and integrations for working with language models.

What is LangChain?

- LangChain is a framework for building context-aware reasoning applications
- Its main features
  - **Components**: composable tools and integrations for working with language models.
  - **Off-the-shelf chains**: built-in assemblages of components for accomplishing higher-level tasks

LangChain Docs

The LangChain Stack



LangChain Docs

Core Elements of LangChain

Core Elements of LangChain

Models

Core Elements of LangChain

Models

- Abstractions over LLM APIs (e.g ChatGPT API)

Core Elements of LangChain

Models

- Abstractions over LLM APIs (e.g ChatGPT API)

```
from langchain_openai.chat_models import ChatOpenAI

chat_model = ChatOpenAI(api_key=os.getenv("OPENAI_API_KEY"), model="gpt-4o")

chat_model.invoke("hi!")
```

Core Elements of LangChain

Models

- Abstractions over LLM APIs (e.g ChatGPT API)

```
from langchain_openai.chat_models import ChatOpenAI

chat_model = ChatOpenAI(api_key=os.getenv("OPENAI_API_KEY"), model="gpt-4o")

chat_model.invoke("hi!")
```

OR

```
# More flexible way to initialize the model (with any provider)
from langchain.chat_models import init_chat_model

model = init_chat_model("gpt-4o-mini", model_provider="openai")
```

Core Elements of LangChain

Prompt Templates

Core Elements of LangChain

Prompt Templates

- Abstractions over traditional text prompts for LLMs

Core Elements of LangChain

Prompt Templates

- Abstractions over traditional text prompts for LLMs

```
from langchain.prompts import ChatPromptTemplate
prompt = ChatPromptTemplate.from_template(template="Name 5 ideas for this product:
prompt.format(product="dog")
# Output
# 'Name 5 ideas for this product: dog'
```

Core Elements of LangChain

Output parser

Core Elements of LangChain

Output parser

- Abstractions for parsing outputs of LLMs

Core Elements of LangChain

Output parser

- Abstractions for parsing outputs of LLMs

```
model = ChatOpenAI(temperature=0)
# Define your desired data structure.
class Joke(BaseModel):
    setup: str = Field(description="question to set up a joke")
    punchline: str = Field(description="answer to resolve the joke")

    # Set up a parser + inject instructions into the prompt template.
parser = JsonOutputParser(pydantic_object=Joke)

prompt = PromptTemplate(
    template="Answer the user query.\n{format_instructions}\n{query}\n",
    input_variables=["query"],
    partial_variables={"format_instructions": parser.get_format_instructions()},
)
# LCEL chain interface
chain = prompt | model | parser

chain.invoke({"query": "Tell me a joke."})
# Output
# {'setup': 'Why did the chicken cross the road?',
#'punchline': 'To get to the other side!'}
```

LCEL - Putting Components Together

LCEL - Putting Components Together

LCEL interface

- Interface that leverages the **|** pipe symbol to compose LangChain components

LCEL - Putting Components Together

LCEL interface

- Interface that leverages the `|` pipe symbol to compose LangChain components

```
from langchain_openai.chat_models import ChatOpenAI
from langchain.prompts import ChatPromptTemplate
from langchain.schema.output_parser import StrOutputParser


model = ChatOpenAI(temperature=0)
prompt = ChatPromptTemplate.from_template(template="Name 5 ideas for this product:
output_parser = StrOutputParser()

chain = prompt | model | output_parser

chain.invoke({"concept": "probability distribution"})
# Output
# - Discrete probability distribution: This concept...
# - Continuous probability....
# ...
```

Whiteboard Demo: Core Components of LangChain

Notebook Demo: Building a Simple LangChain App

Break

Advantages of LLM Agents

Advantages of LLM Agents

- **Flexibility**: Agents can adapt to various tasks by determining the best action to take.

Advantages of LLM Agents

- **Flexibility**: Agents can adapt to various tasks by determining the best action to take.
- **Specialization**: LLM agents can be specialized with a set of tools to perform niche tasks.

Advantages of LLM Agents

- **Flexibility**: Agents can adapt to various tasks by determining the best action to take.
- **Specialization**: LLM agents can be specialized with a set of tools to perform niche tasks.
- **Multi-Agent Collaboration**: Specialized LLM agents can collaborate to perform complex tasks.

LangGraph: Multi-Agent Workflows

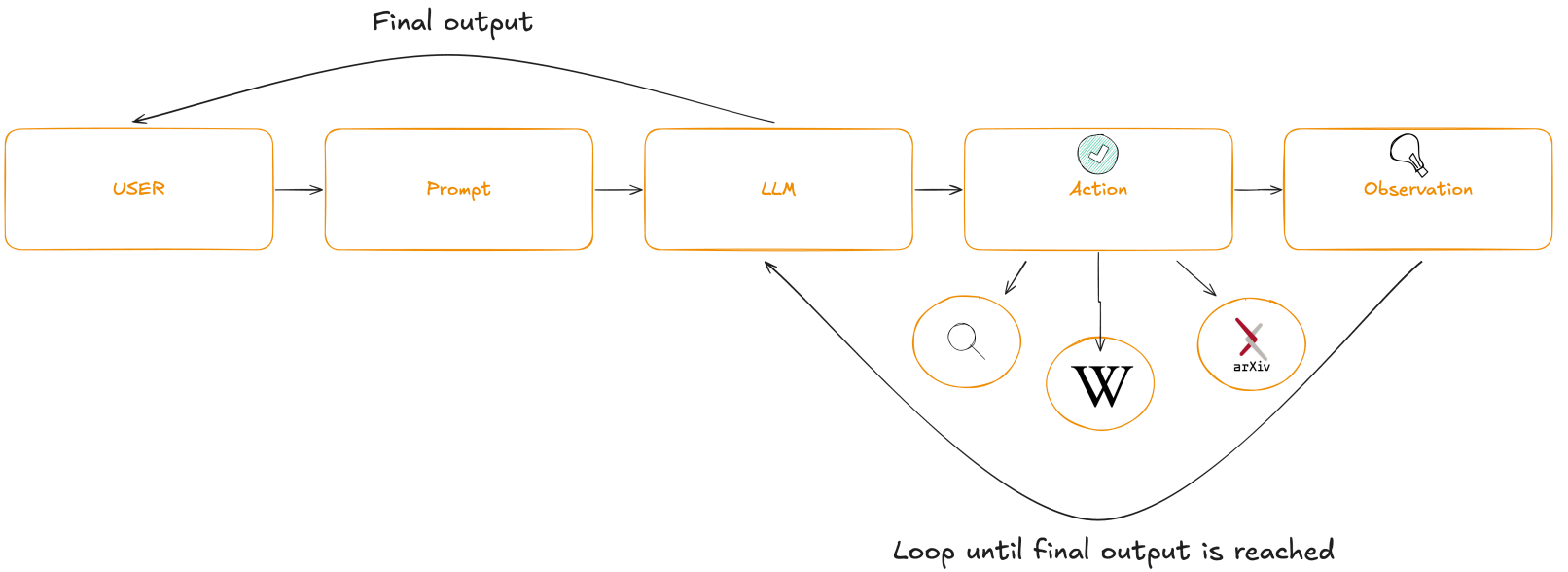
Building Agents with LangChain

Building Agents with LangChain

- LangChain gives us the material for the components

Building Agents with LangChain

- LangChain gives us the material for the components



But that is half the puzzle!

We still need fine-grained control over the connections between components and LCEL is not enough!

Understanding Agentic Systems

- **Agentic Systems**: Systems that utilize large language models (LLMs) to manage the control flow of applications.

Understanding Agentic Systems

- **Agentic Systems**: Systems that utilize large language models (LLMs) to manage the control flow of applications.
- **Key Functions**: Routing decisions, tool selection, and evaluating output sufficiency.

Understanding Agentic Systems

- **Agentic Systems**: Systems that utilize large language models (LLMs) to manage the control flow of applications.
- **Key Functions**: Routing decisions, tool selection, and evaluating output sufficiency.
- **Agent Loop**: Continuous decision-making process that enables agents to solve complex tasks.

Understanding agentic systems in LangGraph.

Key Components of Agentic Systems

Key Components of Agentic Systems

1. **Tool Calling**: Utilizing external tools to perform tasks.

Key Components of Agentic Systems

1. **Tool Calling**: Utilizing external tools to perform tasks.
2. **Action Taking**: Executing actions based on LLM outputs.

Key Components of Agentic Systems

1. **Tool Calling**: Utilizing external tools to perform tasks.
2. **Action Taking**: Executing actions based on LLM outputs.
3. **Memory**: Keeping track of interactions for context-aware responses.

Key Components of Agentic Systems

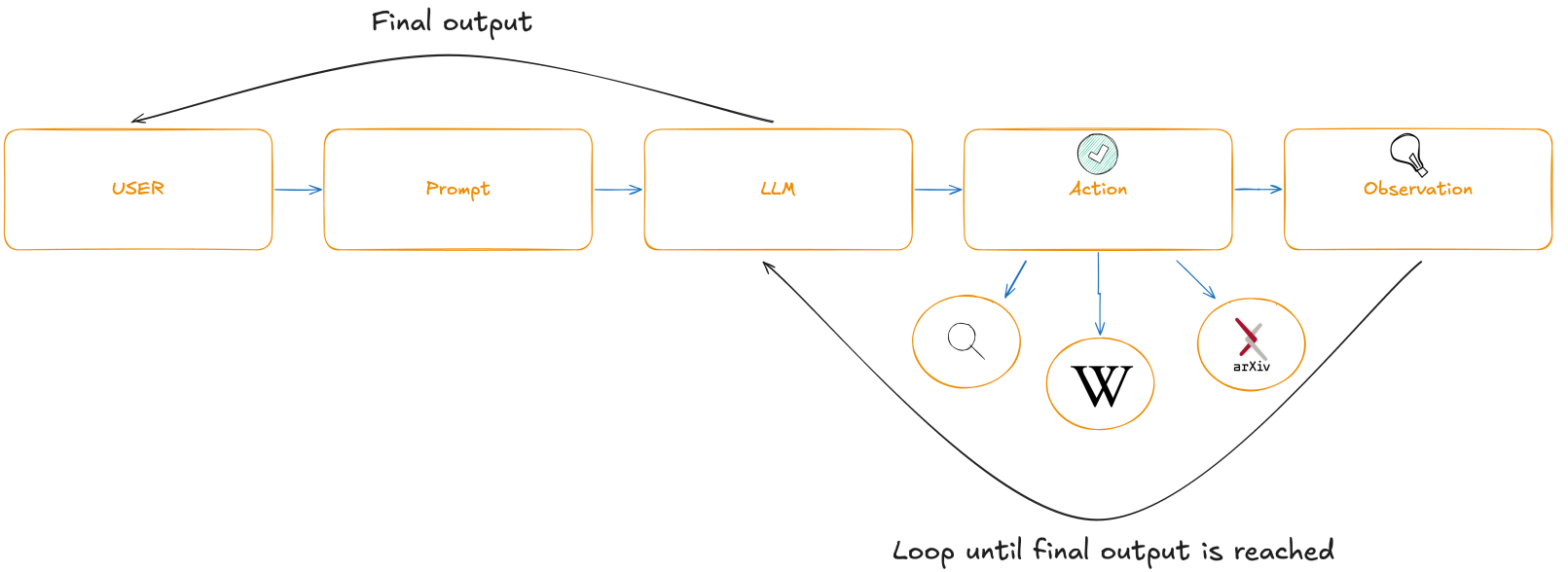
1. **Tool Calling**: Utilizing external tools to perform tasks.
2. **Action Taking**: Executing actions based on LLM outputs.
3. **Memory**: Keeping track of interactions for context-aware responses.
4. **Planning**: Structuring steps to ensure optimal decision-making.

Key components of agentic systems.

Solution = LangGraph

LangGraph = LangChain + Graph Building Capabilities

- LangGraph gives us components from LangChain flexible control for building graphs



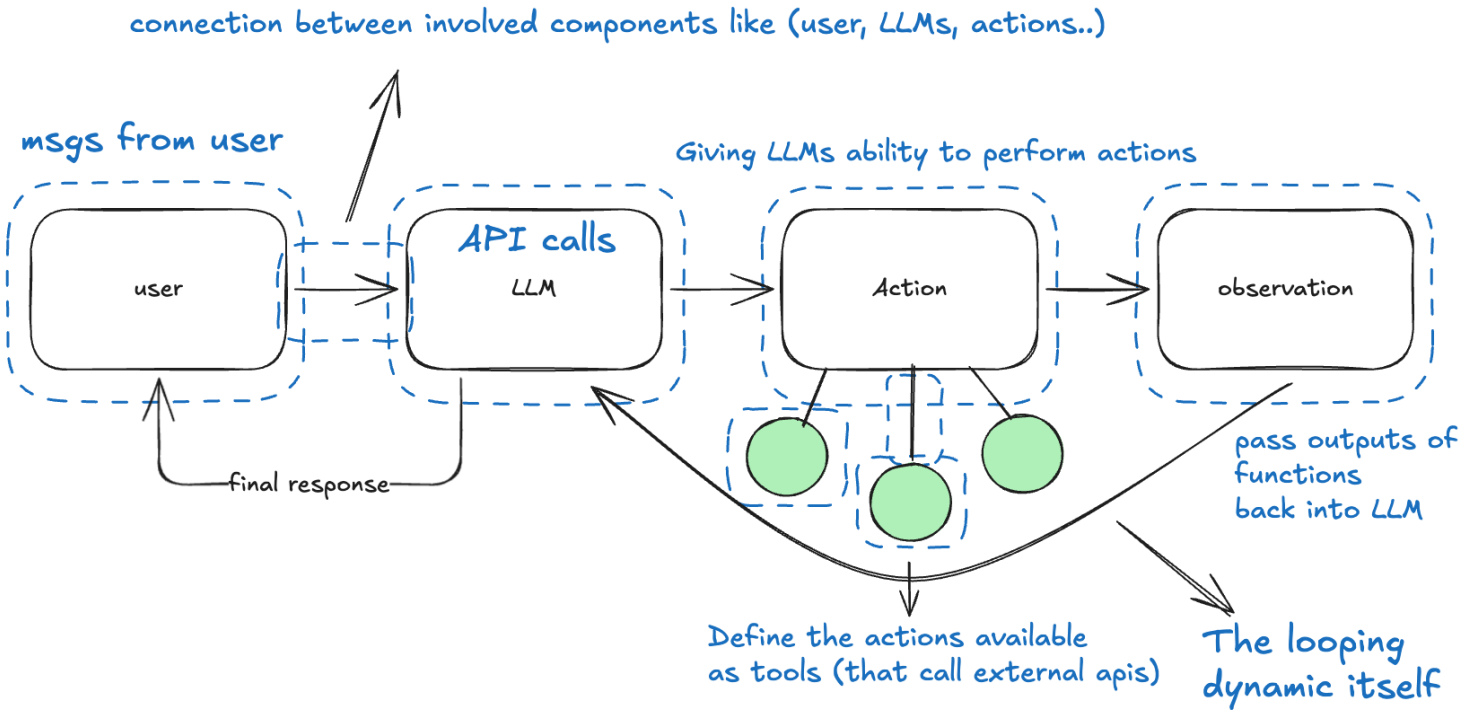
LangGraph = Deterministic + Non-Deterministic Control Flows

- With that we get the ability to build workflows that mix probabilistic outputs (LLMs) + deterministic control flows (Python code)

Agents as Graphs

Agents as Graphs

- Workflows built with agents are usually structured as graphs!!

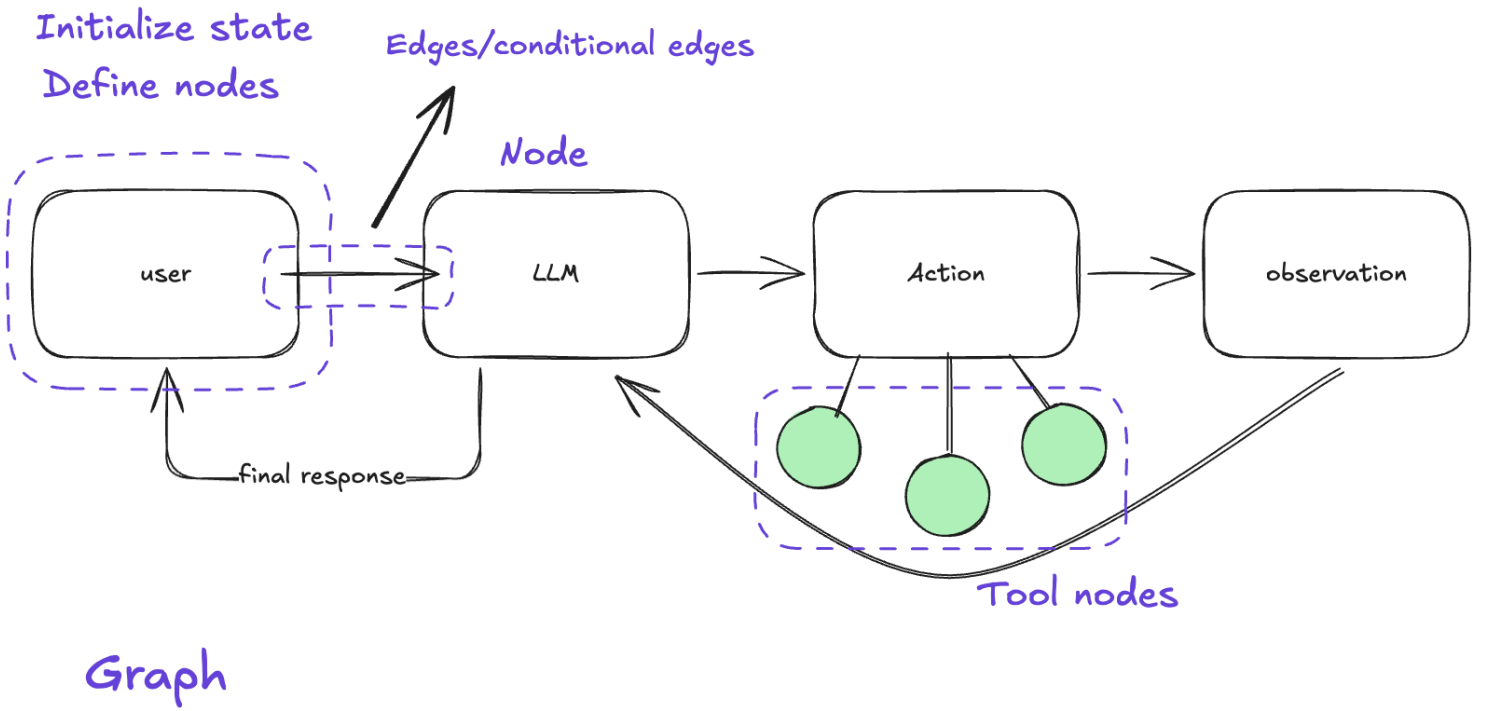


Agent Loop in LangGraph

- Outline of an basic agent loop in langgraph:

Agent Loop in LangGraph

- Outline of an basic agent loop in langgraph:



Why LangGraph?

LangGraph is designed for building agentic applications with some core principles:

Why LangGraph?

LangGraph is designed for building agentic applications with some core principles:

- **Controllability**: Offers low-level control which increases reliability in agentic systems.

Why LangGraph?

LangGraph is designed for building agentic applications with some core principles:

- **Controllability**: Offers low-level control which increases reliability in agentic systems.
- **Human-in-the-Loop**: Built-in persistence layer enhances human-agent interaction patterns.

Why LangGraph?

LangGraph is designed for building agentic applications with some core principles:

- **Controllability**: Offers low-level control which increases reliability in agentic systems.
- **Human-in-the-Loop**: Built-in persistence layer enhances human-agent interaction patterns.
- **Streaming First**: Supports streaming of events and tokens, providing real-time feedback to users.

Overview of LangGraph's purpose and principles.

Notebook Demo: Building a Basic LangGraph Agent

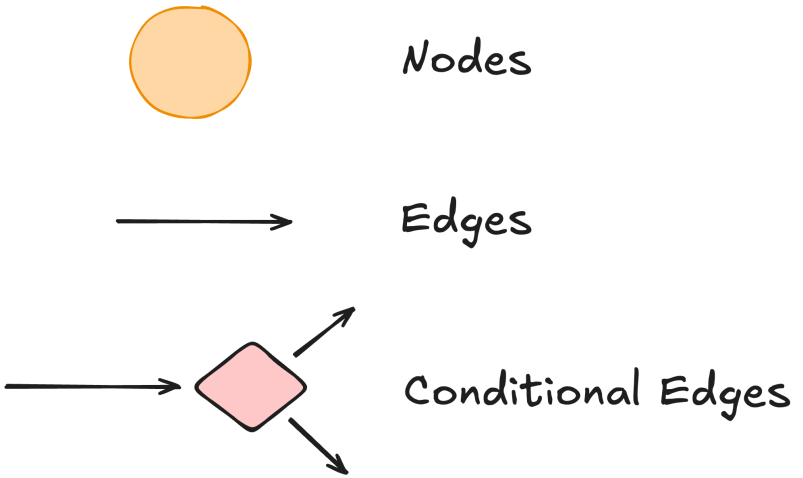
Q&A & Break

The Basic Components of LangGraph

LangGraph models agent workflows as graphs:

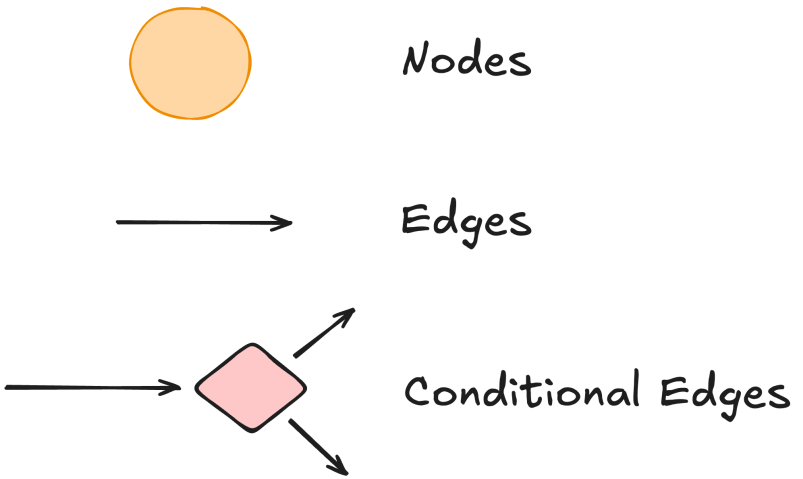
The Basic Components of LangGraph

LangGraph models agent workflows as graphs:



The Basic Components of LangGraph

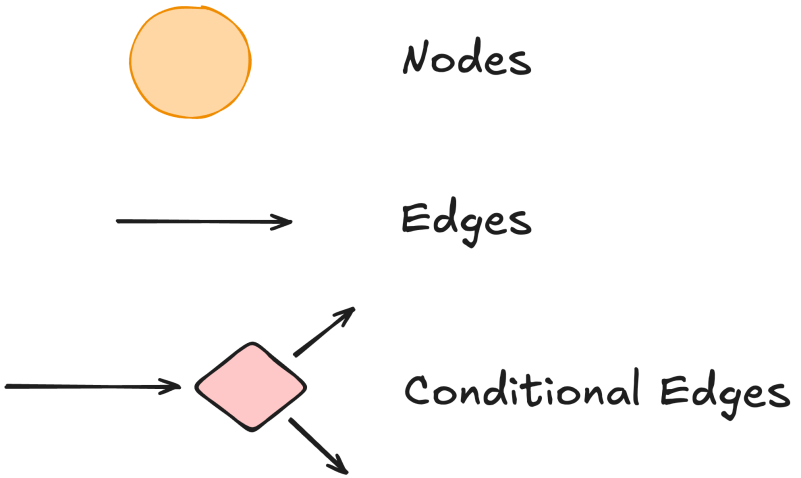
LangGraph models agent workflows as graphs:



- **Nodes**: Python functions that implement the logic of agents, taking the current State as input and returning an updated State.

The Basic Components of LangGraph

LangGraph models agent workflows as graphs:



- **Nodes**: Python functions that implement the logic of agents, taking the current State as input and returning an updated State.
- **Edges/Conditional Edges**: Functions that implement fixed/conditional transitions to determine which Node to execute next based on the current State.

Explanation of LangGraph's components.

States in LangGraph

These graphs in LangGraph are driven by:

States in LangGraph

These graphs in LangGraph are driven by:

- **States**: Shared data structures that evolve over time as Nodes execute and pass messages along Edges.

States in LangGraph

These graphs in LangGraph are driven by:

- **States**: Shared data structures that evolve over time as Nodes execute and pass messages along Edges.
- **Message Passing**: Nodes send messages to activate other Nodes, facilitating the execution of workflows in discrete iterations or "super-steps".

Overview of how graphs and states interact in LangGraph.

Nodes

In LangGraph, Nodes are the core functional units:

Nodes

In LangGraph, Nodes are the core functional units:

- **Functionality**: Each Node is a Python function that processes the current State and outputs an updated State.

Nodes

In LangGraph, Nodes are the core functional units:

- **Functionality**: Each Node is a Python function that processes the current State and outputs an updated State.
- **Execution**: Nodes can run synchronously or asynchronously, and are added to the graph using the `add_node` method.

Nodes

In LangGraph, Nodes are the core functional units:

- **Functionality**: Each Node is a Python function that processes the current State and outputs an updated State.
- **Execution**: Nodes can run synchronously or asynchronously, and are added to the graph using the `add_node` method.
- **Special Nodes**: Includes START and END Nodes to manage the flow of execution in the graph.

Details on the functionality and structure of Nodes in LangGraph.

Edges

Edges define the routing logic in LangGraph:

- **Types of Edges**:

Edges

Edges define the routing logic in LangGraph:

- **Types of Edges**:
    - **Normal Edges**: Direct transitions from one Node to another.

Edges

Edges define the routing logic in LangGraph:

- **Types of Edges**:
  - **Normal Edges**: Direct transitions from one Node to another.
  - **Conditional Edges**: Determine the next Node(s) to execute based on a function's output.

Edges

Edges define the routing logic in LangGraph:

- **Types of Edges**:
  - **Normal Edges**: Direct transitions from one Node to another.
  - **Conditional Edges**: Determine the next Node(s) to execute based on a function's output.
  - **Entry Points**: Specify which Node to invoke first based on user input.

Edges

Edges define the routing logic in LangGraph:

- **Types of Edges**:
  - **Normal Edges**: Direct transitions from one Node to another.
  - **Conditional Edges**: Determine the next Node(s) to execute based on a function's output.
  - **Entry Points**: Specify which Node to invoke first based on user input.
- **Parallel Execution**: Multiple outgoing edges from a Node can trigger parallel execution of destination Nodes.

Explanation of Edges and their role in LangGraph.

Notebook Demo: Building a Local LangGraph Agent

Q&A & Break

Practical Use Case: Customer Support Agent

Practical Use Case: Customer Support Agent

- **Scenario**: An LLM-powered customer support agent.

Practical Use Case: Customer Support Agent

- **Scenario**: An LLM-powered customer support agent.
- **User Input**: Customer asks about order status.

Practical Use Case: Customer Support Agent

- **Scenario**: An LLM-powered customer support agent.
- **User Input**: Customer asks about order status.
- **LLM Decision**: Determines if it can provide the status directly or if it needs to fetch data from the database.

Practical Use Case: Customer Support Agent

- **Scenario**: An LLM-powered customer support agent.
- **User Input**: Customer asks about order status.
- **LLM Decision**: Determines if it can provide the status directly or if it needs to fetch data from the database.
- **Action Taken**: If data fetch is needed, the agent queries the database and updates the user with the order status.

Practical use case of LLM agents in customer support.

Notebook Demo: Building a Research Agent in LangGraph

Whiteboard & Notebook Demo: Agents in 2025 - LangChain, LangGraph and MCPs

References

1. Toolformer - Schick et al., 2023
2. ReACT - Yao, X., et al., 2023
3. A Survey on Large Language Model based Autonomous Agents - Wang et al., 2023
4. BabyAGI
5. AutoGPT
6. GPT-Researcher
7. Custom GPTs
8. OpenAI function calling Docs
9. OpenAI's Bet on a Cognitive Architecture
10. LangChain Docs