



# Introduction to AI Agent: AI Agents Libraries

## LLMs Series

Dinh-Thang Duong – TA  
Truong-Binh Duong – STA

# Objectives

## ❖ Content



# smolagents

```
Curl
curl -X 'POST' \
'http://0.0.0.0:5000/generative_ai' \
-H 'accept: application/json' \
-H 'Content-Type: application/json' \
-d '{
  "question": "what is BERT?"
}'
```

Request URL  
[http://0.0.0.0:5000/generative\\_ai](http://0.0.0.0:5000/generative_ai)

Server response

Code	Details
200	Response body

```
{
  "answer": "BERT is a bidirectional transformer model for natural language processing pre-trained on a large corpus of text. It stands for Bidirectional Encoder Representations from Transformers. BERT was designed to be similar to OpenAI GPT, and it outperforms other language representation models in various tasks. The table shows the GLUE test results for BERT and OpenAI GPT, and the table shows the SQuAD 1.1 results for different models. BERT comes in two sizes: BERT BASE and BERT LARGE. The larger model has more parameters and generally performs better. Fine-tuning BERT on different tasks is illustrated in Figure 4. Additional details and ablation studies are presented in Appendices A, B, and C."
}
```

[Copy](#) [Download](#)

In this study session, we will discuss

about:

1. Introduction to AI Agents.
2. How to implement an AI Agent using LangChain library.
3. Introduction to smolagents library.
4. How to implement an AI Agent using smolagents.
5. How to implement an AI Agent for Vietnamese news summarization using smolagents.

# Outline

- Introduction
- Implement Agents in LangChain
- Implement Agents in smolagents
- Summarize News with Agents
- Question

 AI

**AI VIET NAM**  
@aivietnam.edu.vn

# Introduction

# Introduction

## ❖ Getting Started

### Textual Description (Prompt):

Write a python function that receive an image and plot it using matplotlib library.

 You  
Write a python function that receive an image and plot it using matplotlib library.

 ChatGPT  
To create a Python function that receives an image and plots it using the Matplotlib library, you can use the following code as an example. You'll need to have Matplotlib and a library like PIL (Pillow) installed to work with images:

```
python
import matplotlib.pyplot as plt
from PIL import Image

def plot_image(image_path):
    try:
        # Open the image using PIL (Pillow)
        img = Image.open(image_path)

        # Create a Matplotlib figure and axis
        fig, ax = plt.subplots()
```

 Copy code



# Introduction

## ❖ What are Large Language Models (LLMs)?



You

write a python code that print "hello world"



ChatGPT

Certainly! Here's a simple Python code that prints "Hello, World!" to the console:

python

Copy code

```
print("Hello, World!")
```

You can run this code in a Python interpreter, and it will display "Hello, World!" as the output.



### ChatGPT App:

1. User inputs a prompt (greetings, task...).
2. ChatGPT (Chatbot) outputs a appropriate response.

# Introduction

## ❖ What are Large Language Models (LLMs)?

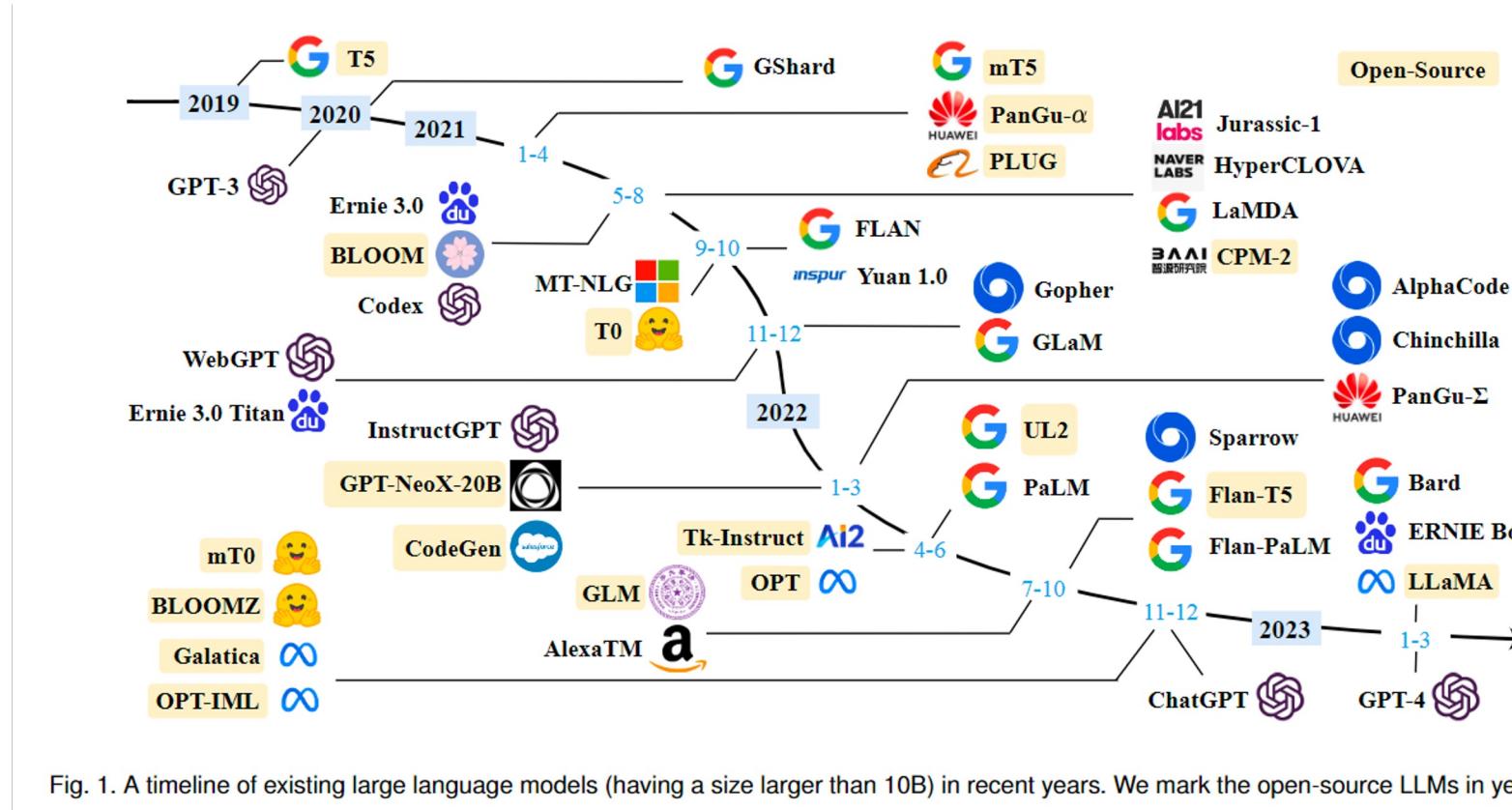


Fig. 1. A timeline of existing large language models (having a size larger than 10B) in recent years. We mark the open-source LLMs in yellow color.

**LLMs (Large Language Models):** AI models (language models) that were trained on a very large corpus of text. This made them capable of performing various NLP tasks with high precision.

# Introduction

## ❖ What are Large Language Models (LLMs)?

LLMs are often pretrained on a vast majority of data and designed to be adaptable to a wide variety of tasks (Foundation models).

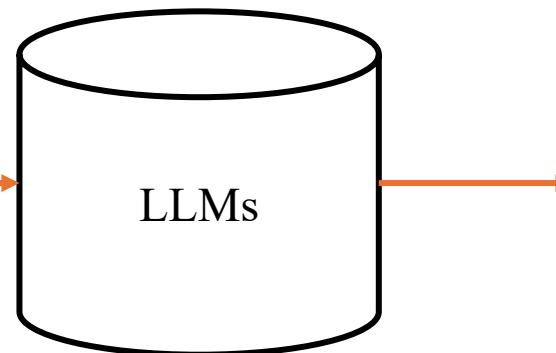


# Introduction

## ❖ LLMs I/O

Input Text

Write a python function that receive an image and plot it using matplotlib library.



Given a “prompt”, LLMs can generate an appropriate response.

Output Text

AI You Write a python function that receive an image and plot it using matplotlib library.

ChatGPT To create a Python function that receives an image and plots it using the Matplotlib library, you can use the following code as an example. You'll need to have Matplotlib and a library like PIL (Pillow) installed to work with images:

```
python
import matplotlib.pyplot as plt
from PIL import Image

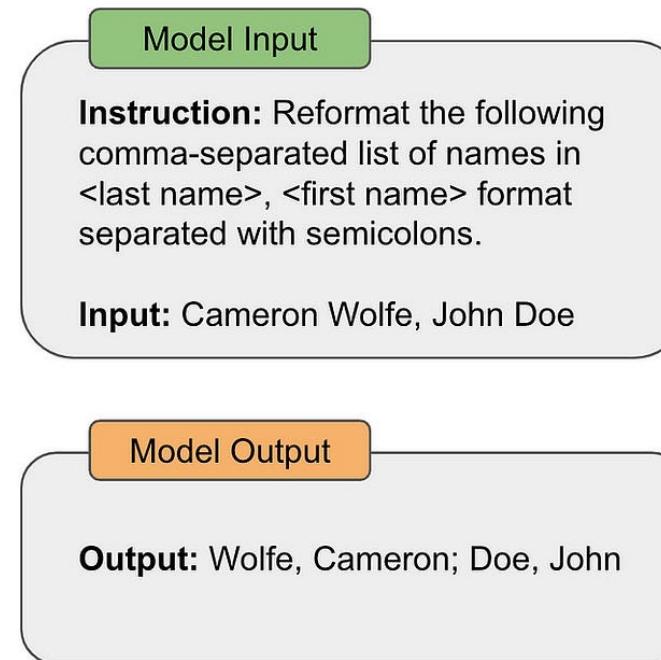
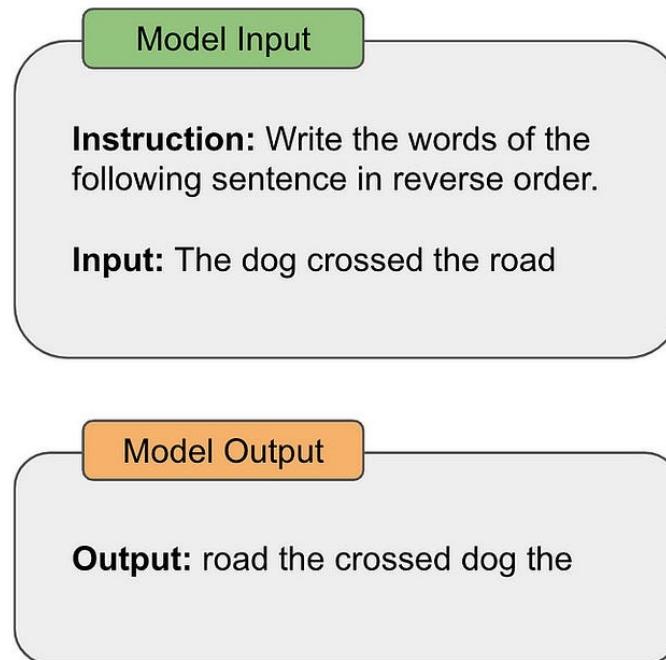
def plot_image(image_path):
    try:
        # Open the image using PIL (Pillow)
        img = Image.open(image_path)

        # Create a Matplotlib figure and axis
        fig, ax = plt.subplots()
```

Copy code

# Introduction

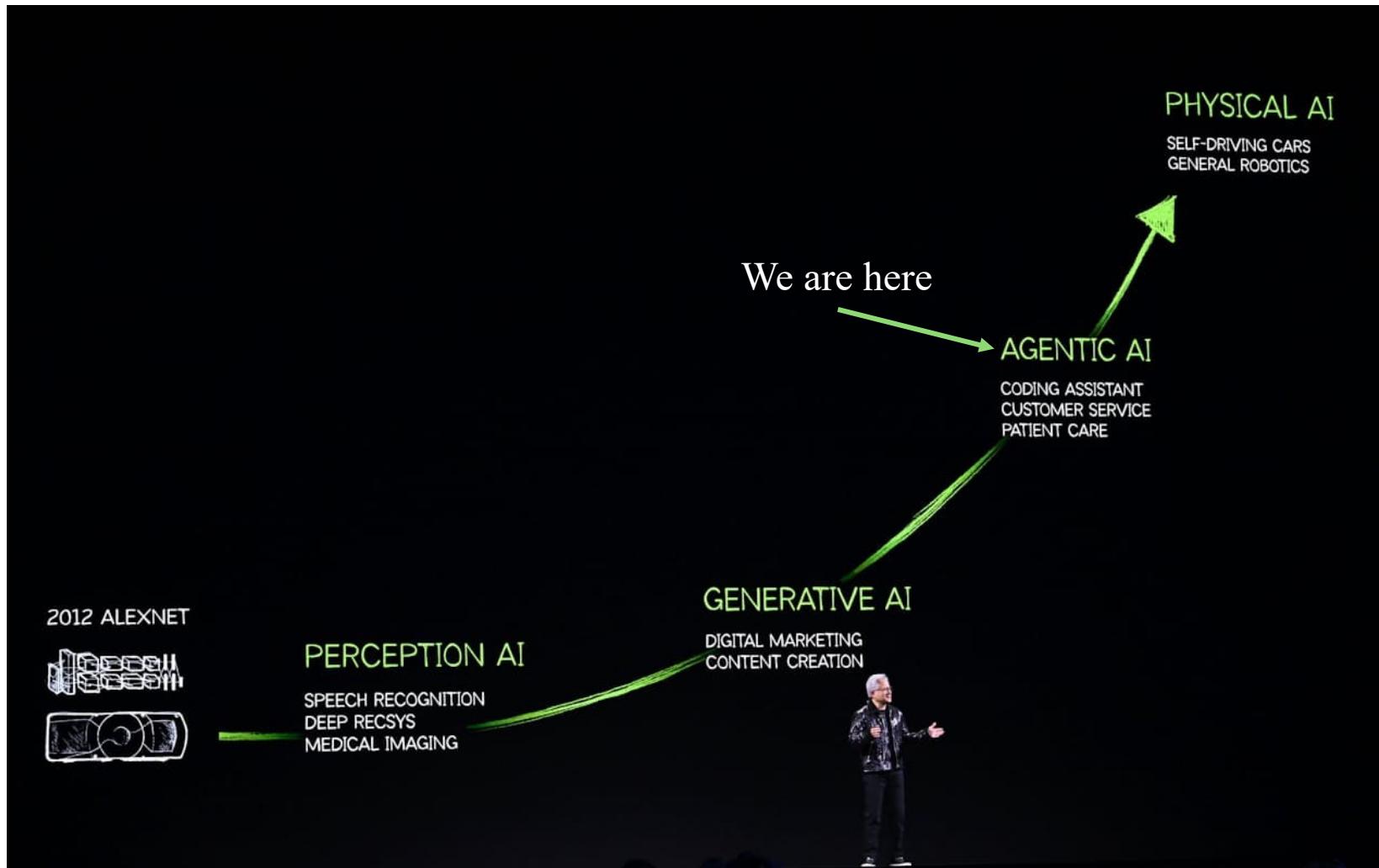
## ❖ Generative AI Prompting



**Prompting:** Refers to a process of providing an input, usually in the form of text data, to a generative AI model to generate a specific output.

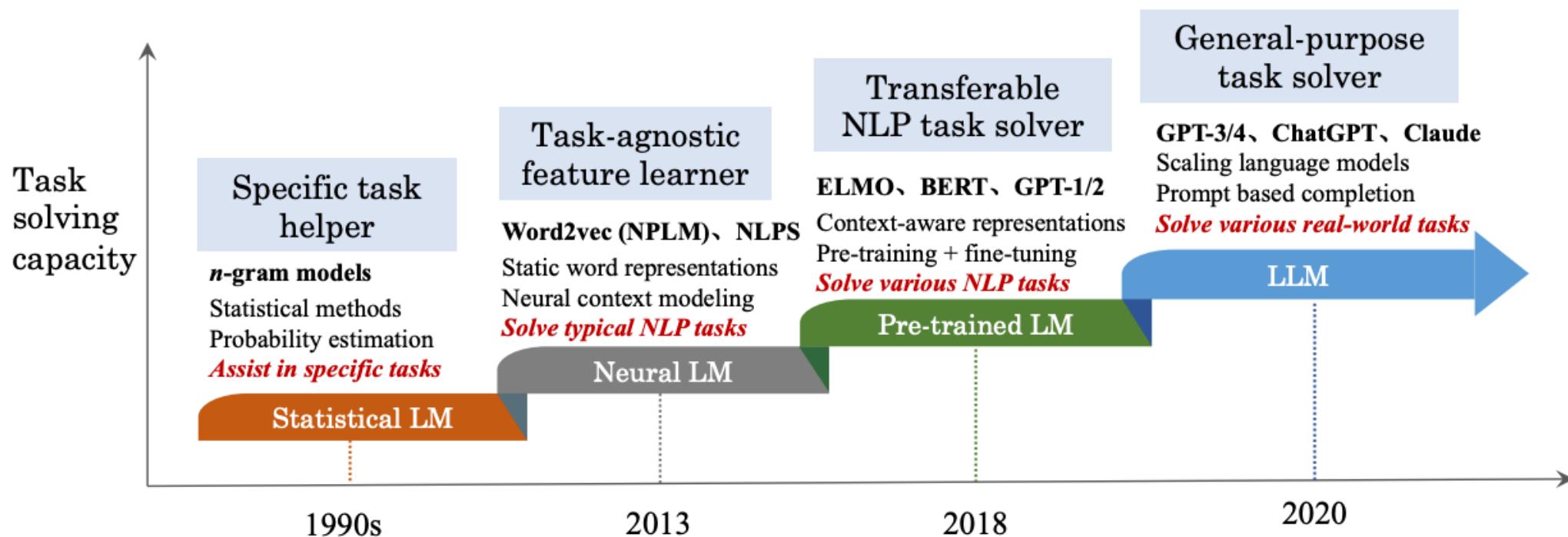
# Introduction

## ❖ Introduction to AI Agents



# Introduction

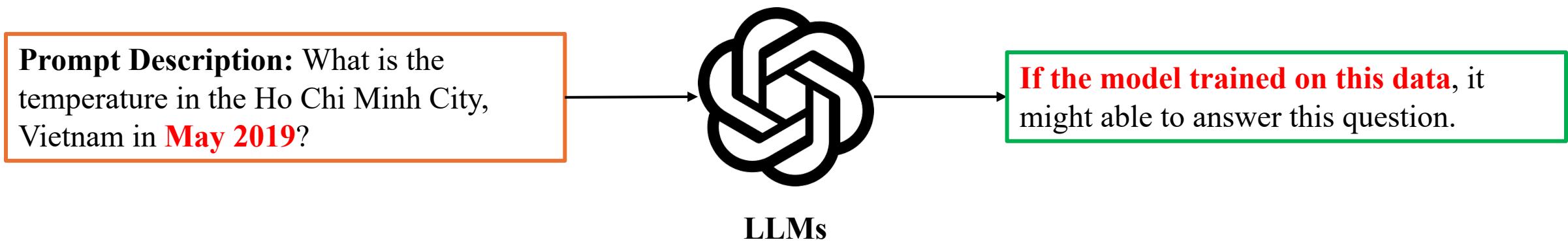
## ❖ Introduction to AI Agents



Pre-trained LLMs are often general purpose.

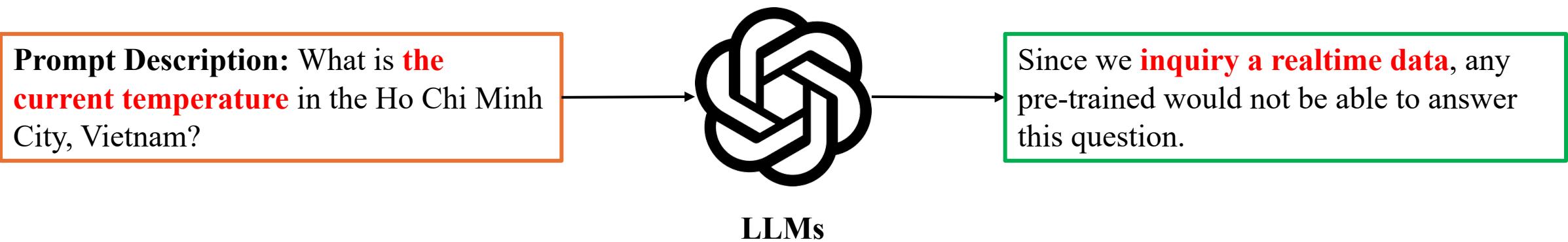
# Introduction

## ❖ Introduction to AI Agents



# Introduction

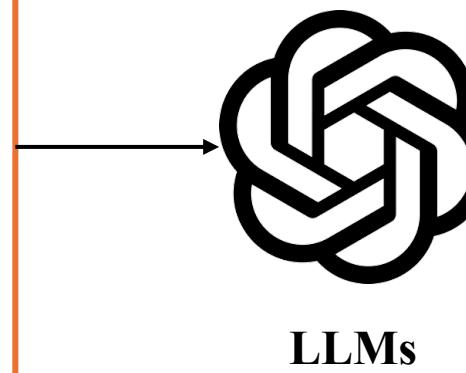
## ❖ Introduction to AI Agents



# Introduction

## ❖ Introduction to AI Agents

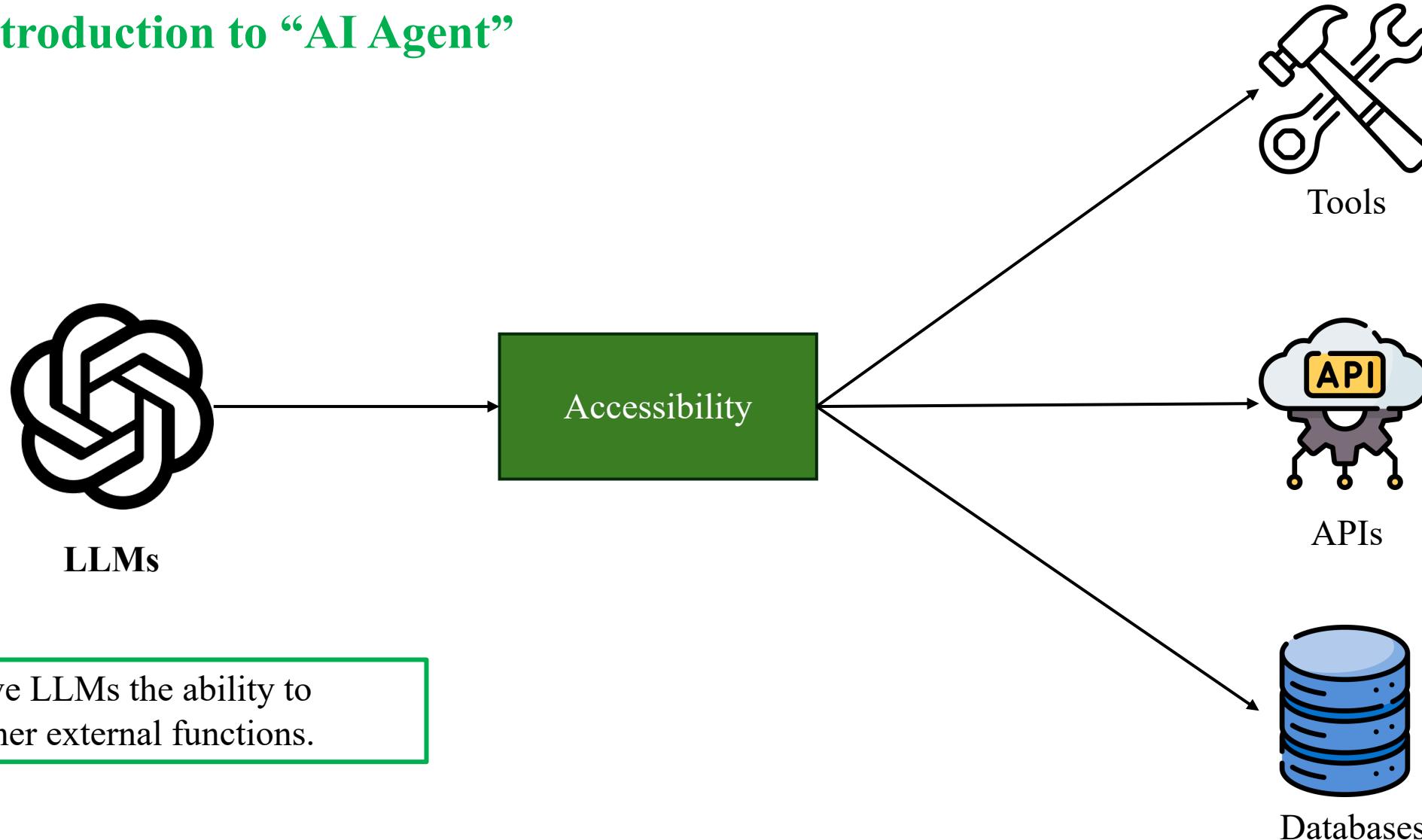
**Prompt Description:** How has the trend in the average daily temperature in Ho Chi Minh City, Vietnam, changed over the last decade, and what impact might this have on local lifestyles and infrastructure? Additionally, can you provide a graphical representation of the trend in temperature fluctuations over this period?



Current pre-trained LLMs might not be able to solve highly complex tasks that require reasoning and planning.

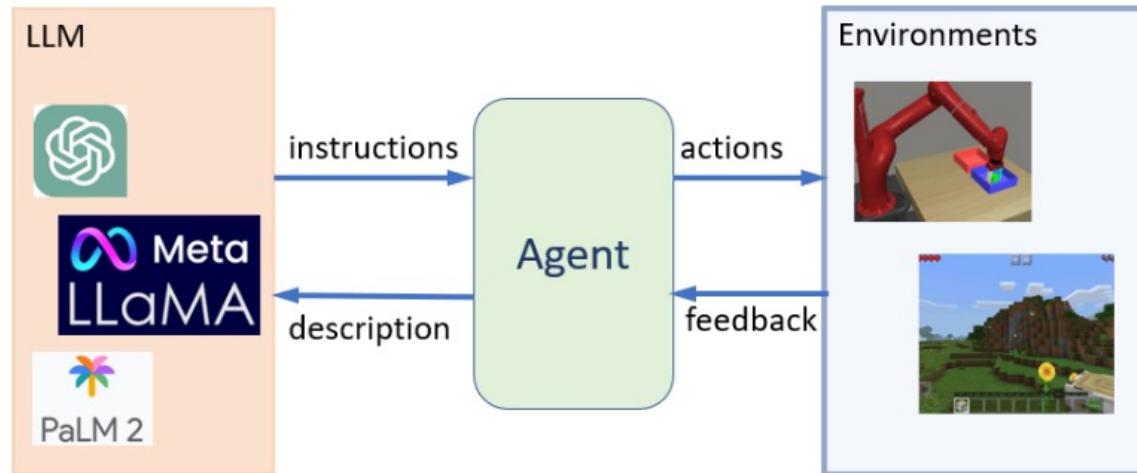
# Introduction

## ❖ Introduction to “AI Agent”



# Introduction

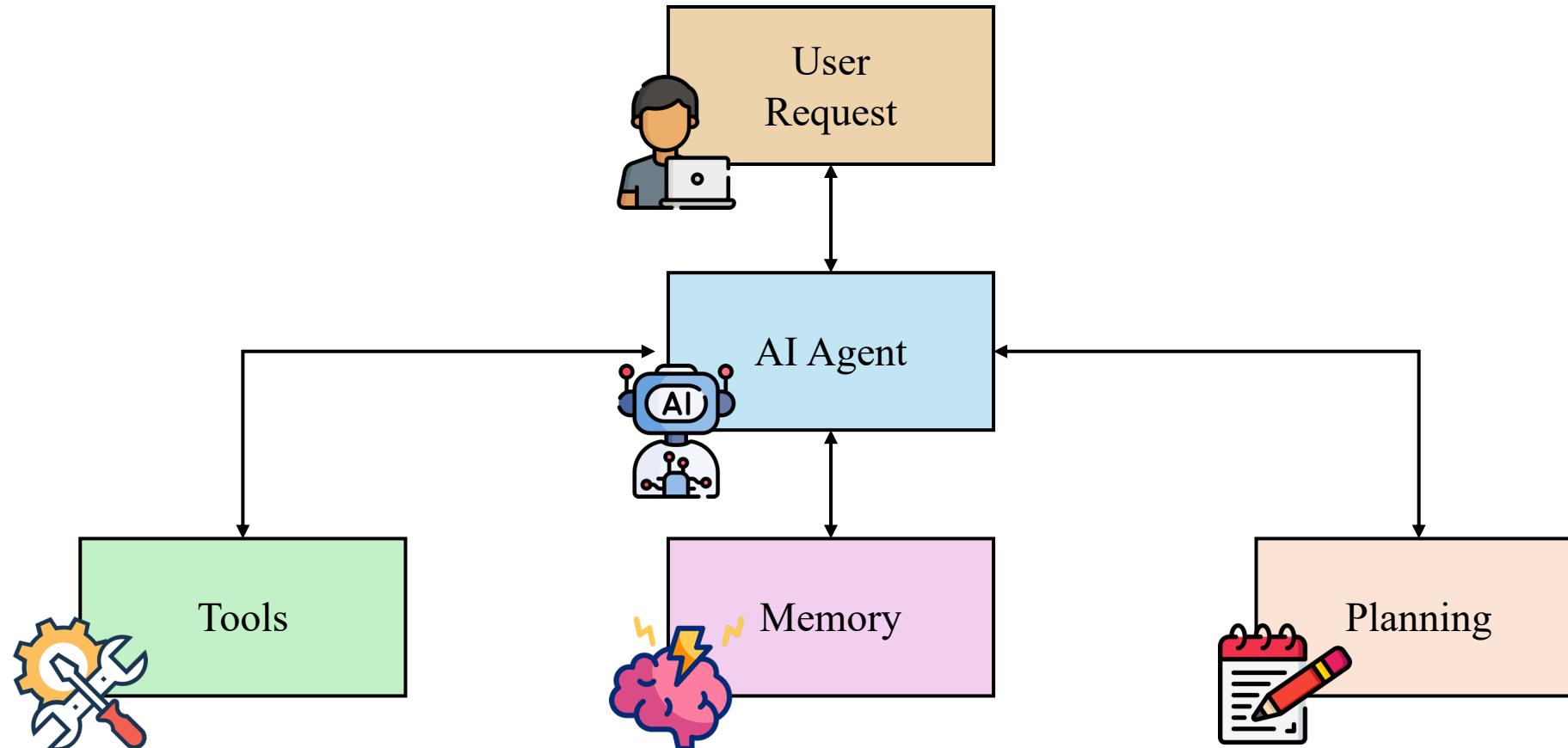
## ❖ LLMs Agent Definition



**LLMs Agent:** Advanced AI systems that utilize large language models (LLMs) as their central computational engine. LLMs Agent is designed to solve complex tasks with required planning ability and accessing to external resources to gather more information.

# Introduction

## ❖ LLMs Agent Framework

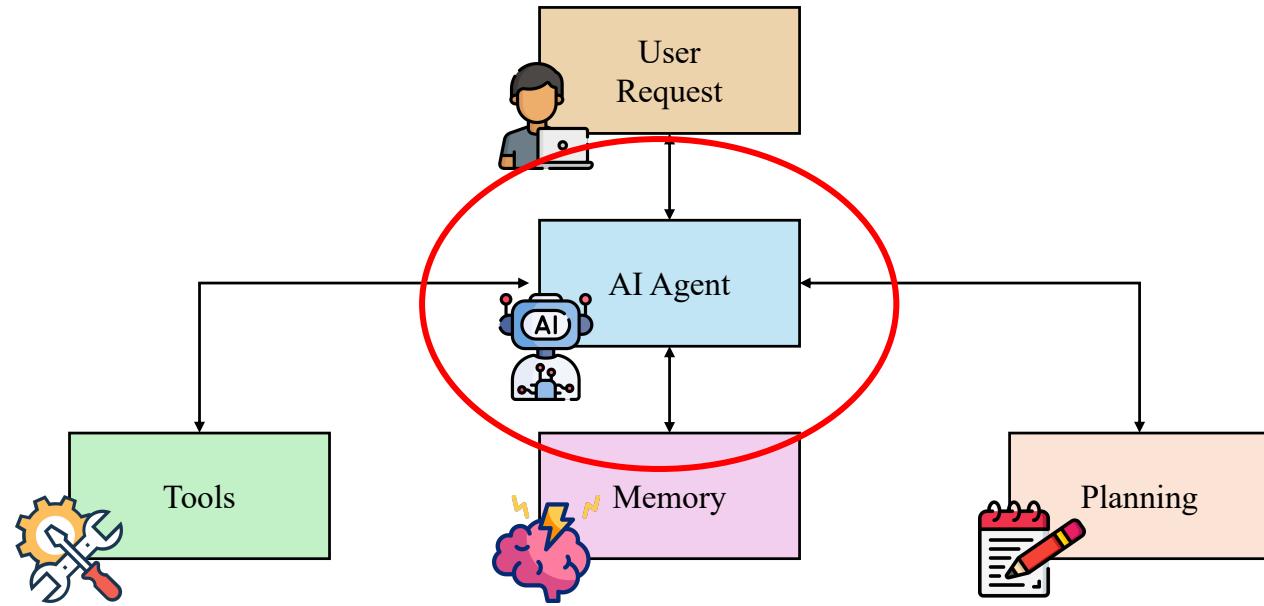


LLMs Agent framework.

# Introduction

## ❖ LLMs Agent: Agent

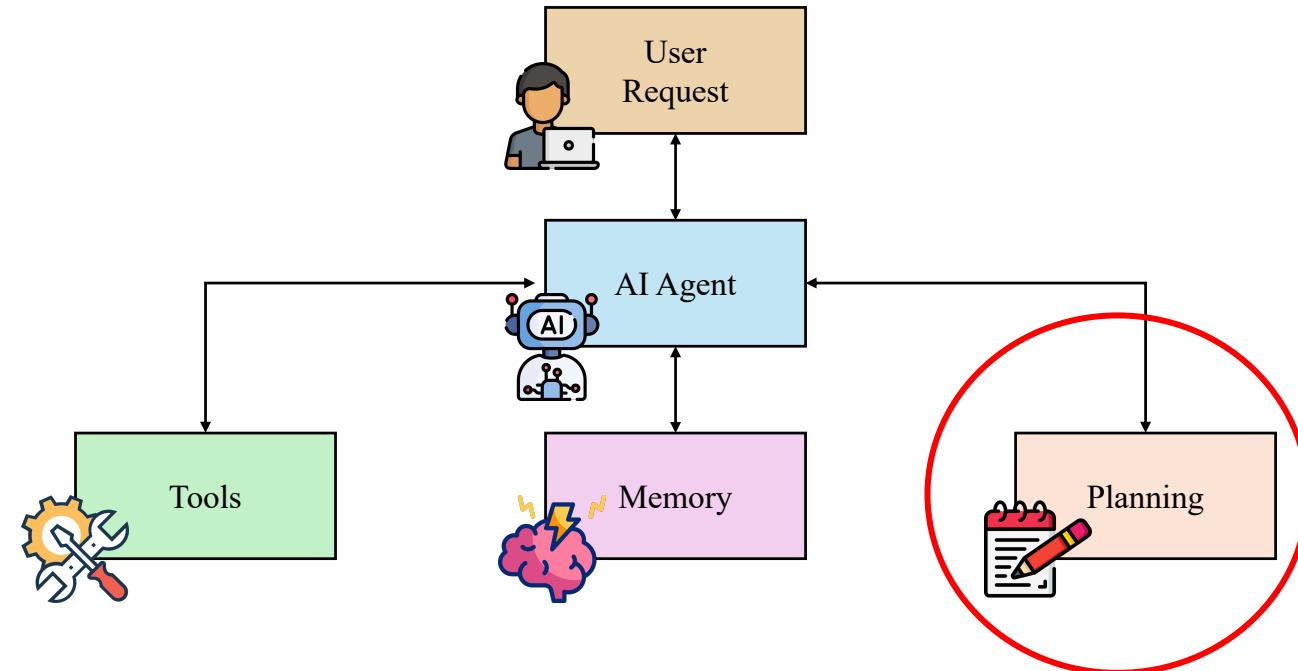
**AI Agent:** An LLM (general-purpose) capabilities serves as the main brain, agent module, or coordinator of the system. This component will be activated using a prompt template that entails important details about how the agent will operate, and the tools it will have access to (along with tool details).



# Introduction

## ❖ LLMs Agent: Planning

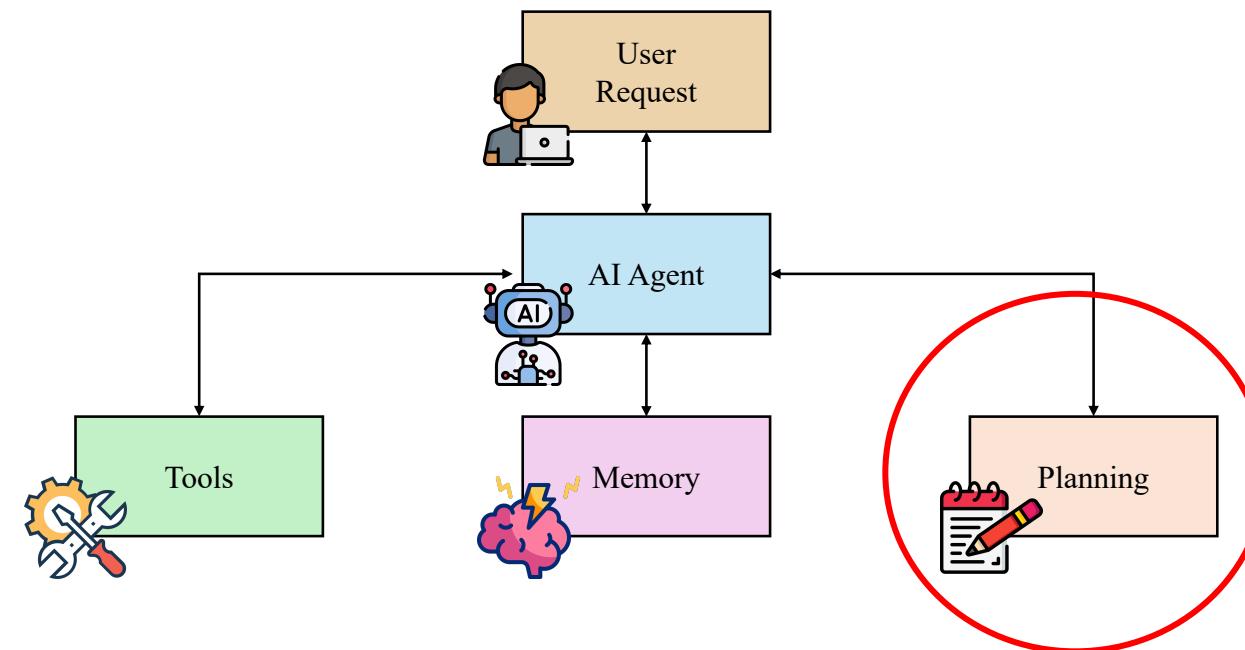
**Planning:** This module helps to break down the necessary steps or subtasks the agent will solve individually to answer the user request.



# Introduction

## ❖ LLMs Agent: Planning

```
Question What is the elevation range for the area that the eastern sector of the Colorado orogeny extends into?  
Thought 1 I need to search Colorado orogeny, find the area that the eastern sector of the Colorado orogeny extends into, then find the elevation range of the area.  
Action 1 Search[Colorado orogeny]  
Observation 1 The Colorado orogeny was an episode of mountain building (an orogeny) in Colorado and surrounding areas.  
Thought 2 It does not mention the eastern sector. So I need to look up eastern sector.  
Action 2 Lookup[eastern sector]  
Observation 2 (Result 1 / 1) The eastern sector extends into the High Plains and is called the Central Plains orogeny.  
Thought 3 The eastern sector of Colorado orogeny extends into the High Plains. So I need to search High Plains and find its elevation range.  
Action 3 Search[High Plains]  
Observation 3 High Plains refers to one of two distinct land regions  
Thought 4 I need to instead search High Plains (United States).  
Action 4 Search[High Plains (United States)]  
Observation 4 The High Plains are a subregion of the Great Plains. From east to west, the High Plains rise in elevation from around 1,800 to 7,000 ft (550 to 2,130 m). [3]  
Thought 5 High Plains rise in elevation from around 1,800 to 7,000 ft, so the answer is 1,800 to 7,000 ft.  
Action 5 Finish[1,800 to 7,000 ft]  
...
```

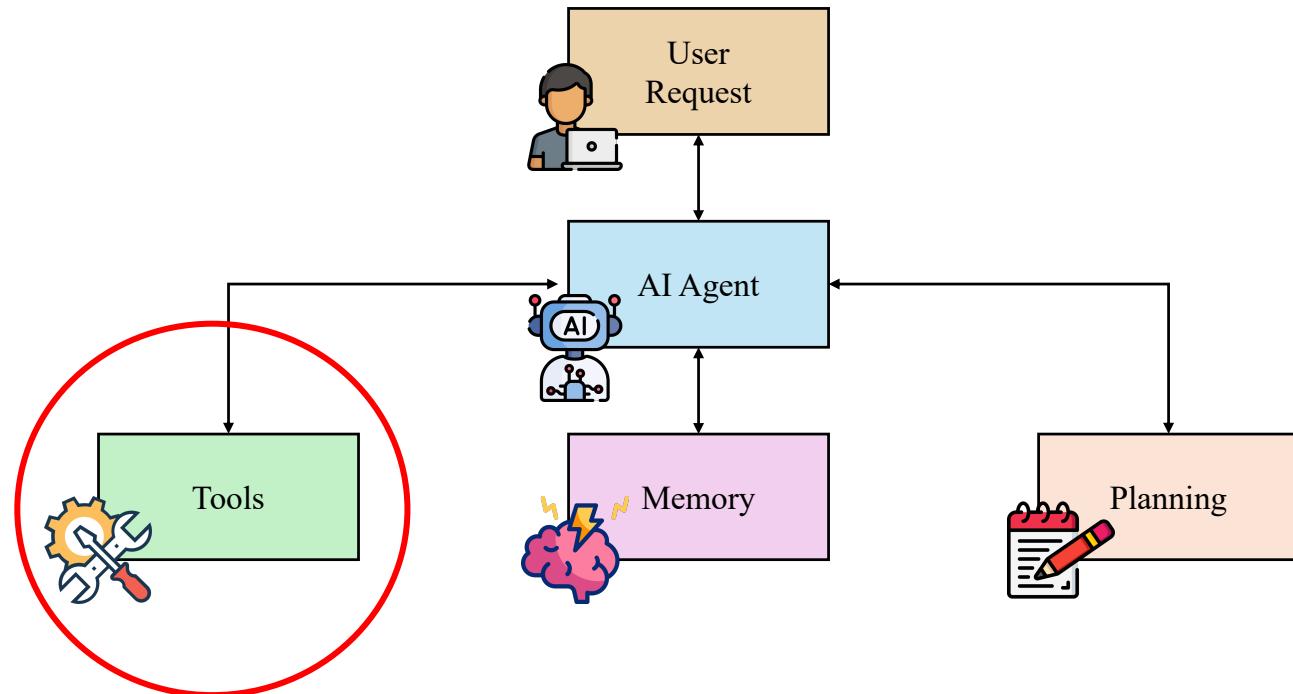


An example of ReAct prompting.

# Introduction

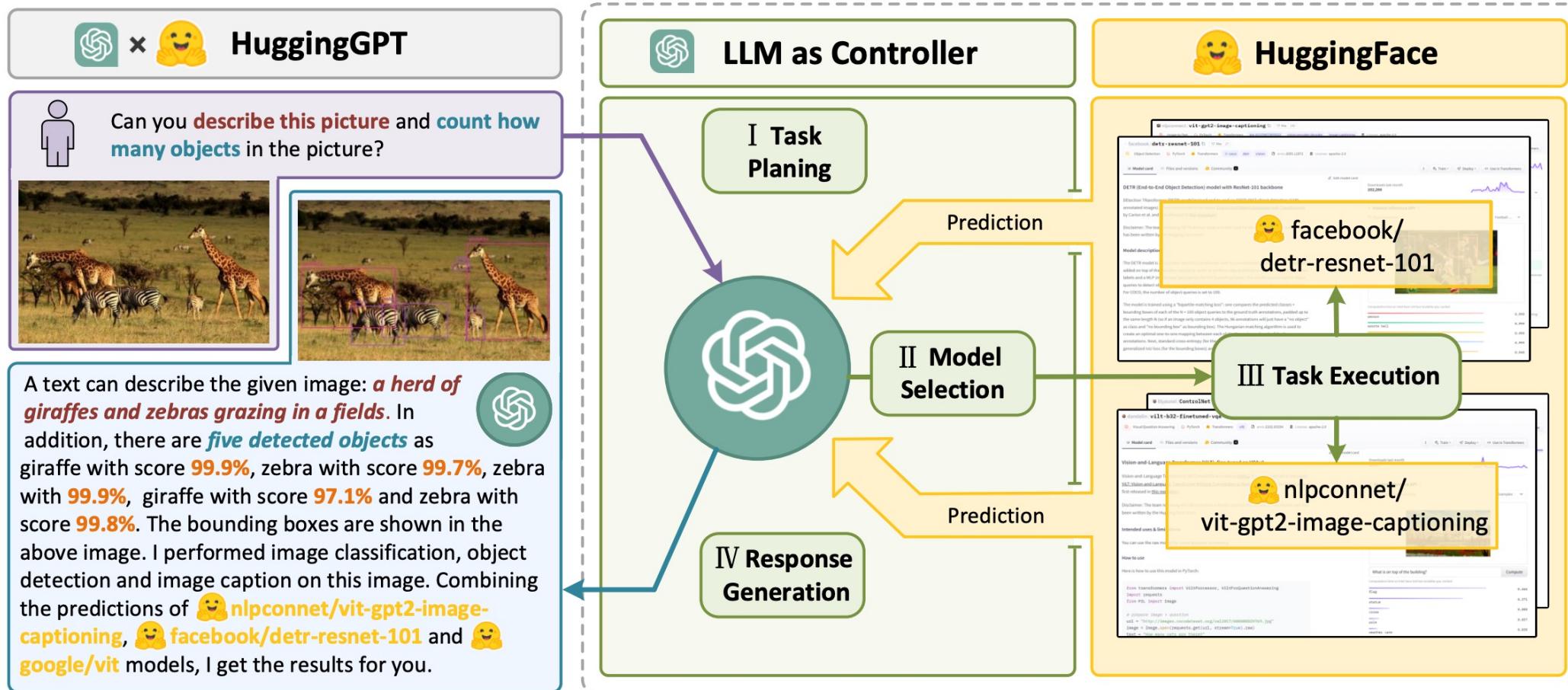
## ❖ LLMs Agent: Tools

**Tools:** A set of tool/s that enables the LLM agent to interact with external environments such as Wikipedia Search API, Code Interpreter, and Math Engine.



# Introduction

## ❖ LLMs Agent: Tools

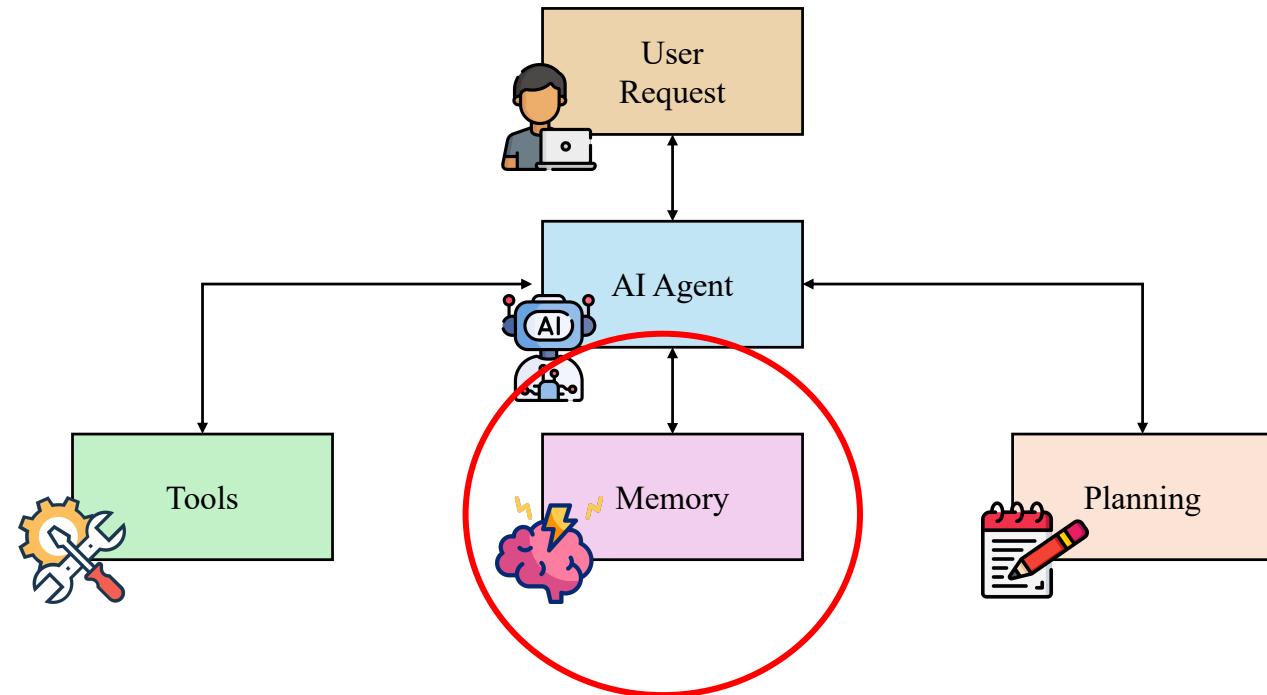


HuggingFaceGPT – Using LLMs to decide which AI models to be used to solve a task.

# Introduction

## ❖ LLMs Agent: Memory

**Memory:** This module helps to store the agent's internal logs including past thoughts, actions, and observations from the environment, including all interactions between agent and user.



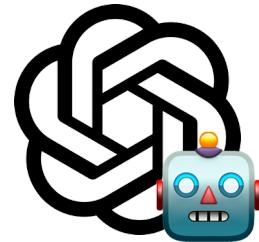
# Introduction

## ❖ General LLMs vs LLMs Agent



**Pre-trained LLMs**

1. Often built with general purpose.
2. Don't have the ability to access external tools.
3. Might not be able to solve highly complex tasks.



**LLMs Agent**

1. Can be either Task specific or General Purpose.
2. Have the ability to access external tools.
3. Might be able to solve highly complex tasks.

# Introduction

## ❖ Introduction

Agency Level	Description	How that's called	Example Pattern
☆☆☆	LLM output has no impact on program flow	Simple processor	<code>process_llm_output(llm_response)</code>
★☆☆	LLM output determines basic control flow	Router	<code>if llm_decision(): path_a() else: path_b()</code>
★★☆	LLM output determines function execution	Tool call	<code>run_function(llm_chosen_tool, llm_chosen_args)</code>
★★★	LLM output controls iteration and program continuation	Multi-step Agent	<code>while llm_should_continue(): execute_next_step()</code>
★★★	One agentic workflow can start another agentic workflow	Multi-Agent	<code>if llm_trigger(): execute_agent()</code>

# Introduction

## ❖ Overview of AI Agent Series

Day 01: AI Agent Libraries

Day 02:  
LLMs Tools Calling

Day 03:  
ReAct AI Agent

- High-level understanding of AI Agent.
- Investigate how to build AI Agent applications through popular libraries.

- First low-level understanding of AI Agent.
- Investigate the simplest and early form of AI Agent: Make LLMs using tools to solve tasks.

- Second low-level understanding of AI Agent.
- Investigate the general baseline of current AI Agent: Thought-Action-Observe through ReAct AI Agent.

# Implement Agents in LangChain

# Implement Agents in LangChain

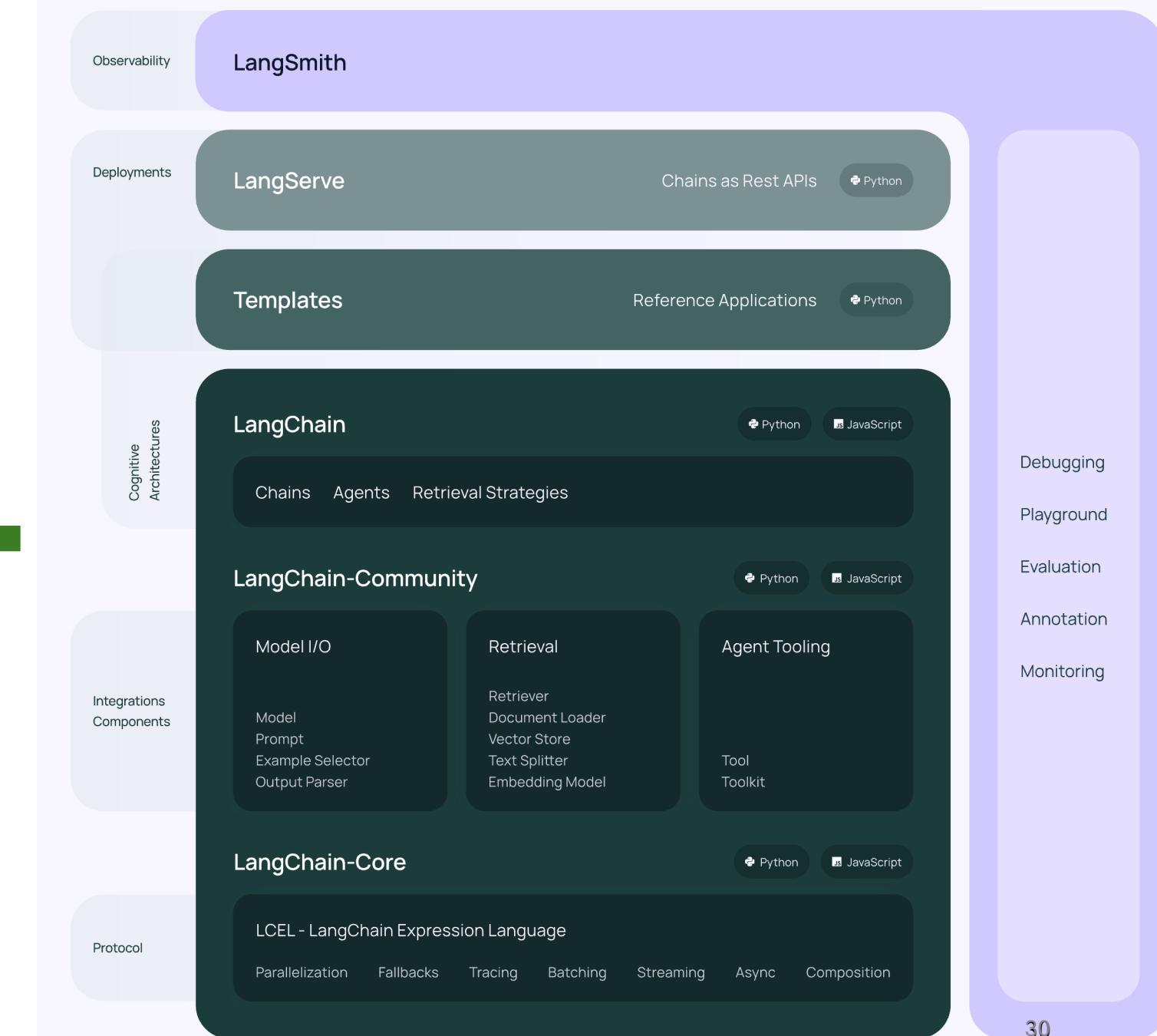
## ❖ Recap



**LangChain:** A framework for developing applications powered by large language models (LLMs). LangChain simplifies every stage of the LLM application lifecycle: Development, Productionization, Deployment.

# Implement Agents in LangChain

## ❖ Recap



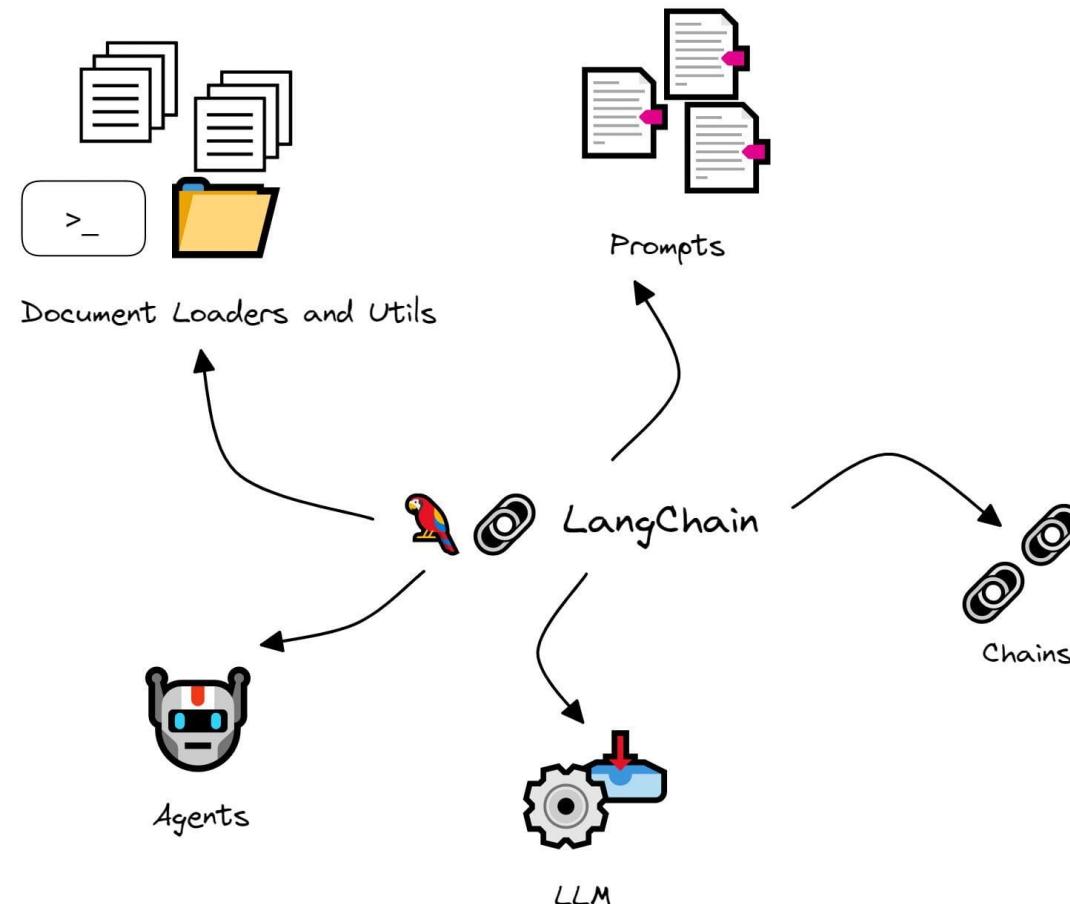
# Implement Agents in LangChain

## ❖ Recap: LangChain Components



# Implement Agents in LangChain

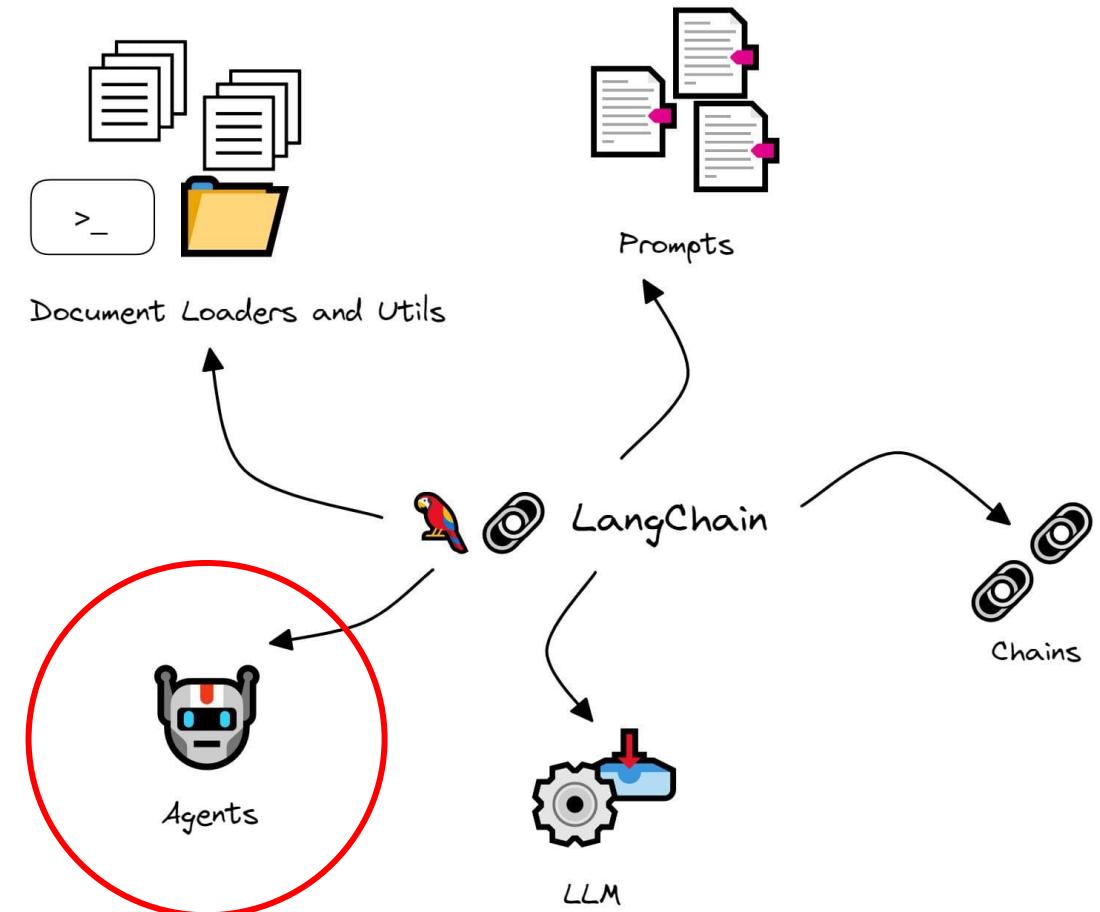
## ❖ LangChain Components



# Implement Agents in LangChain

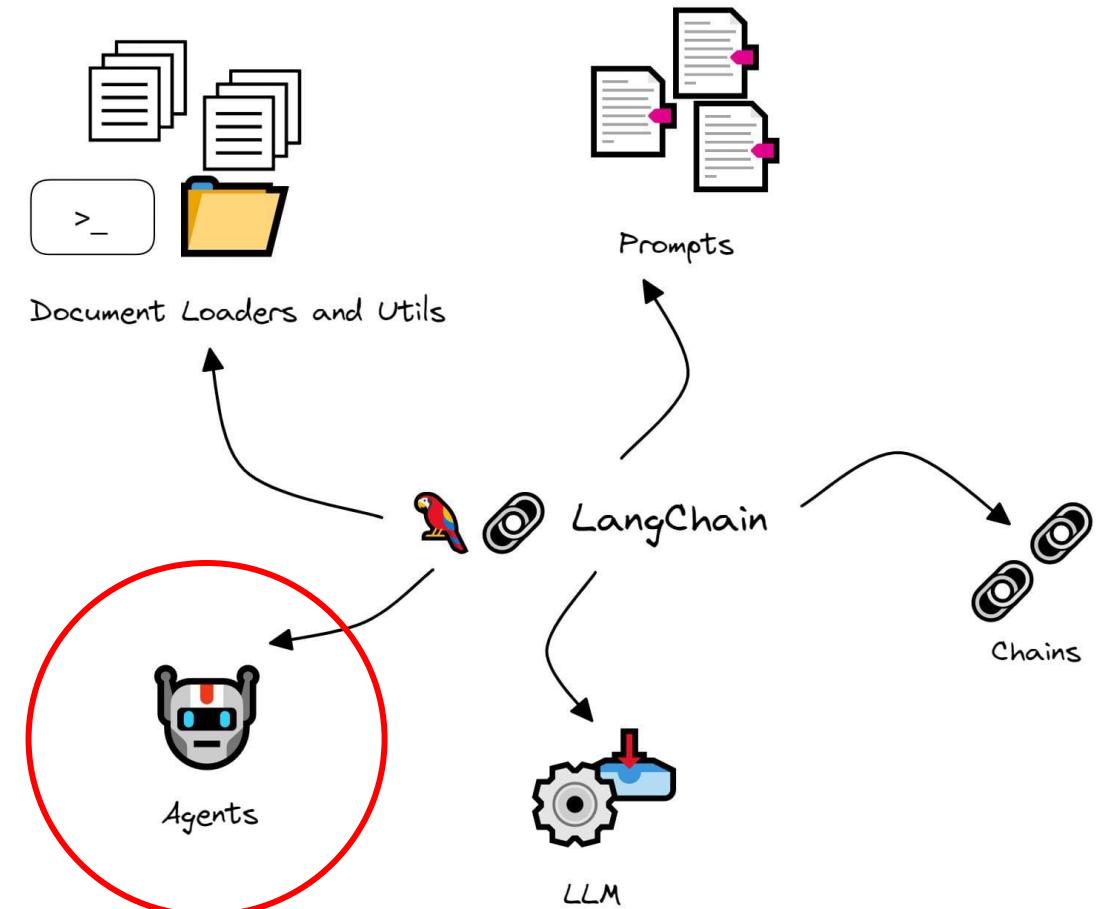
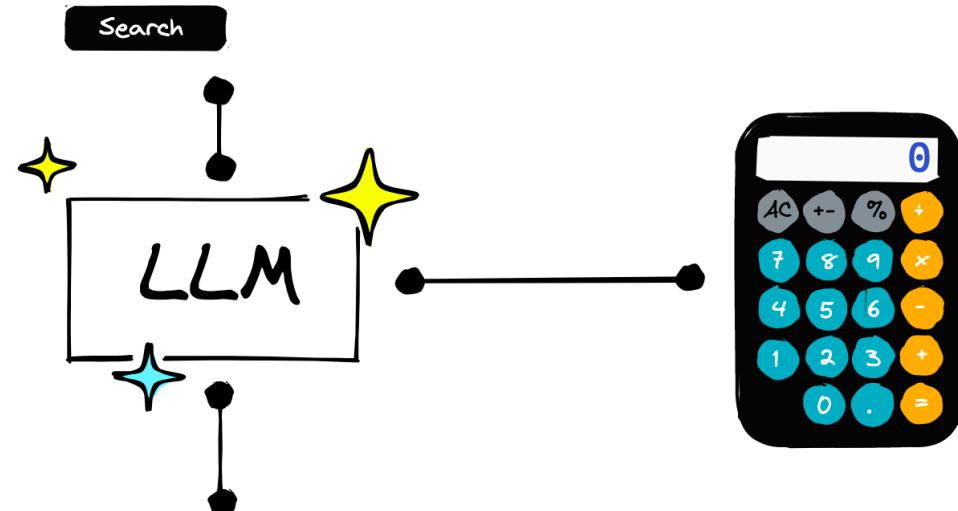
## ❖ LangChain Agent Component

**Agents:** Software agents or programs that can interact with the LLM and other tools or services to perform specific tasks or actions.



# Implement Agents in LangChain

## ❖ LangChain Agent Component





# Implement Agents in LangChain

## ❖ Example: Online Search



```
1 from langchain_community.tools.tavily_search import TavilySearchResults
2 import getpass
3 import os
4
5 os.environ["TAVILY_API_KEY"] = getpass.getpass()
6 search = TavilySearchResults()
```

Tavily

Initialize Tavily Search (or other search API).



# Implement Agents in LangChain

## ❖ Example: Online Search

The screenshot shows the Tavily AI platform's Overview page. On the left, there's a sidebar with links: Overview, Research Assistant, Research Reports, API Playgroud, and Documentation. The main area has a purple header bar titled "Researcher". It displays the "CURRENT PLAN" as "Researcher" and the "API Limit" as "8 / 1,000 Requests". Below this, there's a section for the "API Key" which contains the value "tvly-\*\*\*\*\*". At the bottom, there's a message "Have any questions, feedback or need support? We'd love to hear from you!" and a "Contact us" button.

**Tavily API Key.**

# Implement Agents in LangChain

## ❖ Example: Online Search



```
7 results = search.invoke(  
8     {  
9         "query": "What is quantum mechanics?",  
10        "include_answer": False  
11    }  
12 )
```

```
[{'url': 'https://en.wikipedia.org/wiki/Quantum\_mechanics',  
 'content': 'Therefore, since \n\nsin\n\nu\n\nis proportional to the square root of the energy E, the wavelength \n\nlambda is inversely proportional to the energy E. This is the famous de Broglie relation, which was proposed in 1924 by Louis de Broglie, and it is now known to be one of the basic principles of quantum mechanics.'},  
 {'url': 'https://www.britannica.com/science/quantum-mechanics',  
 'content': 'Given the wavelength of the wavefunction, the energy of the particle can be calculated using the de Broglie relation: E = hc/\n\nlambda, where h is Planck's constant and c is the speed of light.'},  
 {'url': 'https://www.energy.gov/science-energy/quantum-mechanics',  
 'content': 'Quantum mechanics is the theory of the physical properties of nature at the scale of atoms and subatomic particles.'},  
 {'url': 'https://www.livescience.com/35777-what-is-quantum-mechanics.html',  
 'content': "How is quantum mechanics different from classical mechanics? Quantum mechanics is based on the idea that particles can exist in multiple states simultaneously, while classical mechanics assumes that particles have definite positions and velocities at any given time."}]
```

Search data using Tavily API.

# Implement Agents in LangChain

## ❖ Example: Online Search



```
1 from langchain_openai import ChatOpenAI
2
3 os.environ["OPENAI_API_KEY"] = getpass.getpass()
4 tools = [search]
5 llm = ChatOpenAI(model="gpt-3.5-turbo-0125", temperature=0)
```



Initialize GPT from OpenAI (required API Key).

# Implement Agents in LangChain

## ❖ Example: Online Search



```
6 from langchain import hub
7
8 prompt = hub.pull("hwchase17/openai-functions-agent")
9 prompt.messages
```

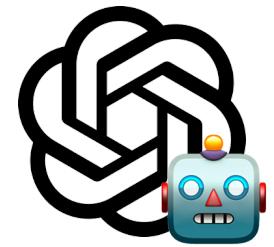
```
[SystemMessagePromptTemplate(prompt=PromptTemplate(input_variables=[], template='You are a helpful assistant')),
MessagesPlaceholder(variable_name='chat_history', optional=True),
HumanMessagePromptTemplate(prompt=PromptTemplate(input_variables=['input'], template='{input')}'),
MessagesPlaceholder(variable_name='agent_scratchpad')]
```

Create prompt to chat with  
OpenAI GPT.

# Implement Agents in LangChain

## ❖ Example: Online Search

```
10 from langchain.agents import create_tool_calling_agent
11 from langchain.agents import AgentExecutor
12
13 agent = create_tool_calling_agent(llm, tools, prompt)
14 agent_executor = AgentExecutor(agent=agent, tools=tools, verbose=True)
```



Agent

Initialize Agent (equipped with  
LLM and tools).

# Implement Agents in LangChain

## ❖ Example: Online Search



```
12 agent_executor.invoke(  
13     {  
14         "input": "whats the weather in Ho Chi Minh City, Vietnam?"  
15     }  
16 )
```

```
[{"url": "https://www.weatherapi.com/", "content": "{'location': {'name': 'Ho Chi Minh City',  
- Temperature: 35.0°C (95.0°F)  
- Condition: Partly cloudy  
- Wind: 22.0 km/h from the south  
- Pressure: 1007.0 mb  
- Humidity: 50%  
- Feels like: 38.5°C (101.3°F)  
- Visibility: 10.0 km  
- UV Index: 9.0  
- Chance of rain: 0.0%"}]
```

*For more detailed information, you can visit [Weather API] (<https://www.weatherapi.com/>) .*

# Implement Agents in LangChain

## ❖ Example: Online Search

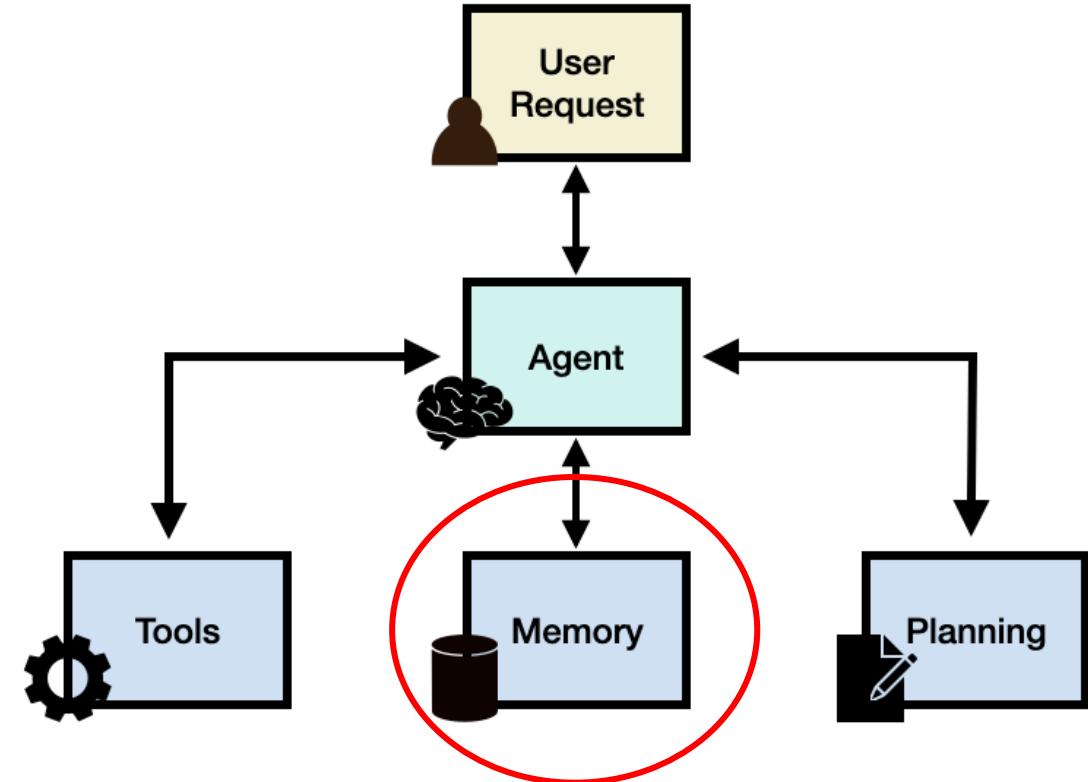


```
1 agent_executor.invoke(  
2   {  
3     "input": "Hi! My name is AI VIETNAM",  
4     "chat_history": []  
5   }  
6 )
```

> Entering new AgentExecutor chain...  
*Hello AI VIETNAM! How can I assist you today?*

> Finished chain.

```
{'input': 'Hi! My name is AI VIETNAM',  
 'chat_history': [],  
 'output': 'Hello AI VIETNAM! How can I assist you today?'}
```





# Implement Agents in LangChain

## ❖ Example: Online Search



```
18 from langchain_core.messages import AIMessage, HumanMessage
19
20 agent_executor.invoke(
21     {
22         "chat_history": [
23             HumanMessage(content="Hi! My name is AI VIETNAM"),
24             AIMessage(content="Hello AI VIETNAM! How can I assist you today?"),
25         ],
26         "input": "what is my name?",
27     }
28 )
```

```
> Entering new AgentExecutor chain...
Your name is AI VIETNAM. How can I help you further?

> Finished chain.
```

LLMs now can remember previous chat.



# Implement Agents in LangChain

## ❖ Example: Online Search

```
30 from langchain_community.chat_message_histories import ChatMessageHistory
31 from langchain_core.runnables.history import RunnableWithMessageHistory
32
33 message_history = ChatMessageHistory()
34
35 agent_with_chat_history = RunnableWithMessageHistory(
36     agent_executor,
37     lambda session_id: message_history,
38     input_messages_key="input",
39     history_messages_key="chat_history",
40 )
```

Using ChatMessageHistory to automatically gather chat history.



# Implement Agents in LangChain

## ❖ Example: Online Search

● ● ●

```
41 agent_with_chat_history.invoke(  
42     {"input": "Hi! I'm AI VIETNAM"},  
43     config={"configurable": {"session_id": "<foo>"},  
44 )
```

```
> Entering new AgentExecutor chain...  
Hello AI VIETNAM! How can I assist you today?  
  
> Finished chain.  
  
{'input': 'Hi! I'm AI VIETNAM',  
 'chat_history': [],  
 'output': 'Hello AI VIETNAM! How can I assist you today?'}
```

● ● ●

```
46 agent_with_chat_history.invoke(  
47     {"input": "What's my name?"},  
48     config={"configurable": {"session_id": "<foo>"},  
49 )
```

```
> Entering new AgentExecutor chain...  
Your name is AI VIETNAM. How can I help you today, AI VIETNAM?  
  
> Finished chain.  
  
{'input': "What's my name?",  
 'chat_history': [HumanMessage(content="Hi! I'm AI VIETNAM"),  
 AIMessage(content='Hello AI VIETNAM! How can I assist you today?')],  
 'output': 'Your name is AI VIETNAM. How can I help you today, AI VIETNAM?'}
```

# Implement Agents in LangChain

## ❖ Example: Find Word Length



# Implement Agents in LangChain

## ❖ Example: Find Word Length



```
1 import os
2 from langchain_openai import ChatOpenAI
3
4 from dotenv import load_dotenv
5
6 load_dotenv()
7
8 OPENAI_API_KEY = os.getenv('OPENAI_API_KEY')
9 os.environ["OPENAI_API_KEY"] = OPENAI_API_KEY
10 llm = ChatOpenAI(model="gpt-3.5-turbo-0125", temperature=0)
```



Initialize GPT from OpenAI (required API Key).

# Implement Agents in LangChain

## ❖ Example: Find Word Length



```
11 from langchain.agents import tool
12
13 @tool
14 def get_word_length(word: str) -> int:
15     """Returns the length of a word."""
16     return len(word)
17
18 get_word_length.invoke("abc")
19
20 tools = [get_word_length]
```

**@decorator**

Using `@tool` decorator to custom an agent.

# Implement Agents in LangChain

## ❖ Example: Find Word Length

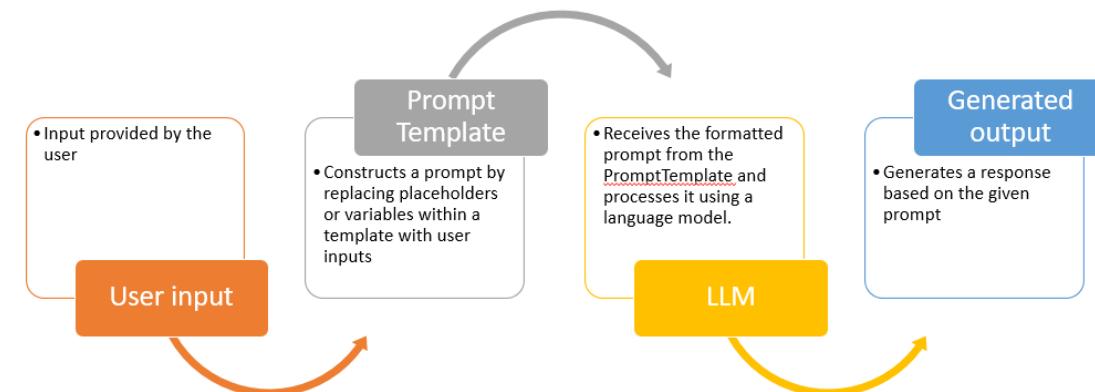


```
22 from langchain_core.prompts import ChatPromptTemplate, MessagesPlaceholder
23
24 prompt = ChatPromptTemplate.from_messages(
25     [
26         (
27             "system",
28             "You are very powerful assistant, but don't know current events",
29         ),
30         ("user", "{input}"),
31         MessagesPlaceholder(variable_name="agent_scratchpad"),
32     ]
33 )
```

# Implement Agents in LangChain

## ❖ Example: Find Word Length

```
34 from langchain.agents.format_scratchpad.openai_tools import (
35     format_to_openai_tool_messages,
36 )
37 from langchain.agents.output_parsers.openai_tools import
38     OpenAIToolsAgentOutputParser
39
40 llm_with_tools = llm.bind_tools(tools)
41 agent = (
42     {
43         "input": lambda x: x["input"],
44         "agent_scratchpad": lambda x: format_to_openai_tool_messages(
45             x["intermediate_steps"]
46         ),
47     },
48     | prompt
49     | llm_with_tools
50     | OpenAIToolsAgentOutputParser()
51 )
```



# Implement Agents in LangChain

## ❖ Example: Find Word Length



```
52 from langchain.agents import AgentExecutor
53
54 agent_executor = AgentExecutor(agent=agent, tools=tools, verbose=True)
55
56 agent_executor.stream({"input": "How many letters in the word AIVIETNAM"})
```

> Entering new AgentExecutor chain...

*Invoking: `get\_word\_length` with `{'word': 'AIVIETNAM'}`*

*9The word "AIVIETNAM" has 9 letters.*

> Finished chain.

# Implement Agents in smolagents

# Implement Agents in smolagents

## ❖ Introduction

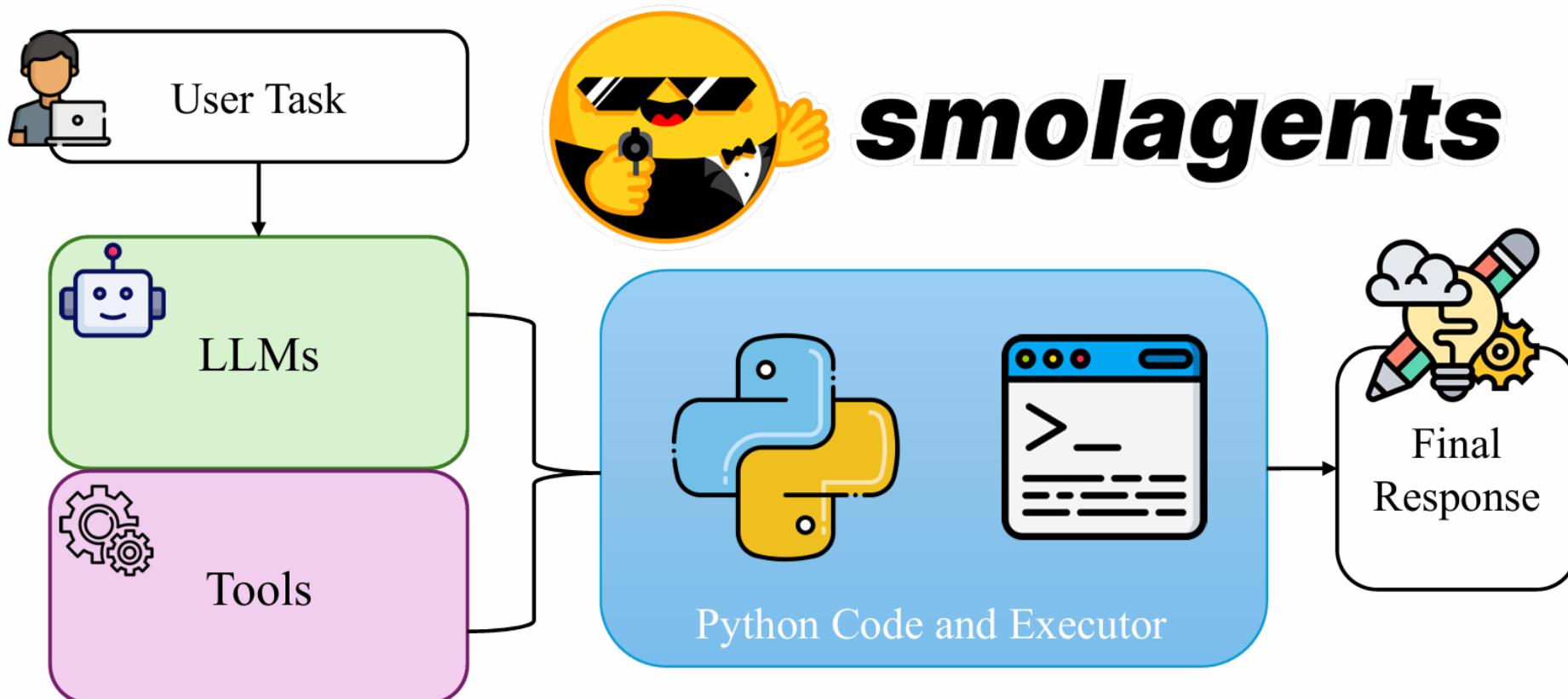


# smolagents

**Smolagents**, developed by HuggingFace team, is an open-source, lightweight AI agent library that allows developers to create powerful agents with minimal code. With a core codebase of approximately 1,000 lines in `agents.py`, smolagents reduces unnecessary abstractions, making the development process straightforward and accessible.

# Implement Agents in smolagents

## ❖ Introduction





# Implement Agents in smolagents

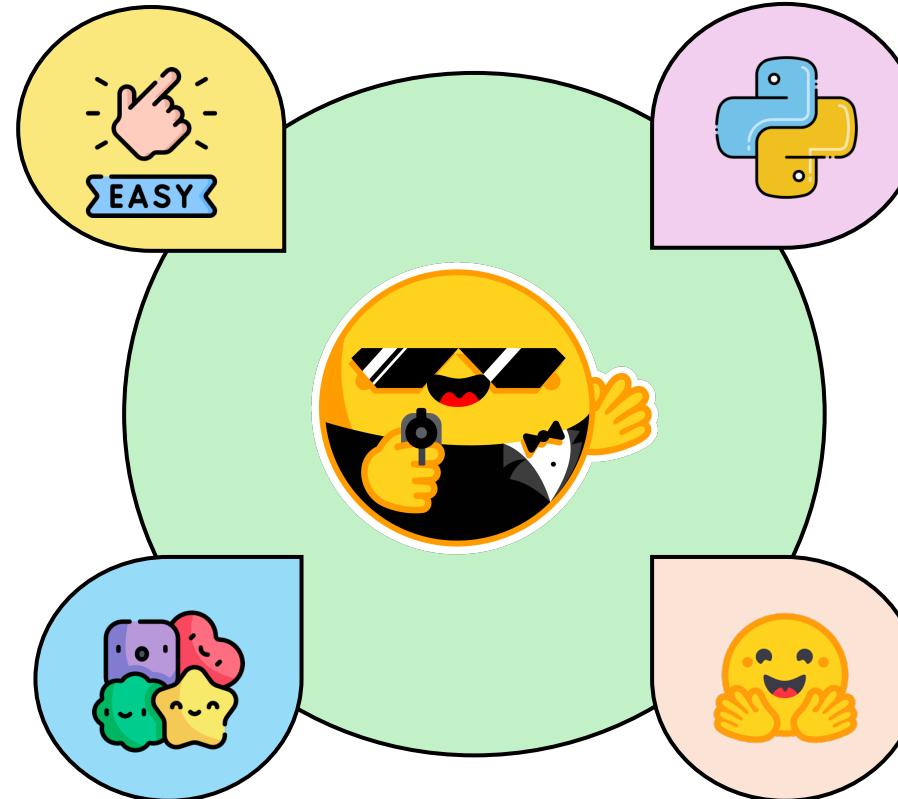
## ❖ Introduction

```
In [1]: from smolagents import CodeAgent, DuckDuckGoSearchTool, HfApiModel
.....
.... agent = CodeAgent(tools=[DuckDuckGoSearchTool()], model=HfApiModel())
.... agent.run("How many seconds would it take for a leopard at full speed to run through Pont des Arts?")
```

# Implement Agents in smolagents

## ❖ Introduction

Simplicity



Code-First Approach

Support for any LLM

Hub integrations

# Implement Agents in smolagents

## ❖ Introduction



Simplicity

EASY

Minimal code complexity and abstractions, to make the framework easy to understand, adopt and extend.



```
from smolagents import CodeAgent, HfApiModel

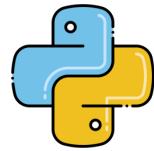
agent = CodeAgent(
    tools=[],
    model=HfApiModel()
)

agent.run("Số Fibonacci thứ 50 là bao nhiêu?")
```



# Implement Agents in smolagents

## ❖ Introduction



### Code-First Approach

First-class support for Code Agents that write their actions directly in code, removing the need for parsing and simplifying tool calling.

*Output message of the LLM:* \_\_\_\_\_

Thought: To calculate the 50th Fibonacci number, I will use a recursive function with memoization to optimize performance. This approach avoids redundant calculations and significantly speeds up the process compared to a naive recursive solution.

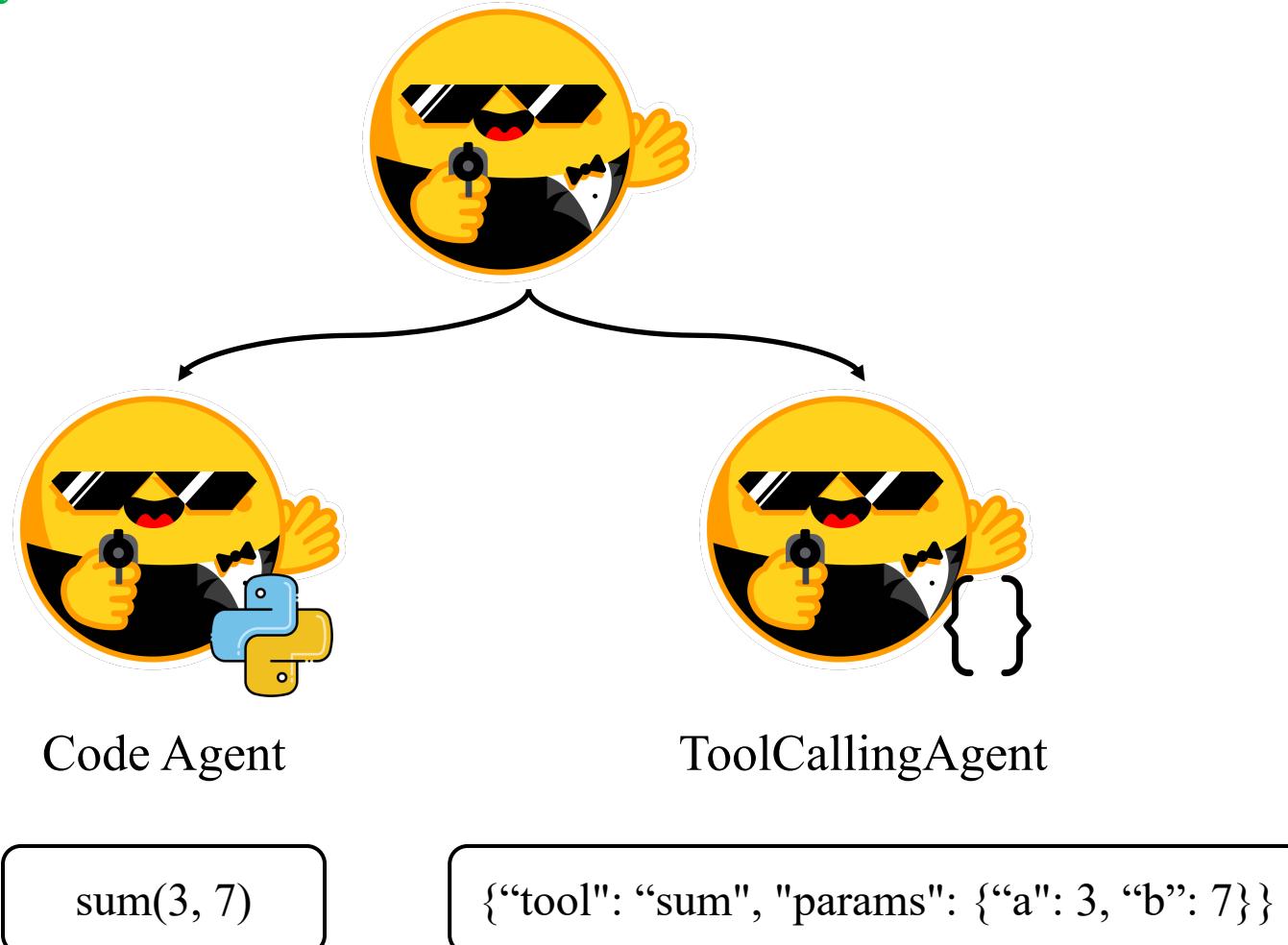
Code:

```
```py
def fibonacci(n, memo={}):
    if n in memo:
        return memo[n]
    if n <= 1:
        return n
    memo[n] = fibonacci(n - 1, memo) + fibonacci(n - 2, memo)
    return memo[n]

fib_50 = fibonacci(50)
final_answer(fib_50)
```
```

# Implement Agents in smolagents

## ❖ Agents in smolagent

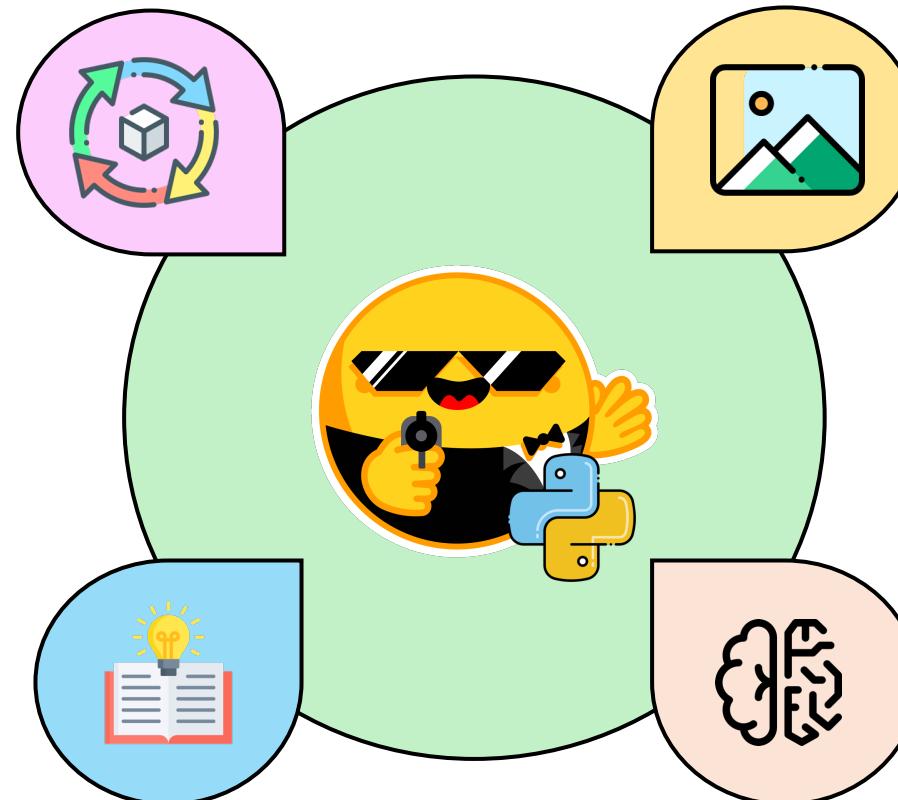


# Implement Agents in smolagents

## ❖ Code Agent

### Composability

Easily combine & reuse actions



### Generality

Express any computationally possible task

### Object Management

Work directly with complex structures like images

### Natural for LLMs

High-quality code is already present in LLM training data

# Implement Agents in smolagents

## ❖ Code Agent

**Instruction:** Determine the most cost-effective country to purchase the smartphone model "CodeAct 1". The countries to consider are the USA, Japan, Germany, and India.

### Available APIs

- [1] lookup\_rates(country: str) -> (float, float)
- [2] convert\_and\_tax(price: float, exchange\_rate: float, tax\_rate: float) -> float
- [3] estimate\_final\_price(converted\_price: float, shipping\_cost: float) -> float
- [4] lookup\_phone\_price(model: str, country: str) -> float
- [5] estimate\_shipping\_cost(destination\_country: str) -> float

### LLM Agent using [Text/JSON] as Action

**Think** I should calculate the phone price in USD for each country, then find the most cost-effective country.

**Action** Text: `lookup_rates, Germany`  
JSON: `{"tool": "lookup_rates", "country": "Germany"}`

**Environment** 1.1, 0.19

**Action** Text: `lookup_phone_price, CodeAct 1, Germany`  
JSON: `{"tool": "lookup_phone_price", "model": "CodeAct 1", "country": "Germany"}`

**Environment** 700

**Action** Text: `convert_and_tax, 700, 1.1, 0.19`  
JSON: `{"tool": "convert_and_tax", "price": 700, "exchange_rate": 1.1, "tax_rate": 0.19}`

**Environment** 916.3

[... interactions omitted (look up shipping cost and calculate final price) ...]

**Action** Text: `lookup_rates, Japan`  
JSON: `{"tool": "lookup_rates", "country": "Japan"}`

[... interactions omitted (calculate final price for all other countries)...]

**Response** The most cost-effective country to purchase the smartphone model is Japan with price 904.00 in USD.

Fewer Actions Required!

### CodeAct: LLM Agent using [Code] as Action

**Think** I should calculate the phone price in USD for each country, then find the most cost-effective country.

**Action**

```
countries = ['USA', 'Japan', 'Germany', 'India']
final_prices = {}

for country in countries:
    exchange_rate, tax_rate = lookup_rates(country)
    local_price = lookup_phone_price("xAct 1", country)
    converted_price = convert_and_tax(
        local_price, exchange_rate, tax_rate
    )
    shipping_cost = estimate_shipping_cost(country)
    final_price = estimate_final_price(converted_price, shipping_cost)
    final_prices[country] = final_price

most_cost_effective_country = min(final_prices, key=final_prices.get)
most_cost_effective_price = final_prices[most_cost_effective_country]
print(most_cost_effective_country, most_cost_effective_price)
```

Control & Data Flow of Code  
Simplifies Complex Operations

Re-use `min` Function from Existing  
Software Infrastructures (Python library)

**Environment** 1.1, 0.19

**Response** The most cost-effective country to purchase the smartphone model is Japan with price 904.00 in USD.



# Implement Agents in smolagents

## ❖ Code Agent

```
1 from smolagents import CodeAgent, DuckDuckGoSearchTool, TransformersModel
2
3 model = TransformersModel(
4     model_id="Qwen/Qwen2.5-Coder-7B-Instruct",
5     torch_dtype=torch.bfloat16,
6     trust_remote_code=True,
7     max_new_tokens=3000
8 )
9
10 agent = CodeAgent(
11     tools=[],
12     model=model
13 )
14
15 agent.run("Số Fibonacci thứ 50 là bao nhiêu?", max_steps=5)
```

### Generated code



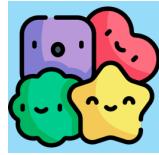
```
def fibonacci(n):
    if n <= 0:
        return 0
    elif n == 1:
        return 1
    else:
        a, b = 0, 1
        for _ in range(2, n + 1):
            a, b = b, a + b
        return b

result = fibonacci(50)
final_answer(result)
```



# Implement Agents in smolagents

## ❖ Introduction



Support for any LLM

**smolagents** supports models hosted on the Hub loaded in their transformers version, or through Inference providers. It also supports models from OpenAI, Anthropic...



**LiteLLM**



**OpenAI**



# Implement Agents in smolagents

## ❖ Introduction



Hub integrations

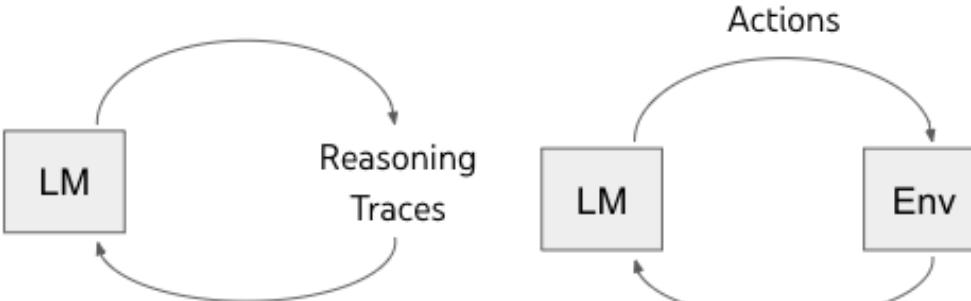
Seamless integration with the **Hugging Face Hub**, allowing the use of Gradio Spaces as tools.

The screenshot shows the "smolagents and tools gallery" interface. At the top, there is a search bar, a "type" dropdown set to "Tools", and a "sort by" dropdown set to "Most Likes". Below the header, there are four tool cards:

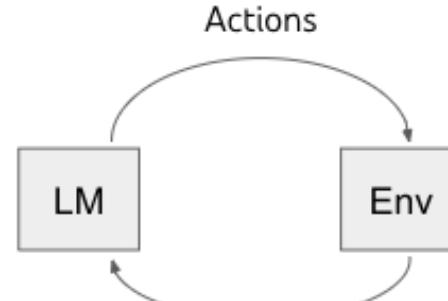
- agents-course/First\_agent\_template**: A template agent card with a placeholder "Agent" section and a "Chat Message" input field at the bottom. It includes a "Built with Gradio" and "Settings" link at the very bottom.
- acidtib/Travel-Planning-Agent**: A travel planning agent card titled "Travel Planning Agent". It describes its function: "I can help you plan your trips, find the best time to travel, and provide information about destinations. Ask me about travel costs, best seasons to visit, or specific routes!". It also has a "Agent" section and a "Chat Message" input field.
- agents-course/text-to-image**: A text-to-image tool card titled "TextToImageTool". It features a "prompt" input field, a "Clear" button, and an orange "Submit" button. Below it is a preview area showing a generated image thumbnail.
- m-ric/text-to-image**: Another text-to-image tool card titled "TextToImageTool", identical in structure to the one above it, with its own "prompt" input field, "Clear" button, orange "Submit" button, and preview area.

# Implement Agents in smolagents

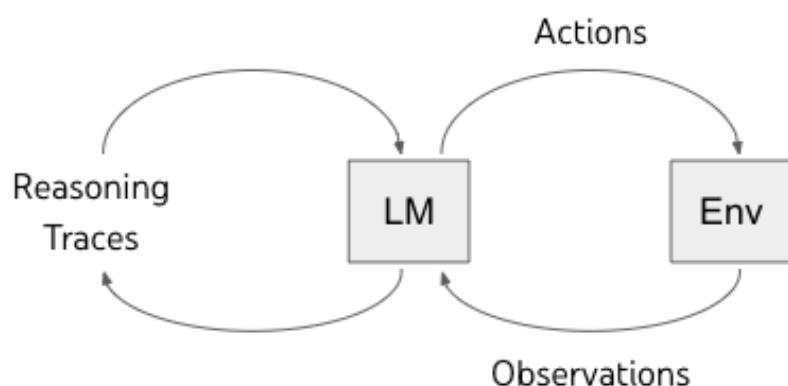
## ❖ ReAct Framework



Reason Only



Act Only

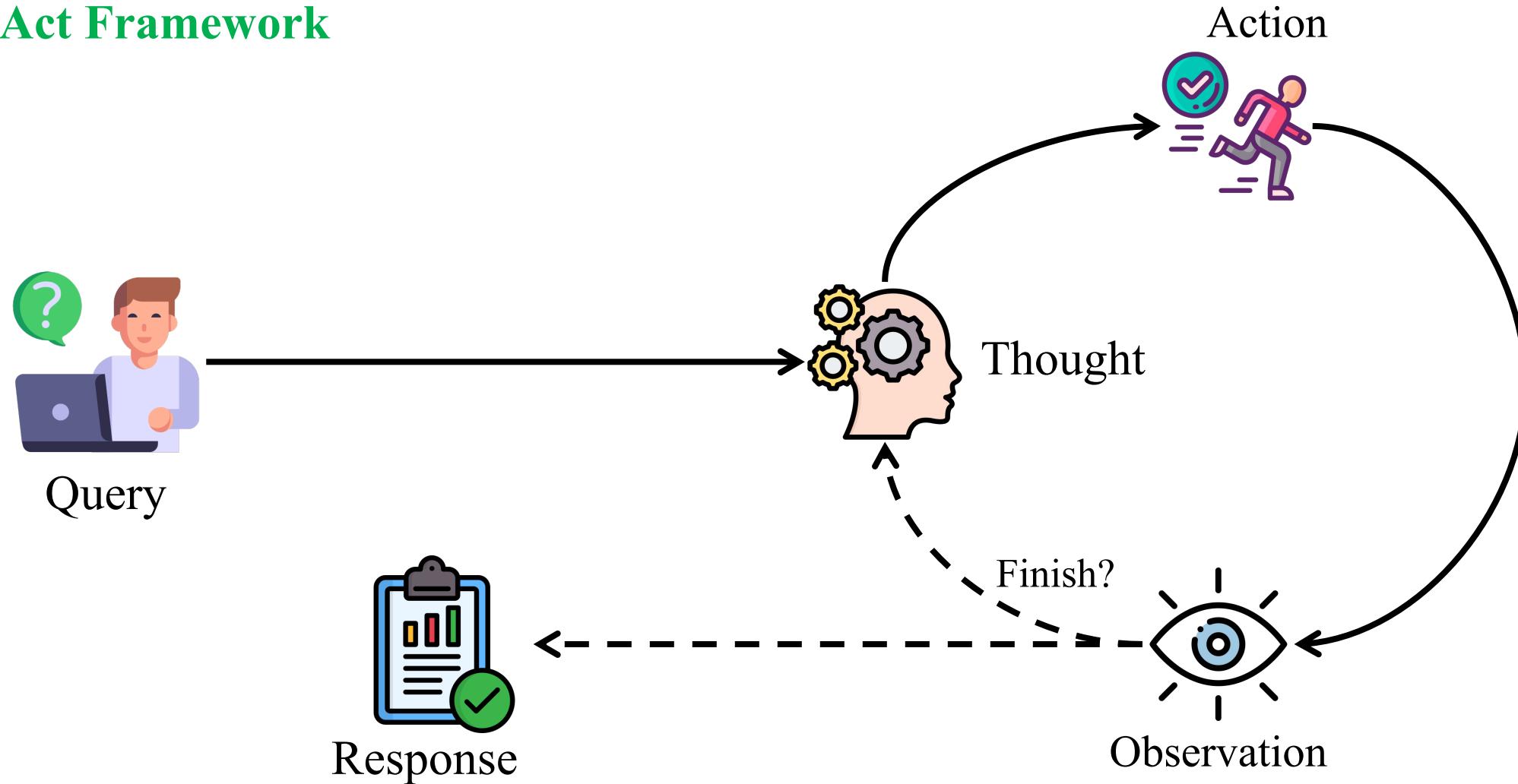


ReAct (Reason + Act)

**ReAct** (Reason + Act) is a prompting paradigm for LLM that interleaves explicit “thought” (reasoning traces) and “action” steps in a single workflow, enabling the model to plan dynamically and interact with external tools or environments.

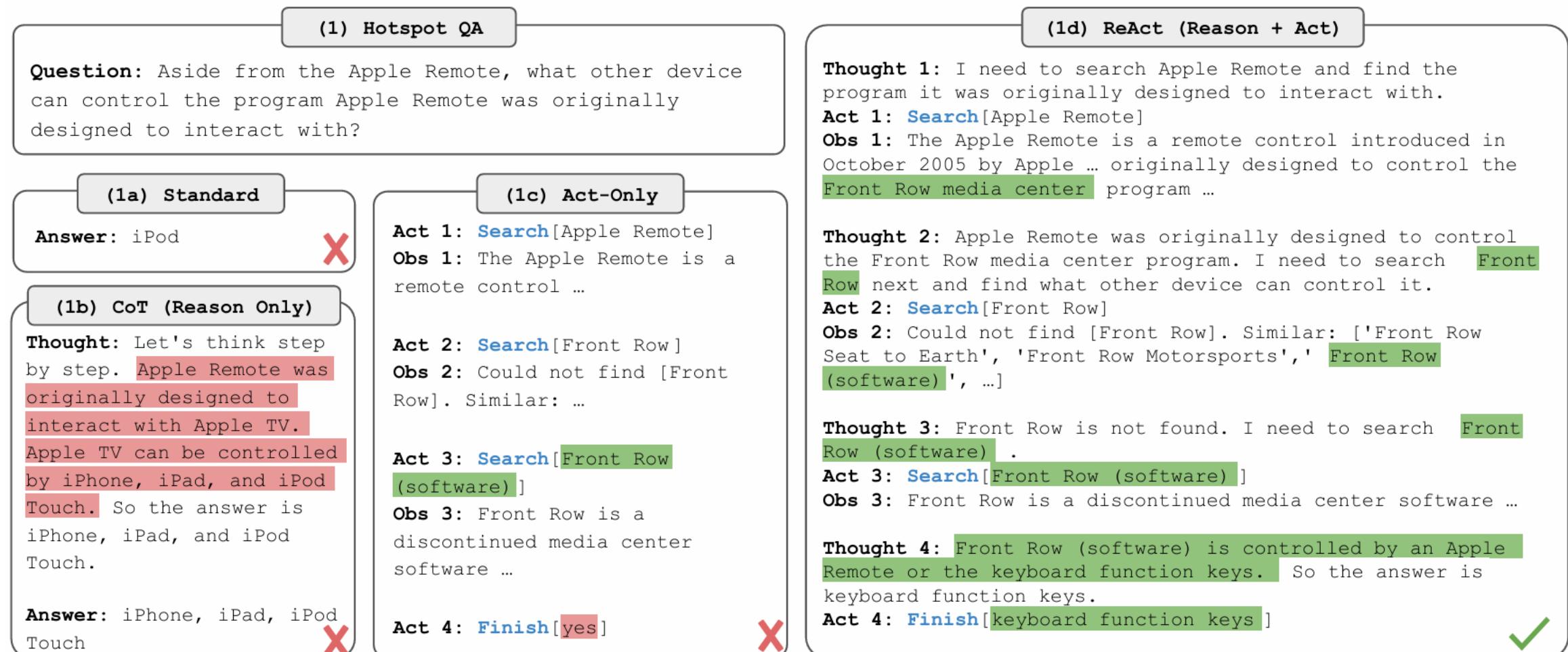
# Implement Agents in smolagents

## ❖ ReAct Framework



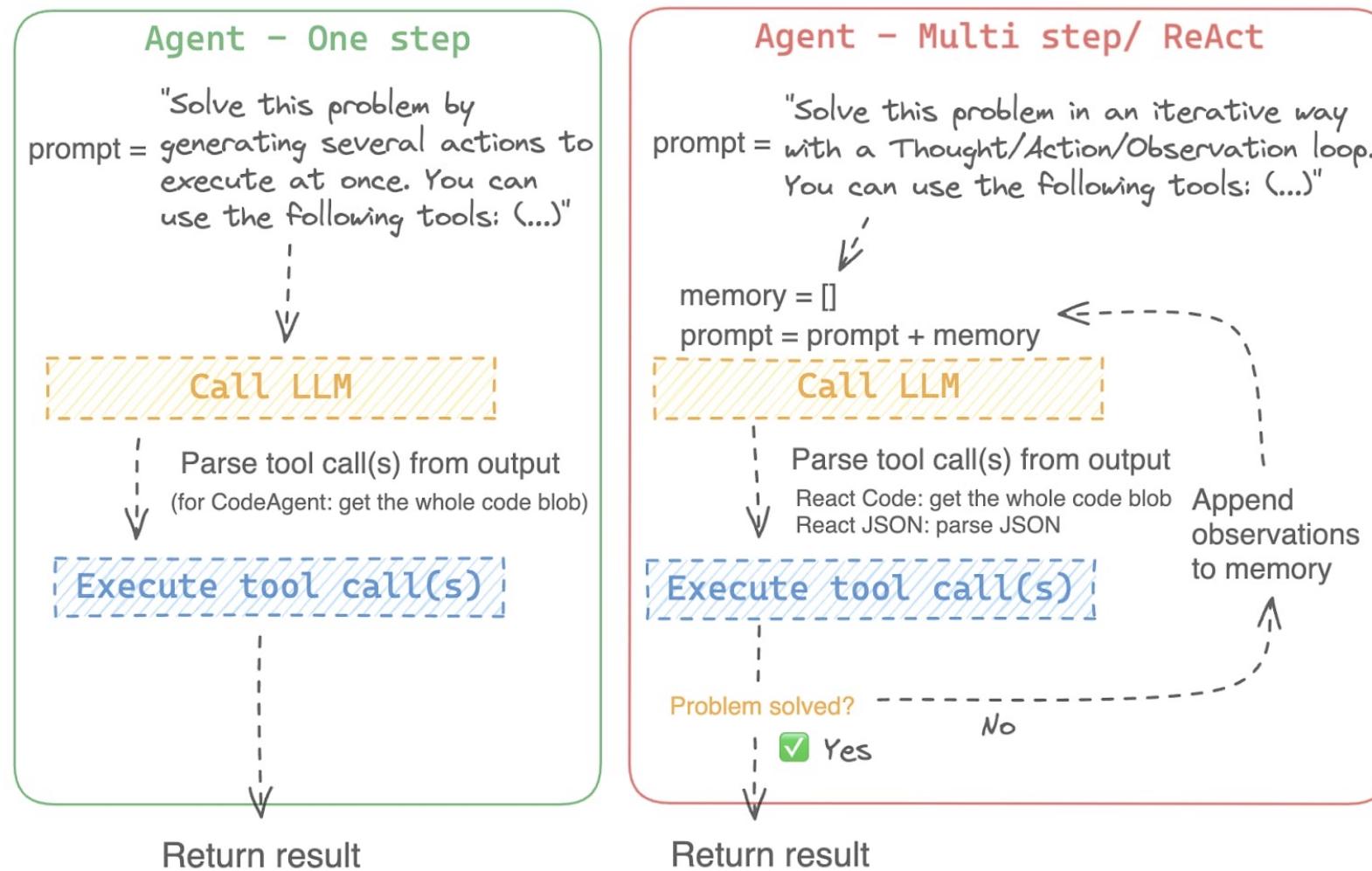
# Implement Agents in smolagents

## ❖ ReAct Framework



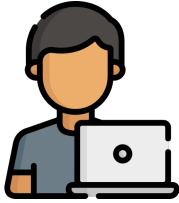
# Implement Agents in smolagents

## ❖ ReAct Framework



# Implement Agents in smolagents

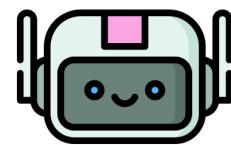
## ❖ ReAct Framework



What's the current temperature in New York City?

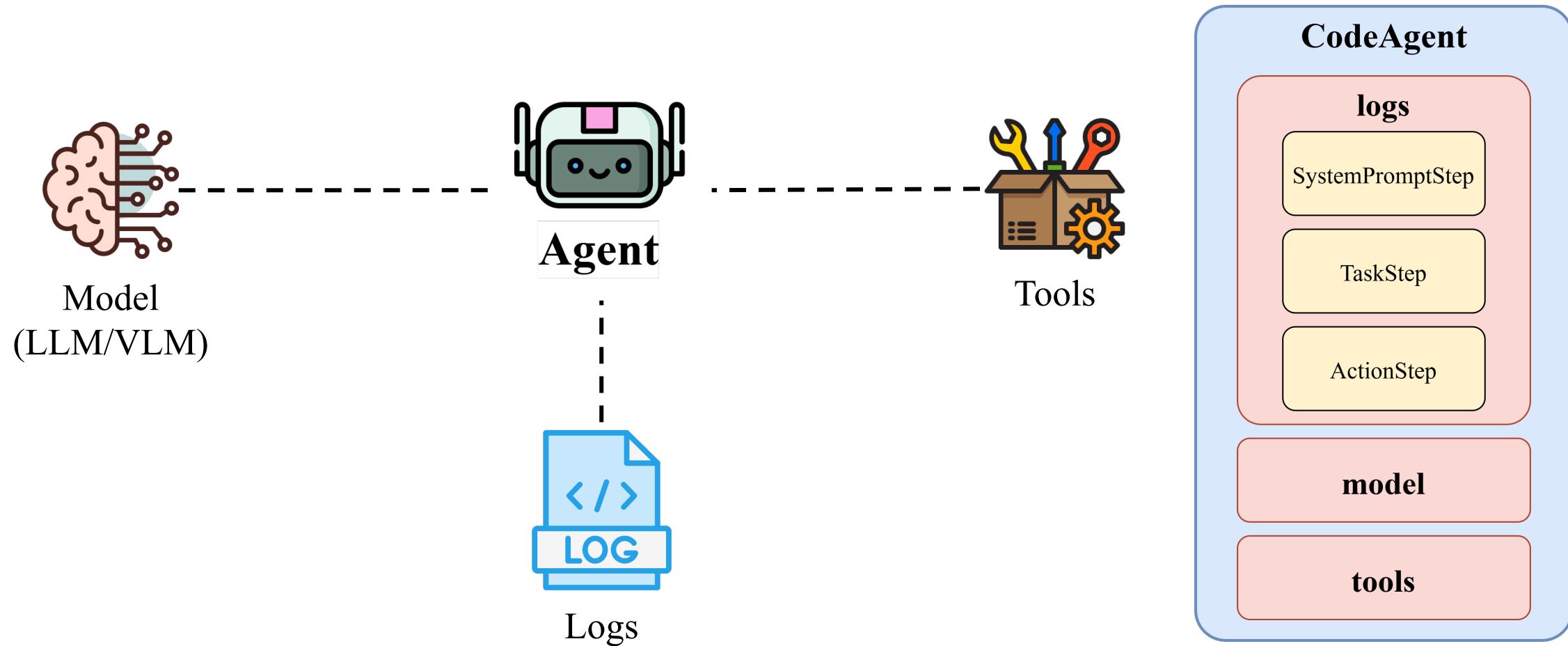
- **Thought:** I need to find the latest weather data for New York City to answer accurately.
- **Action:** Call a weather API  
GET [https://api.weather.com/...](https://api.weather.com/)
- **Observation:** API returns:

```
{  
  "temperature": 22,  
  "units": "C",  
  "condition": "Partly Cloudy"  
}
```
- **Final Answer:** The current temperature in New York City is 22 °C with partly cloudy skies.



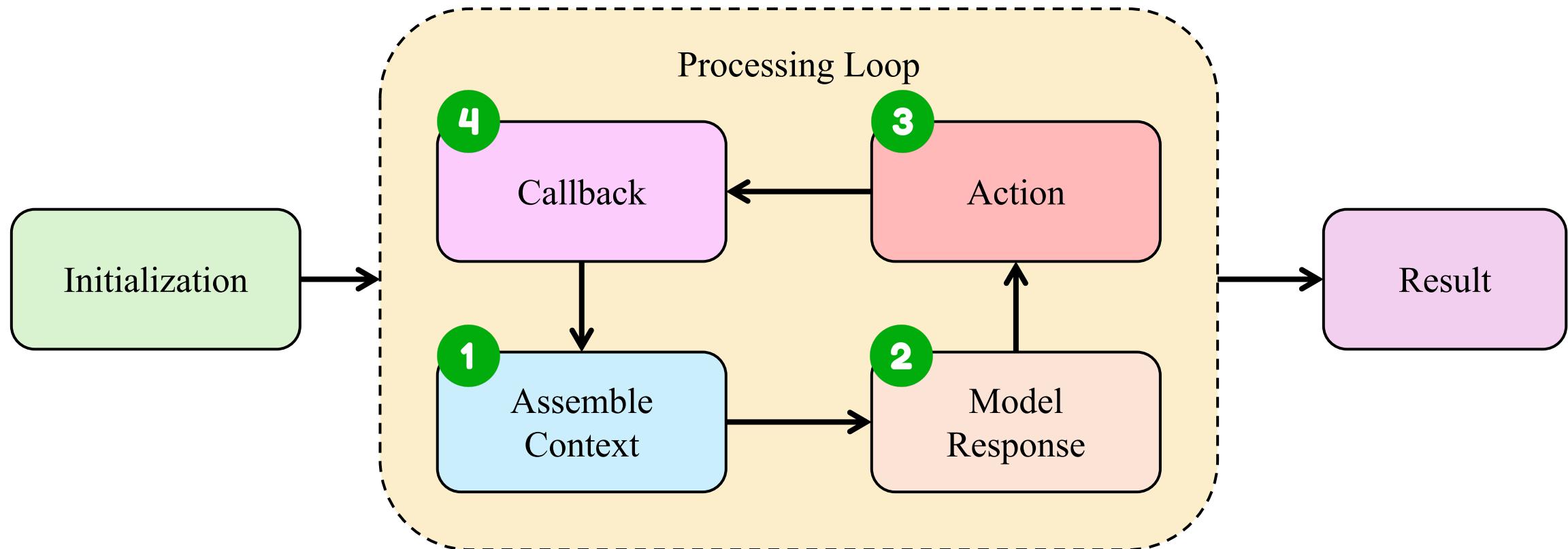
# Implement Agents in smolagents

## ❖ Agent in smolagents



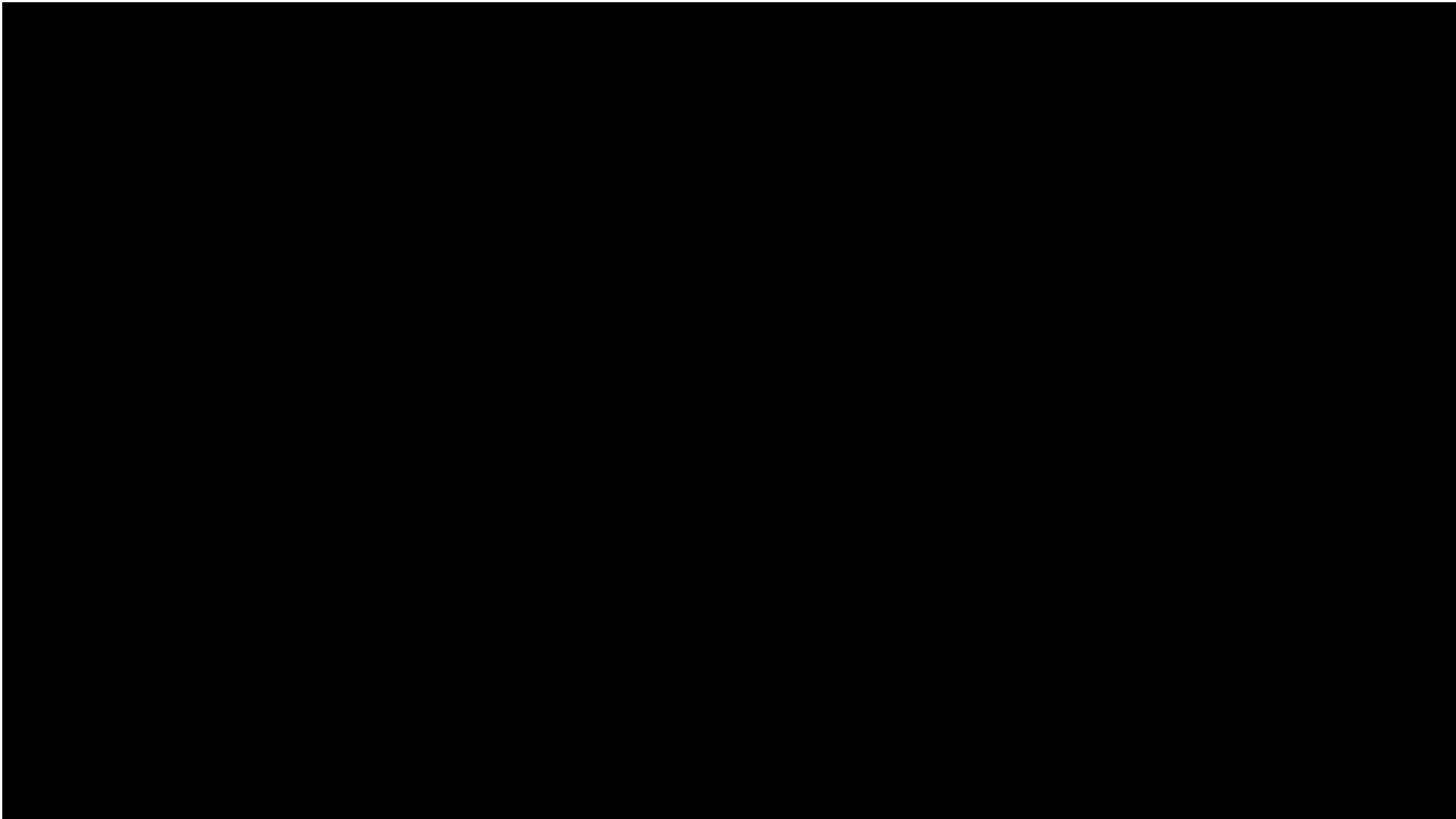
# Implement Agents in smolagents

## ❖ Agent in smolagents



# Implement Agents in smolagents

## ❖ Agent in smolagents



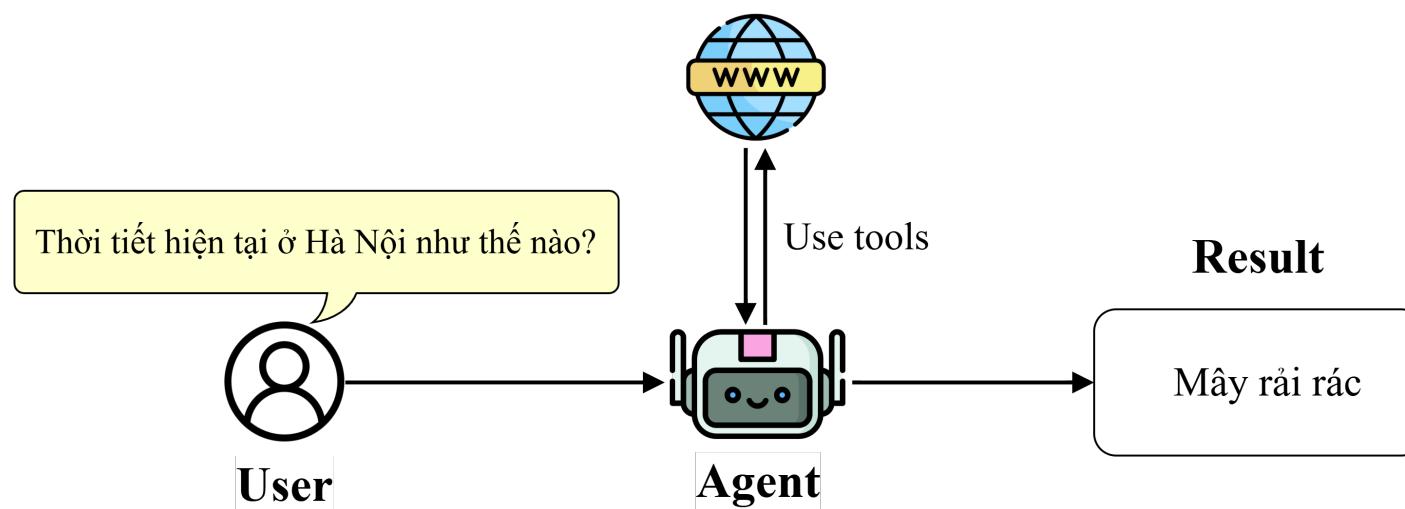
# Implement Agents in smolagents

## ❖ Example: Weather Agent

```
from smolagents import CodeAgent, HfApiModel, FinalAnswerTool, tool

agent = CodeAgent(
    tools=[get_coordinates, get_current_weather, FinalAnswerTool()],
    model=HfApiModel(),
    additional_authorized_imports=["requests"],
    name="get_weather_agent",
)

agent.run("Hãy cho tôi biết thời tiết hiện tại ở Hà Nội.")
```





# Implement Agents in smolagents

## ❖ Example: Weather Agent

```
● ● ●

@tool
def get_coordinates(city: str) -> tuple[float, float]:
    """
    Retrieves the geographical coordinates (latitude and longitude) of a city.

    Args:
        city (str): The name of the city to retrieve coordinates for.

    Returns:
        tuple[float, float]: The latitude and longitude of the city, or None if the coordinates could
    not be retrieved.
    """
    import requests
    API_KEY = "your_api_key"
    api_url = f"http://api.openweathermap.org/geo/1.0/direct?q={city}&limit=1&appid={API_KEY}"

    response = requests.get(api_url)
    response.raise_for_status()
    data = response.json()
    if data:
        return data[0]["lat"], data[0]["lon"]
    else:
        return None
```

# Implement Agents in smolagents

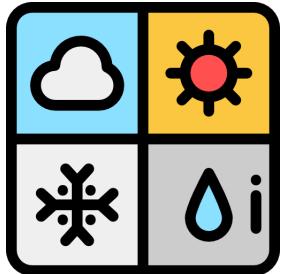
## ❖ Example: Weather Agent



```
@tool
def get_current_weather(lat: float, lon: float) -> str:
    """
    Retrieves the current weather description for a specific geographical location.

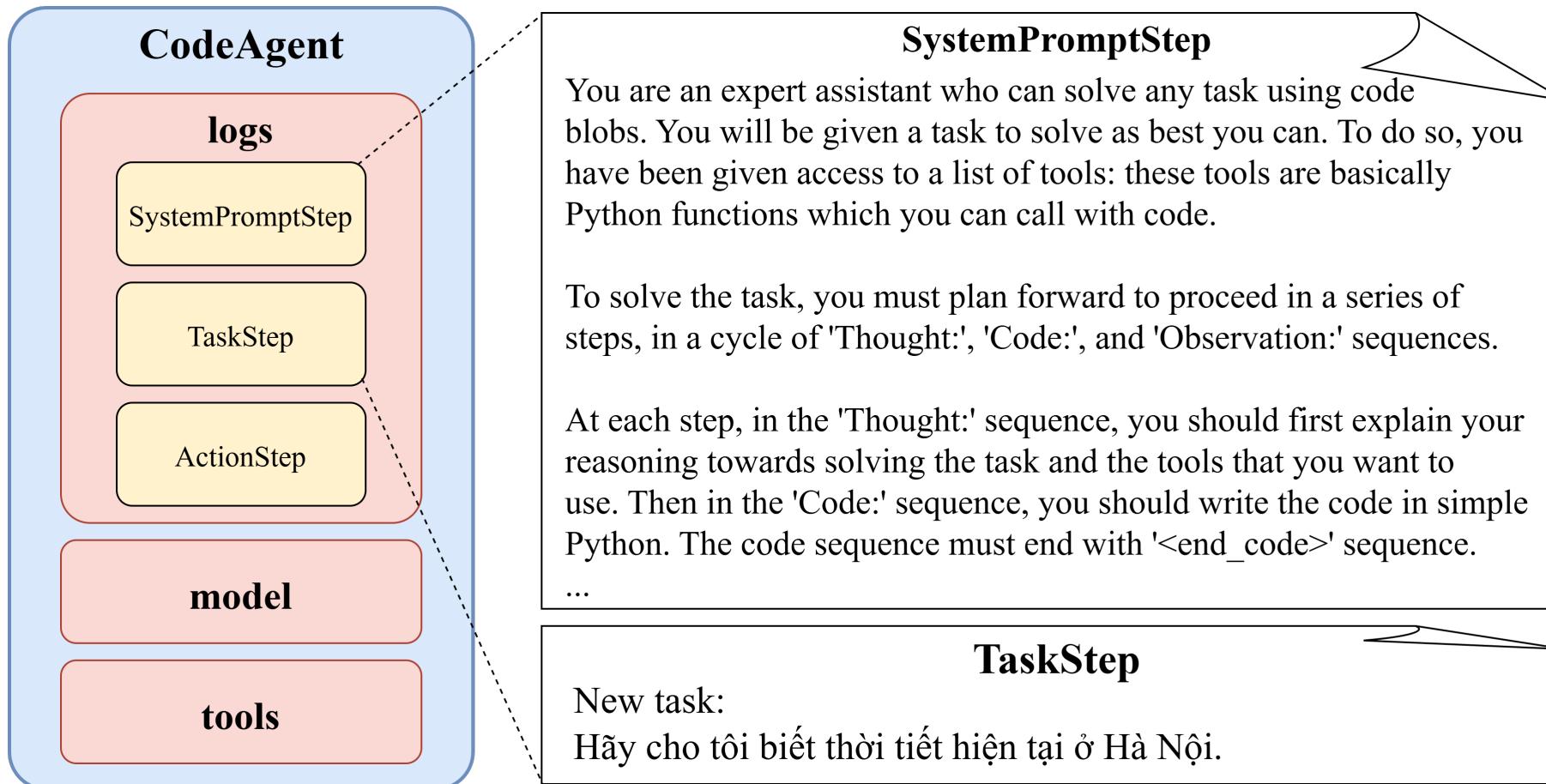
    Args:
        lat: The latitude of the location.
        lon: The longitude of the location.

    Returns:
        str: The current weather description for the location, or None if the weather information could
        not be retrieved.
    """
    import requests
    API_KEY = "your_api_key"
    api_url = f"https://api.openweathermap.org/data/2.5/weather?lat={lat}&lon={lon}&appid={API_KEY}"
    response = requests.get(api_url)
    response.raise_for_status()
    data = response.json()
    return data["weather"][0]["description"]
```



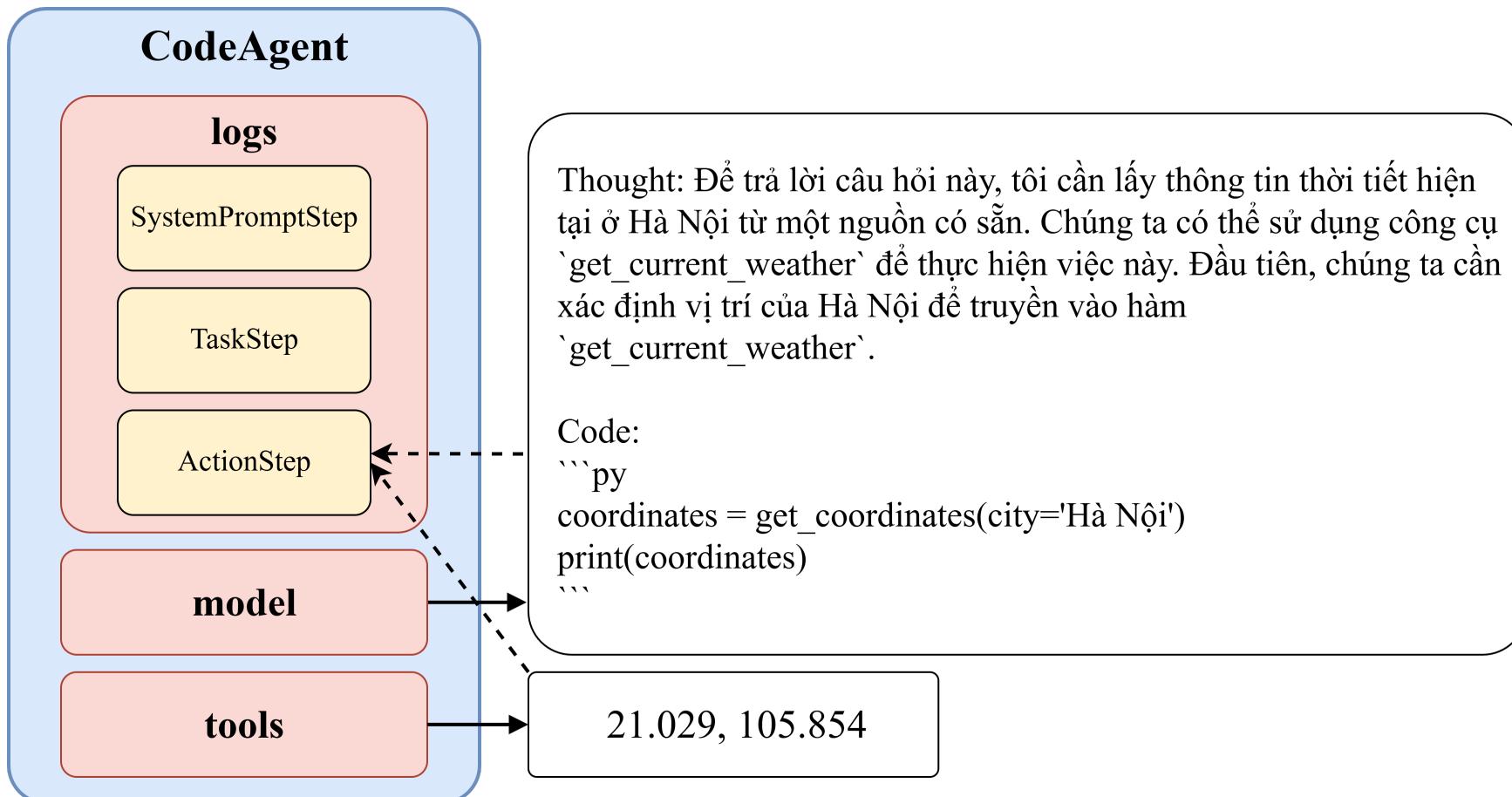
# Implement Agents in smolagents

## ❖ Weather Agent: Init



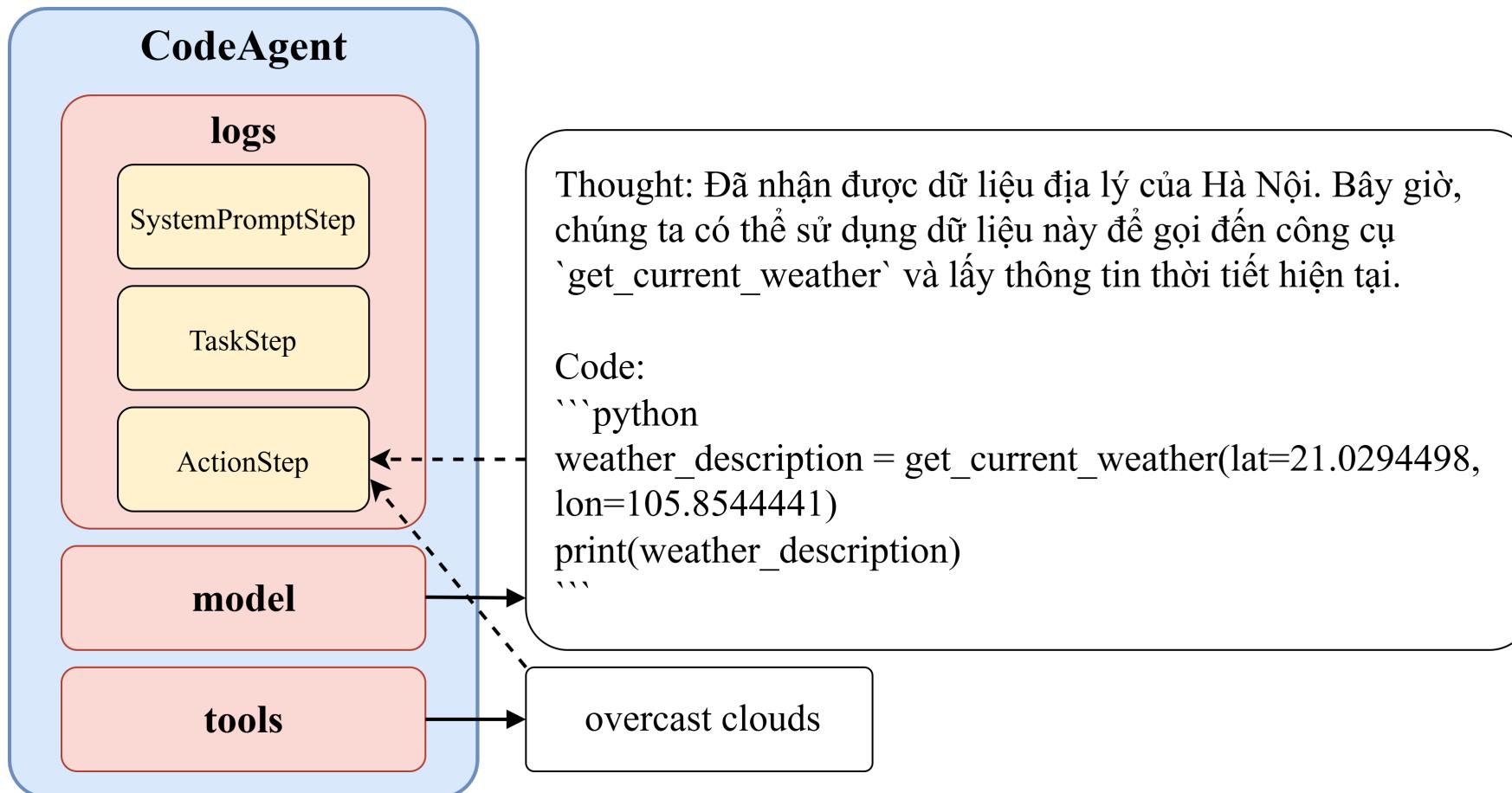
# Implement Agents in smolagents

## ❖ Weather Agent: Loop 1



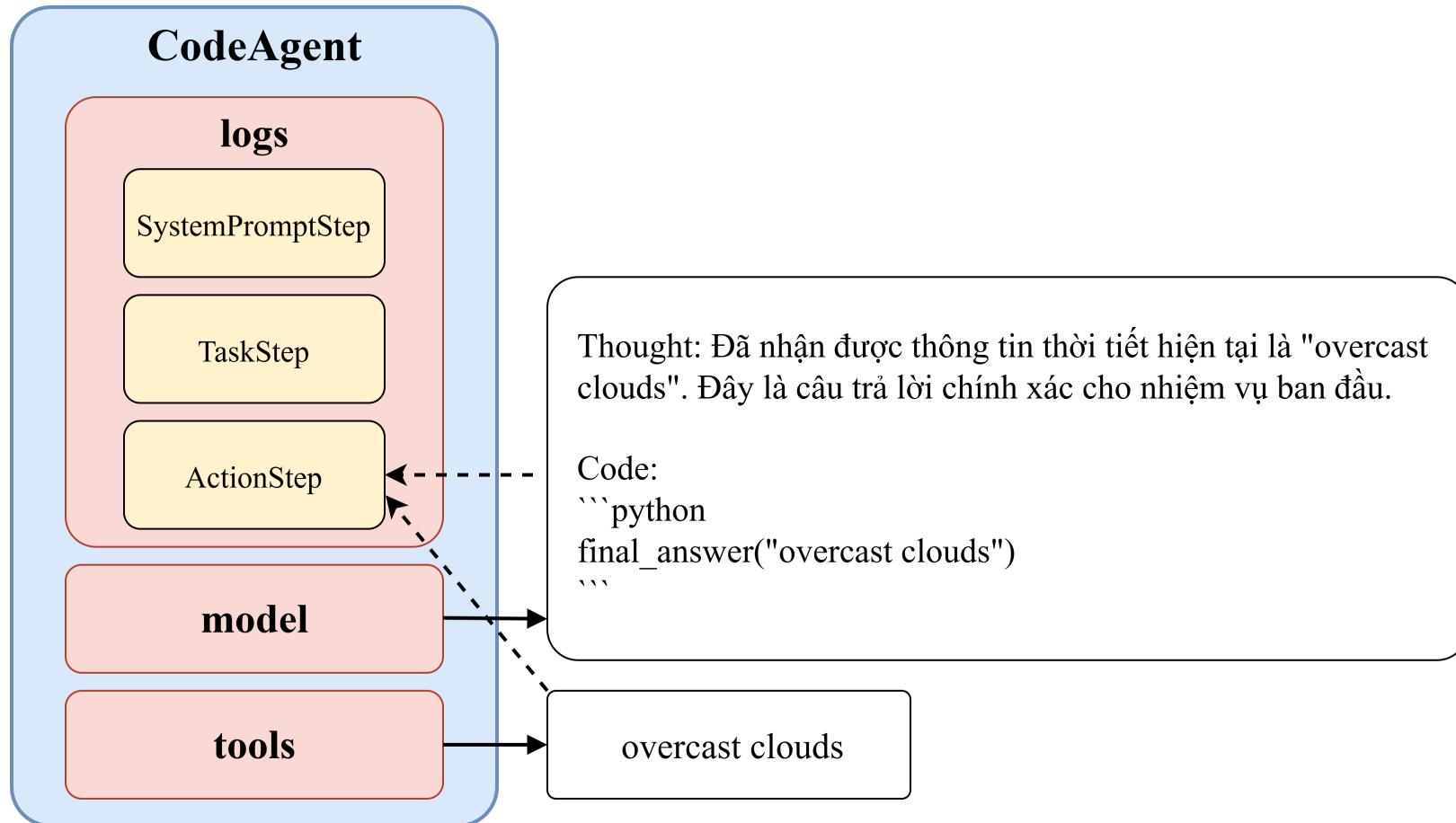
# Implement Agents in smolagents

## ❖ Weather Agent: Loop 2



# Implement Agents in smolagents

## ❖ Weather Agent: Loop 3



# Implement Agents in smolagents

## ❖ Weather Agent: Result

```
1 task = "Hãy cho tôi biết thời tiết hiện tại ở Hà Nội."  
2 agent.run(task, max_steps=5)
```

New run - get\_weather\_agent

Hãy cho tôi biết thời tiết hiện tại ở Hà Nội.

TransformersModel - Qwen/Qwen2.5-Coder-3B-Instruct

Step 1

Output message of the LLM:

Thought: Để trả lời câu hỏi này, tôi cần lấy thông tin thời tiết hiện tại ở Hà Nội từ một nguồn có sẵn. Chúng ta có thể sử dụng công cụ `get\_current\_weather` để thực hiện việc này. Đầu tiên, chúng ta cần xác định vị trí của Hà Nội để truyền vào hàm `get\_current\_weather`.

Code:

```
```py  
coordinates = get_coordinates(city='Hà Nội')  
print(coordinates)  
```<end_code>
```

Observation:

Latitude: 21.0283

Longitude: 105.8524

- Executing parsed code:

```
coordinates = get_coordinates(city='Hà Nội')  
print(coordinates)
```

Execution logs:

(21.0294498, 105.8544441)

# Implement Agents in smolagents

## ❖ Weather Agent: Result

Step 2

*Output message of the LLM:*

Thought: Đã nhận được dữ liệu địa lý của Hà Nội. Bây giờ, chúng ta có thể sử dụng dữ liệu này để gọi đến công cụ `get\_current\_weather` và lấy thông tin thời tiết hiện tại.

Code:

```
```py
weather_description = get_current_weather(lat=21.0294498, lon=105.8544441)
print(weather_description)
```<end_code>
```

Observation:

Executing code snippet...

Weather description: Sunny

– Executing parsed code:

```
weather_description = get_current_weather(lat=21.0294498, lon=105.8544441)
print(weather_description)
```

Execution logs:

scattered clouds

# Implement Agents in smolagents

## ❖ Weather Agent: Result

Step 3

*Output message of the LLM:*

Thought: Đã nhận được thông tin thời tiết hiện tại là "scattered clouds". Đây là câu trả lời chính xác cho câu hỏi "Thời tiết hiện tại ở Hà Nội là gì? ".

Code:

```
```py
final_answer("scattered clouds")
```<end_code>
```

Observation:

Executing code snippet...

Final answer: scattered cloudsCalling tools:

```
[{'id': 'call_3', 'type': 'function', 'function': {'name': 'python_interpreter', 'arguments':
'final_answer("scattered clouds")'}}]
```

– Executing parsed code:

```
final_answer("scattered clouds")
```

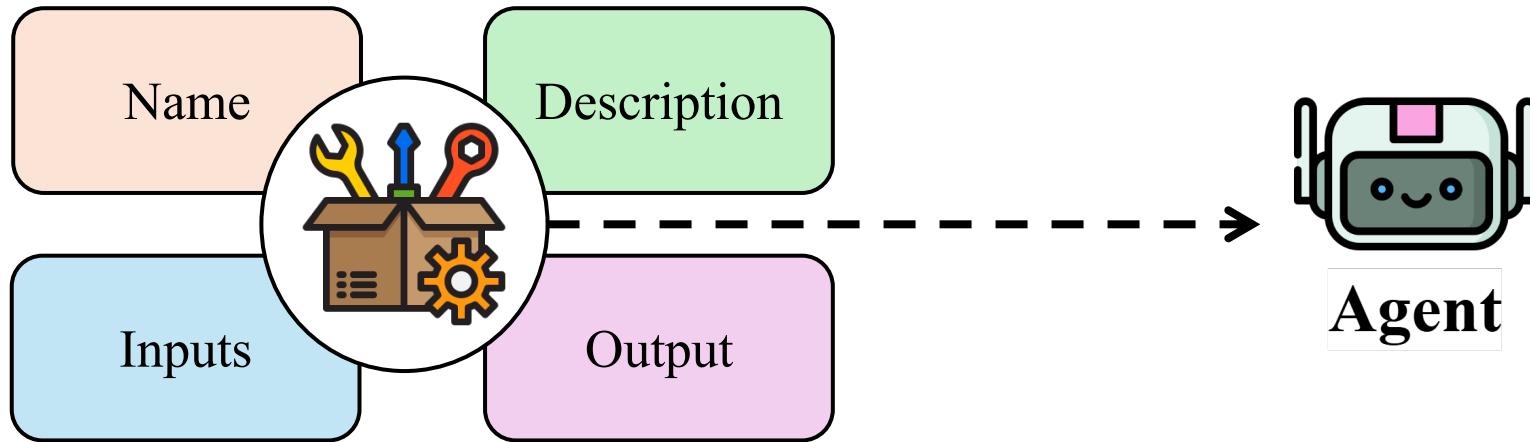
**Out - Final answer: scattered clouds**

[Step 3: Duration 14.90 seconds | Input tokens: 7,093 | Output tokens: 348]

'scattered clouds'

# Implement Agents in smolagents

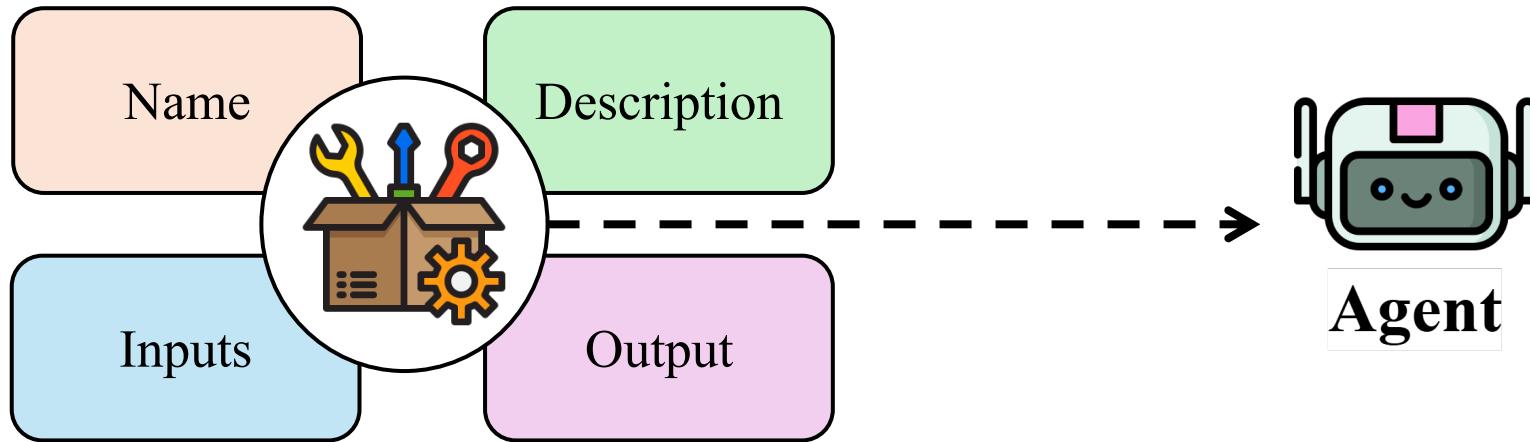
## ❖ Tools



**Tools** are external functions—implemented as standardized Python routines in the smolagents library—that the agent can invoke to extend its capabilities. Each tool is defined by a unique **name**, a brief **description**, a specification of its **input** parameters (including names, data types, and roles), and the expected **output** data type, enabling the model to accurately understand and integrate the tool.

# Implement Agents in smolagents

## ❖ Tools



- **Name:** web\_search.
- **Description:** Searches the internet for information based on a keyword.
- **Input:** query (str): the keyword to search for.
- **Output:** str: the search results as text.

# Implement Agents in smolagents

## ❖ Docstring

```
1 def greet(name):
2     """
3         Return a greeting message for the given name.
4
5     Args:
6         name (str): The name of the person to greet.
7
8     Returns:
9         str: A personalized greeting.
10    """
11    return f"Hello, {name}!"
```

A **docstring** is a string literal placed as the very first statement in a module, class, function, or method definition—stored in the object's `__doc__` attribute and accessible at runtime for introspection and automated documentation.

# Implement Agents in smolagents

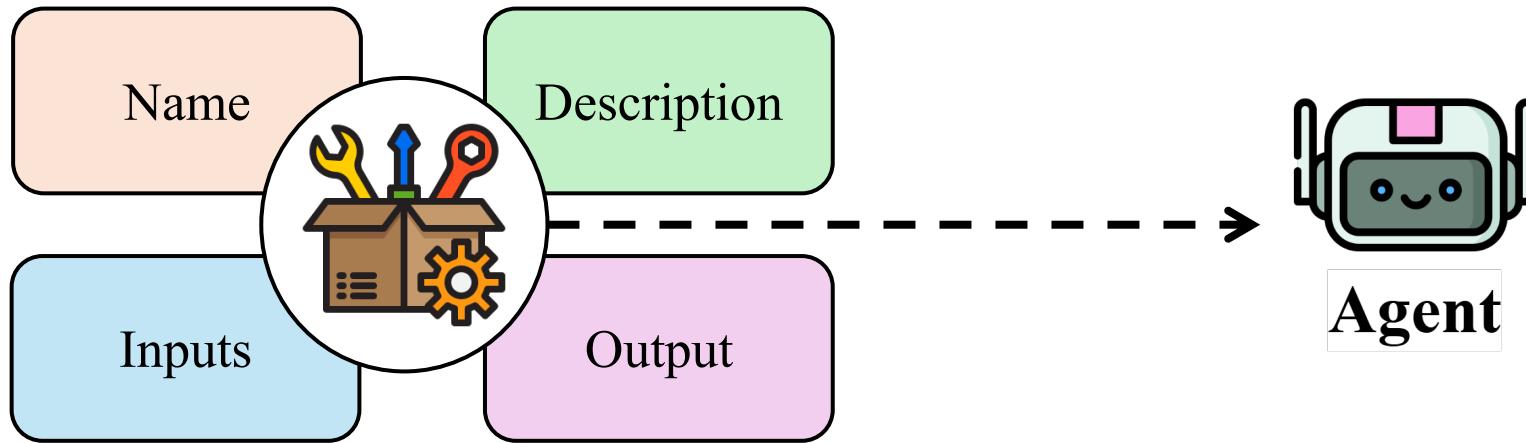
## ❖ Type Hint

```
1 def filter_even(numbers: list[int]) -> list[int]:  
2     return [n for n in numbers if n % 2 == 0]  
3  
4 result = filter_even([1, 2, 3, 4, 5, 6])  
5 print(result) # Output: [2, 4, 6]
```

A **type hint** is an optional PEP 484 annotation syntax for indicating the expected data types of variables, function parameters, and return values—ignored at runtime but used by static type checkers and IDEs to improve code readability and catch type errors early.

# Implement Agents in smolagents

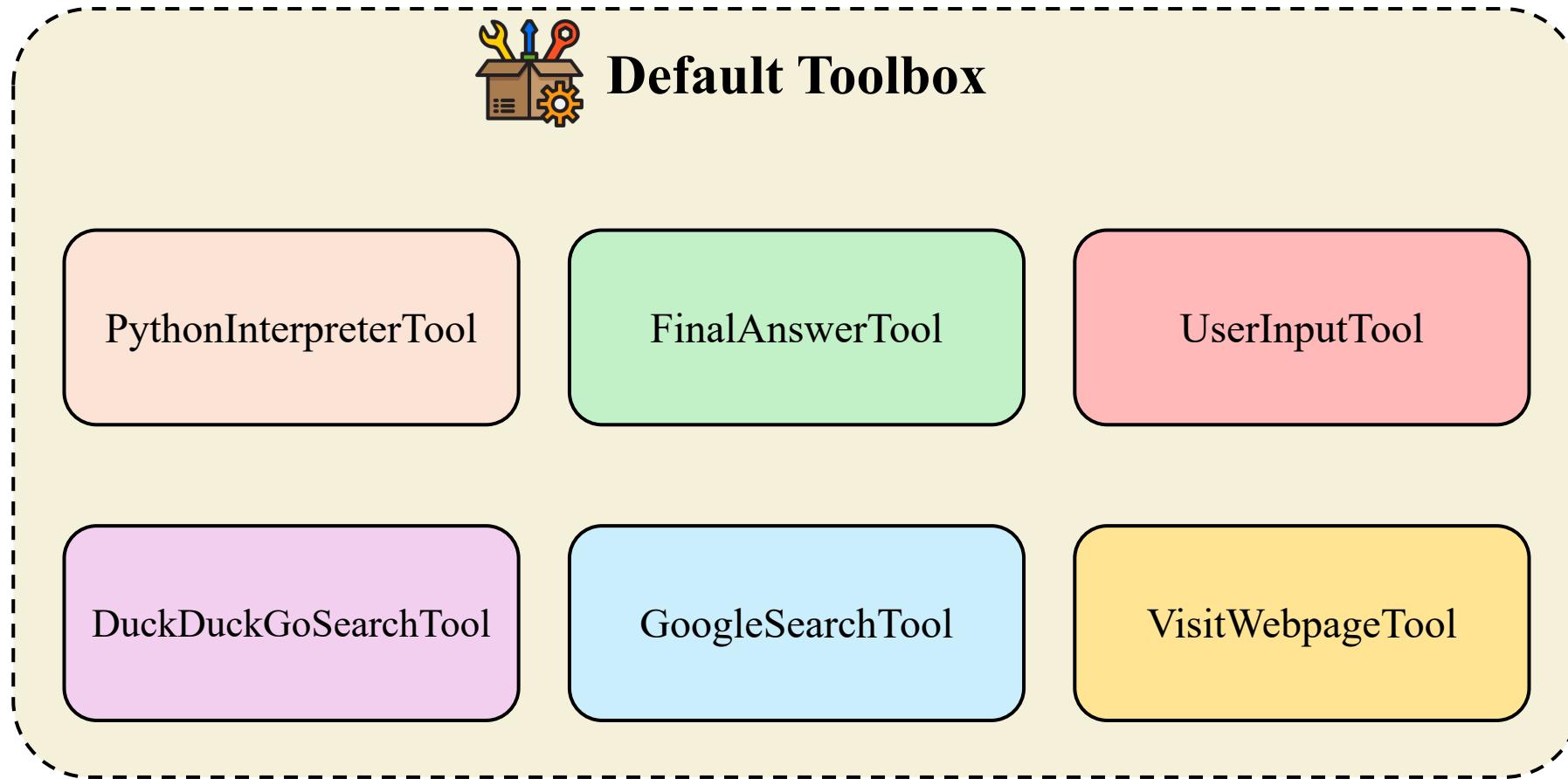
## ❖ Tools



```
1 def get_todays_date(date_format: str) -> str:  
2     """  
3         Returns today's date formatted according to the specified pattern.  
4     """  
5     Args:  
6         date_format (str): The desired date format string.  
7             Examples include "%Y-%m-%d", "%d", "%m", "%Y".  
8     Returns:  
9         str: Today's date as a string formatted with the given pattern.  
10    """  
11    from datetime import datetime  
12    return datetime.now().strftime(date_format)
```

# Implement Agents in smolagents

## ❖ Tools



# Implement Agents in smolagents

## ❖ Tools

Decorator

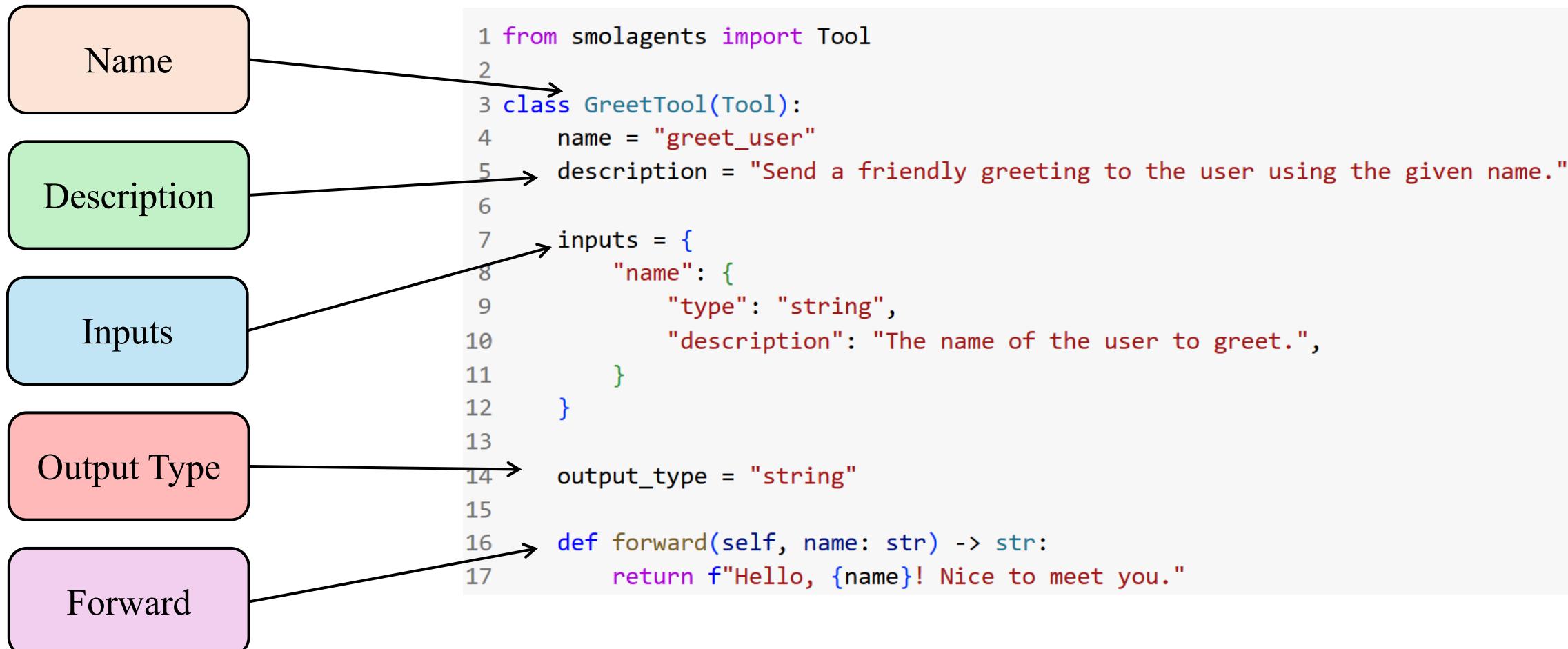
```
1 from smolagents import Tool
2
3 @tool
4 def get_todays_date(date_format: str) -> str:
5     """
6         Returns today's date formatted according to the specified pattern.
7
8     Args:
9         date_format (str): The desired date format string.
10            Examples include "%Y-%m-%d", "%d", "%m", "%Y".
11
12    Returns:
13        str: Today's date as a string formatted with the given pattern.
14    """
15    from datetime import datetime
16    return datetime.now().strftime(date_format)
```

Type hint

Docstring

# Implement Agents in smolagents

## ❖ Tools





**AI VIET NAM**  
@aivietnam.edu.vn

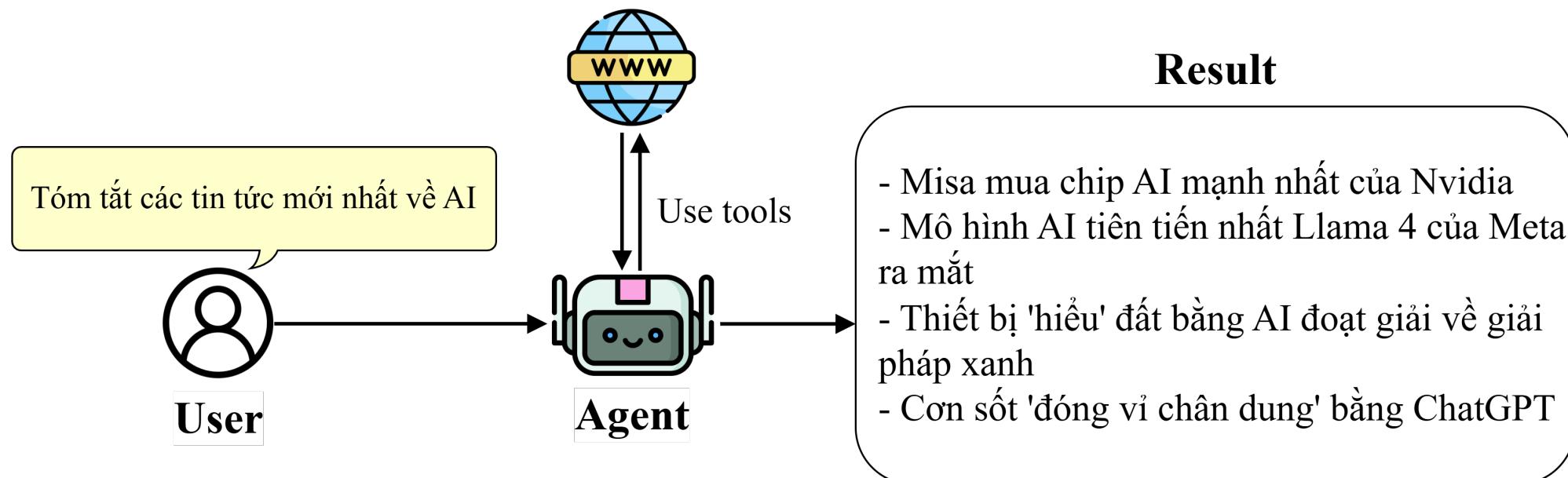
# QUIZ

# Summarize News with Agents

# Summarize News with Agents

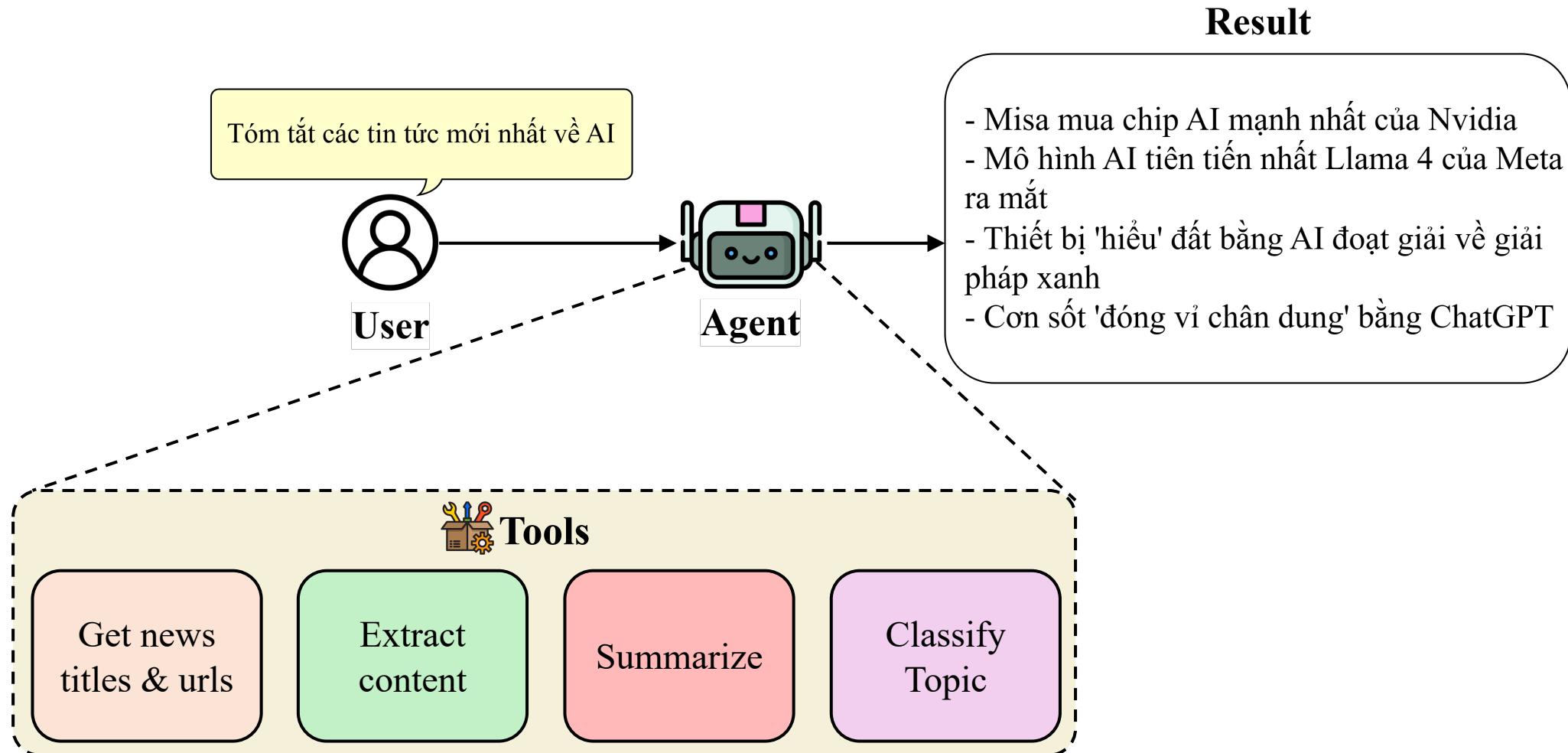
## ❖ Summarize news agent

**Description:** Serve an AI Agents application as an API that receive a command to summarize Vietnamese news and return the response of the AI Agent.



# Summarize News with Agents

## ❖ Summarize News Agent



# Summarize News with Agents

## ❖ Step 1. Install and import libraries

```
1 !pip install 'smolagents[transformers]'
```



# smolagents

```
1 from smolagents import TransformersModel, CodeAgent, tool, FinalAnswerTool
2 import os
3 import torch
4
5 def set_seed(seed: int):
6     torch.manual_seed(seed)
7     torch.cuda.manual_seed_all(seed)
8     torch.cuda.manual_seed(seed)
9     torch.backends.cudnn.deterministic = True
10    torch.backends.cudnn.benchmark = False
11
12 set_seed(42)
```

# Summarize News with Agents

## ❖ Step 2. Define tools for agent

Decorator

```
1 @tool
2 def fetch_latest_news_titles_and_urls(url: str) -> list[tuple[str, str]]:
3     """
4         This tool extracts the titles and URLs of the latest news articles from a news website's homepage.
5
6     Args:
7         url: The URL of the news website's homepage.
8
9     Returns:
10        list[tuple[str, str]]: A list of titles and URLs of the latest news articles.
11        """
```

Type hint

Docstring

# Summarize News with Agents

## ❖ Step 2. Define tools for agent

```
2 def fetch_latest_news_titles_and_urls(url: str) -> list[tuple[str, str]]:
12     import requests
13     from bs4 import BeautifulSoup
14
15     article_urls = []
16     article_titles = []
17     navigation_urls = []
18
19     response = requests.get(url)
20     soup = BeautifulSoup(response.text, 'html.parser')
21
22     navigation_bar = soup.find('nav', class_='main-nav')
23     if navigation_bar:
24         for header in navigation_bar.ul.find_all('li')[2:7]:
25             navigation_urls.append(url + header.a['href'])
26         for section_url in navigation_urls:
27             response = requests.get(section_url)
28             section_soup = BeautifulSoup(response.text, 'html.parser')
29             for article in section_soup.find_all('article')[:10]:
30                 title_tag = article.find('h3', class_='title-news')
31                 if title_tag:
32                     title = title_tag.text.strip()
33                     article_url = article.find('a')['href']
34                     article_titles.append(title)
35                     article_urls.append(article_url)
36
37     return list(zip(article_titles, article_urls))
```



# Summarize News with Agents

## ❖ Step 2. Define tools for agent

```
1 @tool
2 def extract_news_article_content(url: str) -> str:
3     """
4         This tool extracts the content of a news article from its URL.
5
6     Args:
7         url (str): The URL of the news article.
8
9     Returns:
10        str: The content of the news article.
11    """
12    import requests
13    from bs4 import BeautifulSoup
14
15    response = requests.get(url)
16    soup = BeautifulSoup(response.text, 'html.parser')
17    content = ""
18    for paragraph in soup.find_all('p'):
19        content += paragraph.get_text().strip() + " "
20    return content
```



# Summarize News with Agents

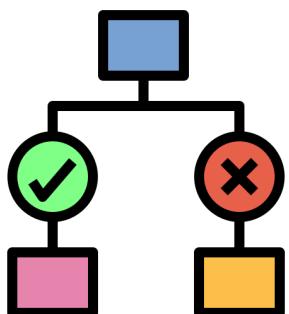
## ❖ Step 2. Define tools for agent



```
1 @tool
2 def summarize_news(text: str) -> str:
3     """
4         This tool summarizes the given Vietnamese news text.
5
6     Args:
7         text (str): The Vietnamese news text to be summarized.
8
9     Returns:
10        str: The summarized version of the input text.
11    """
12    from transformers import AutoTokenizer, AutoModelForSeq2SeqLM
13    import torch
14    device = "cuda" if torch.cuda.is_available() else "cpu"
15    model_name = "VietAI/vit5-base-vietnews-summarization"
16    tokenizer = AutoTokenizer.from_pretrained(model_name)
17    model = AutoModelForSeq2SeqLM.from_pretrained(model_name, torch_dtype=torch.bfloat16).cuda()
18
19    formatted_text = "vietnews: " + text + " </s>"
20    encoding = tokenizer(formatted_text, return_tensors="pt")
21    input_ids = encoding["input_ids"].to(device)
22    attention_masks = encoding["attention_mask"].to(device)
23
24    with torch.no_grad():
25        outputs = model.generate(input_ids=input_ids, attention_mask=attention_masks, max_length=256)
26
27    summary = tokenizer.decode(outputs[0], skip_special_tokens=True, clean_up_tokenization_spaces=True)
28    return summary
```

# Summarize News with Agents

## ❖ Step 2. Define tools for agent

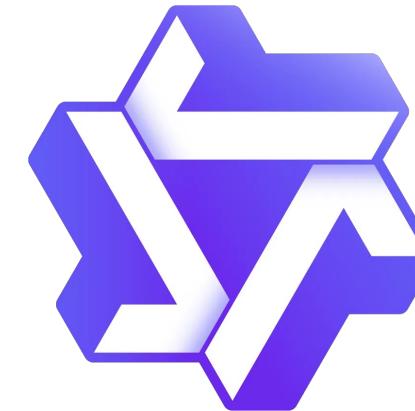


```
1 @tool
2 def classify_topic(text: str, topic: str) -> bool:
3     """
4         This tool classifies whether the given Vietnamese text is related to the specified topic.
5
6     Args:
7         text: The Vietnamese text to be classified.
8         topic: The string representing the topic to be checked.
9
10    Returns:
11        bool: True if the text is related to the topic; False otherwise.
12    """
13    from transformers import pipeline
14    import torch
15    device = "cuda" if torch.cuda.is_available() else "cpu"
16    classifier = pipeline(
17        "zero-shot-classification",
18        model="vicgalle/xlm-roberta-large-xnli-anli",
19        device=device,
20        trust_remote_code=True,
21    )
22
23    candidate_labels = [topic, f"không liên quan {topic}"]
24    result = classifier(text, candidate_labels)
25    predicted_label = result["labels"][0]
26
27    return predicted_label == topic
```

# Summarize News with Agents

## ❖ Step 3. Create an agent

```
1 model = TransformersModel(  
2     model_id="Qwen/Qwen2.5-Coder-3B-Instruct",  
3     torch_dtype=torch.bfloat16,  
4     trust_remote_code=True,  
5     max_new_tokens=2000  
6 )
```



```
8 agent = CodeAgent(  
9     model=model,  
10    tools=[fetch_latest_news_titles_and_urls,  
11          summarize_news,  
12          extract_news_article_content,  
13          classify_topic,  
14          FinalAnswerTool()],  
15    additional_authorized_imports=["requests", "bs4"],  
16    verbosity_level=2,  
17    name="news_agent",  
18    description="This agent is a smart news aggregator that fetches, summarizes, and classifies real-time news updates."  
19 )
```

# Summarize News with Agents

## ❖ Step 3. Create an agent

CodeAgent | Qwen/Qwen2.5-Coder-3B-Instruct

- [-]  Authorized imports: ['requests', 'bs4']
- [-]  Tools:

agent.visualize()

Name	Description	Arguments
fetch_lastest_news_titles_and_urls	This tool extracts the titles and URLs of the latest news articles from a news website's homepage.	url (`string`): The URL of the news website's homepage.
summarize_news	This tool summarizes the given Vietnamese news text.	text (`string`): The Vietnamese news text to be summarized.
extract_news_article_content	This tool extracts the content of a news article from its URL.	url (`string`): The URL of the news article.
classify_topic	This tool classifies whether the given Vietnamese text is related to the specified topic.	text (`string`): The Vietnamese text to be classified. topic (`string`): The string representing the topic to be checked.
final_answer	Provides a final answer to the given problem.	answer (`any`): The final answer to the problem



# Summarize News with Agents

## ❖ Push Agent to Hub

```
1 agent.push_to_hub('VLAIResearchLab/smolagents-vnexpress-news-codeagent')
```

The screenshot shows the AI VIET NAM News Summary Agent interface. At the top, it displays the AI logo, the name 'AI VIET NAM', and the handle '@aivietnam.edu.vn'. Below this, the title 'News Summary Agent' is shown along with sub-titles: 'AIO2024 Module 10', 'Real-time News Fetch & Summarization', and 'Topic Classification & Insight Extraction'. A large input field labeled 'Your request' contains the text 'Enter your prompt here and press Shift+Enter or the button'. A red 'Submit' button is positioned below the input field. In the main workspace, there is a step titled 'Step 1' with the sub-headline 'Thought: To summarize the latest AI news, I'll use the fetch\_lastest\_news\_titles\_and\_urls tool to get the titles and URLs of the latest news articles, and then use the summarize\_news tool to summarize each article.' It includes a 'Code:' section with the following Python code:

```
news_data = fetch_lastest_news_titles_and_urls(url="https://www.techcrunch.com/")
print(news_data)
```

Below the code, there are two dropdown menus: 'Used tool python\_interpreter' and 'Execution Logs'. A green callout bubble to the right of the workspace contains the text 'tổng hợp tin tức AI mới nhất hôm nay'.

# Summarization and Q&A

# Summarization and Q&A



Curl

```
curl -X 'POST' \
  'http://0.0.0.0:5000/generative_ai' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
    "question": "what is BERT?"
}'
```

Request URL

```
http://0.0.0.0:5000/generative_ai
```

Server response

Code	Details
200	Response body

```
{ "answer": "BERT is a bidirectional transformer model for natural language processing pre-trained on a large corpus of text. It stands for Bidirectional Encoder Representations from Transformers. BERT was designed to be similar to OpenAI GPT, and it outperforms other language representation models in various tasks. The table shows the GLUE test results for BERT and OpenAI GPT, and the table shows the SQuAD 1.1 results for different models. BERT comes in two sizes: BERT BASE and BERT LARGE. The larger model has more parameters and generally performs better. Fine-tuning BERT on different tasks is illustrated in Figure 4. Additional details and ablation studies are presented in Appendices A, B, and C." }
```

Download

In this study session, we have discussed about:

1. Introduction to AI Agents:
  1. The concept of AI Agents.
  2. Why do we need AI Agents and its application.
2. Implementing an AI Agent using LangChain library.
3. Introduction to smolagents library.
4. Implementing an AI Agent using smolagents.
5. Implementing an AI Agent for Vietnamese news summarization using smolagents.

# Summarization and Q&A

---

