

AI VIETNAM
All-in-One Course
(TA Session)

Multi-agent LLMs

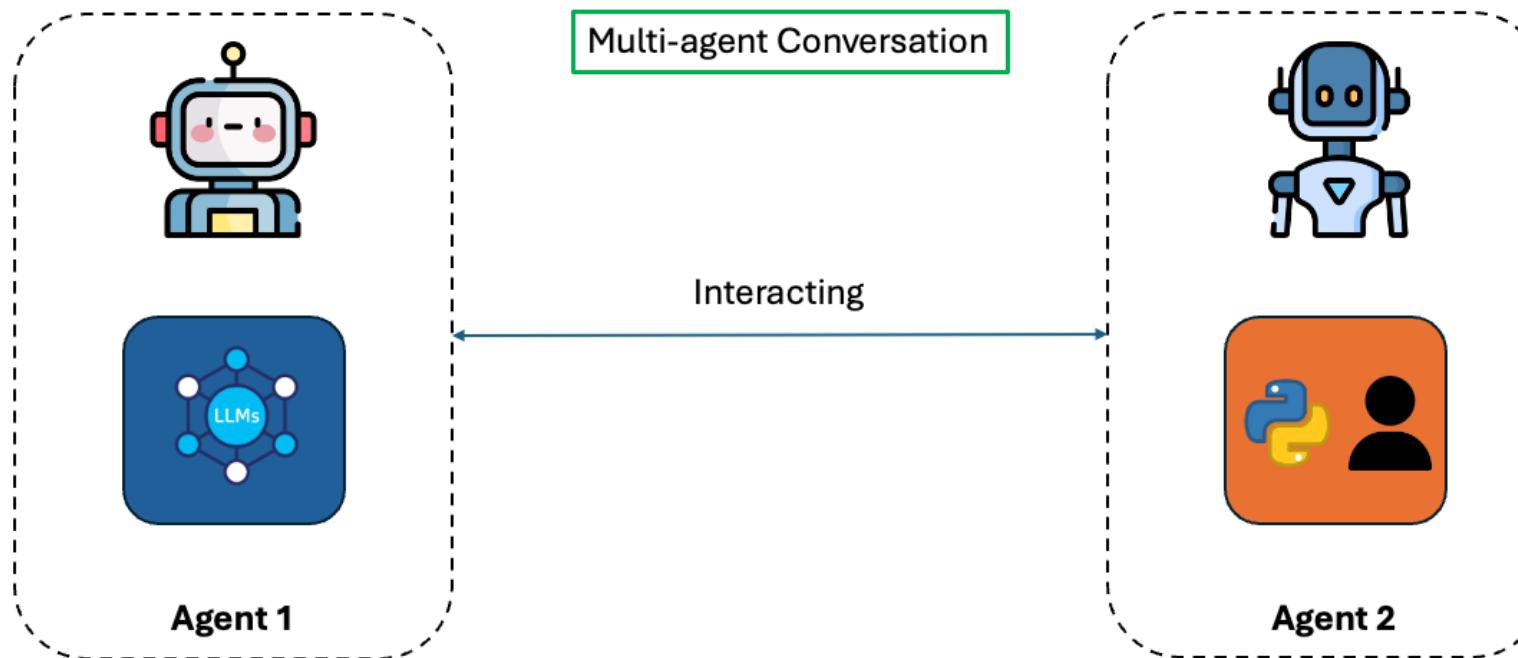
Project



AI VIET NAM
@aivietnam.edu.vn

Dinh-Thang Duong – TA
Nguyen-Thuan Duong – TA

Objectives



In this lecture, we will discuss about:

1. What is LLMs agent?
2. What is Multi-agent LLMs?
3. How to implement LLMs agent with LangChain?
4. How to implement Multi-agent LLMs with Autogen?

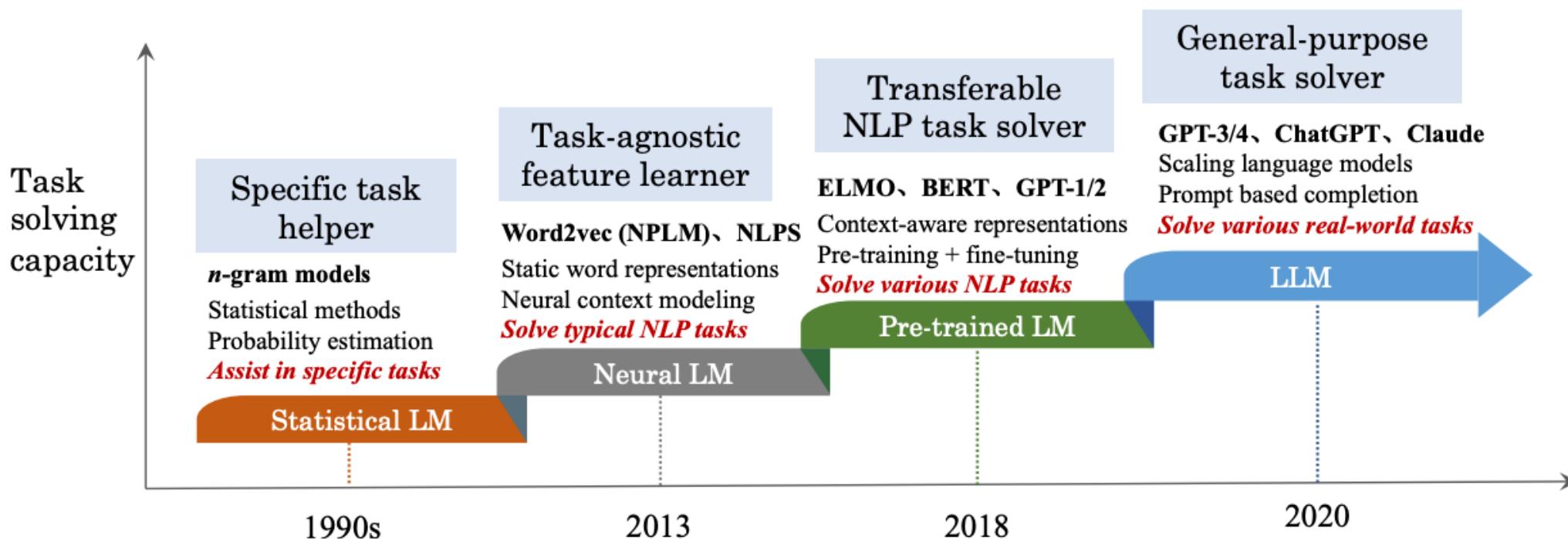
Outline

- Introduction
- LLMs Agent with LangChain
- Multi-agent LLMs with Autogen
- Question

Introduction

Introduction

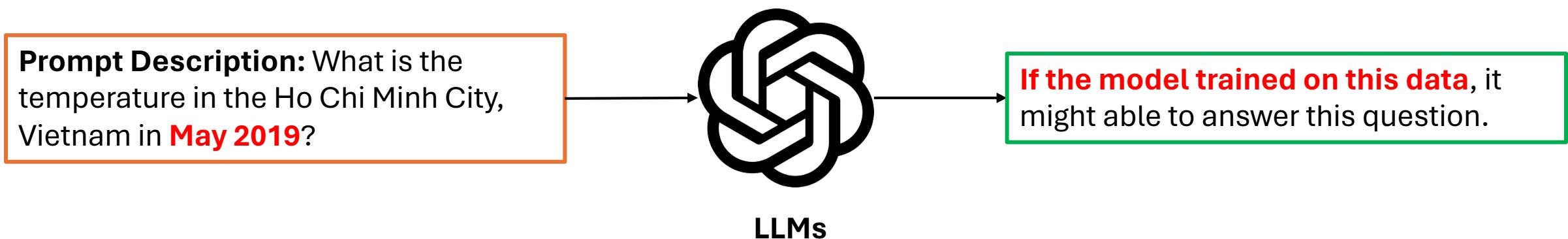
❖ Getting Started



Pre-trained LLMs are often general purpose

Introduction

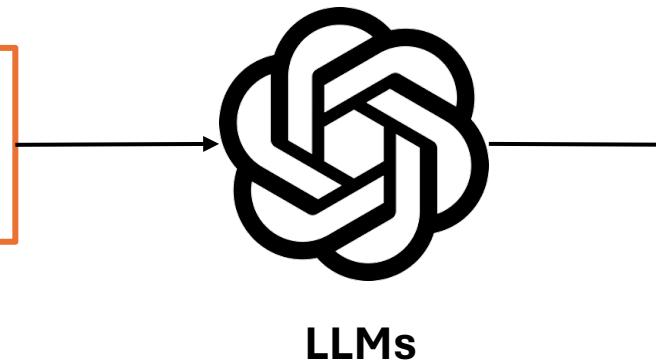
❖ Getting Started



Introduction

❖ Getting Started

Prompt Description: What is **the current temperature** in the Ho Chi Minh City, Vietnam?

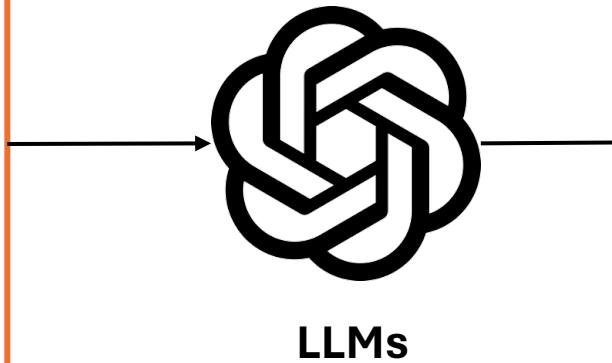


Since we **inquiry a realtime data**, any pre-trained would not be able to answer this question.

Introduction

❖ Getting Started

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.

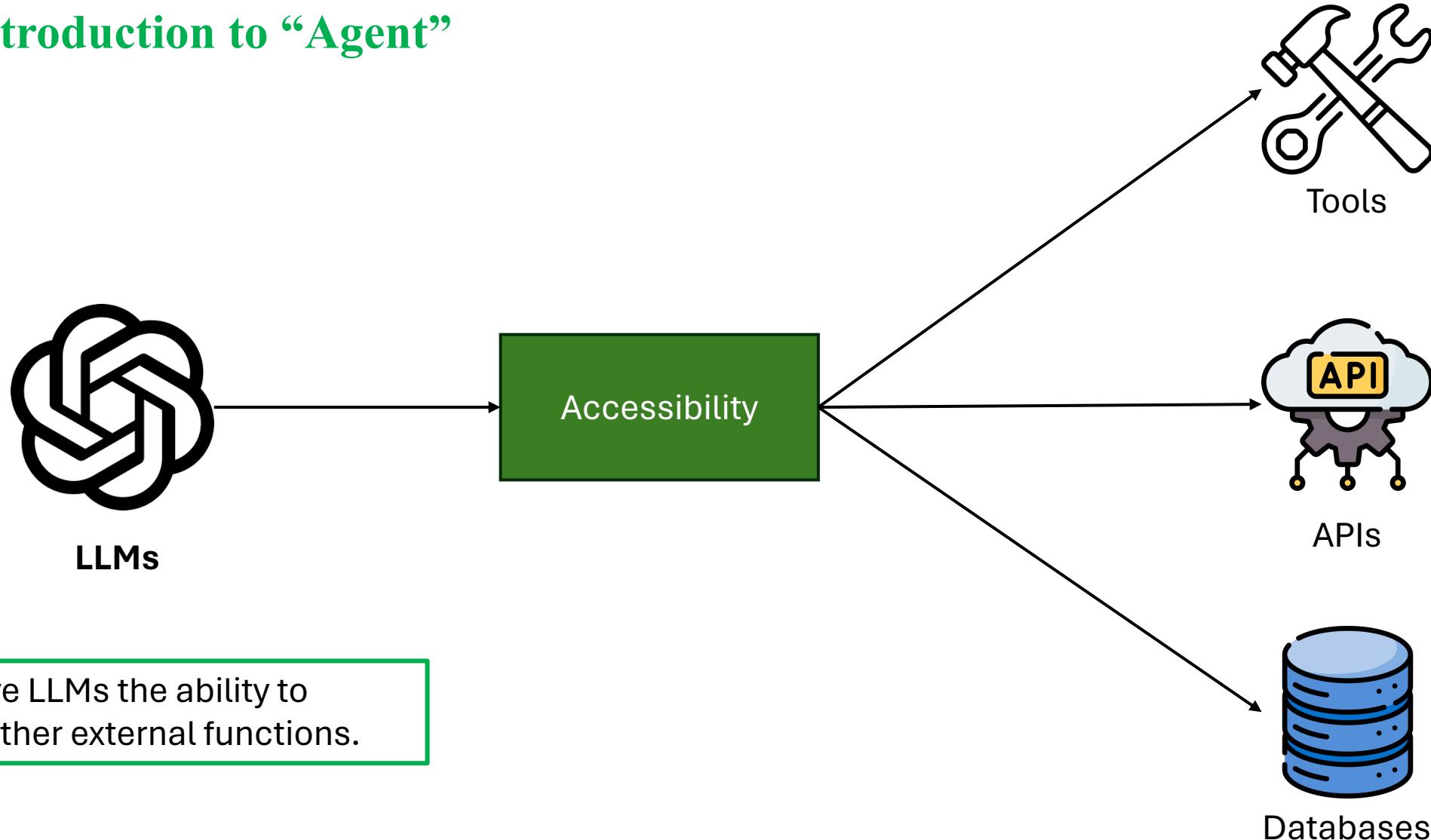


ChatGPT

Certainly! Unfortunately, I can't provide real-time data or create graphical representations directly, but I can offer some insights into the trends in temperature in Ho Chi Minh City over the past decade and discuss their potential impacts.

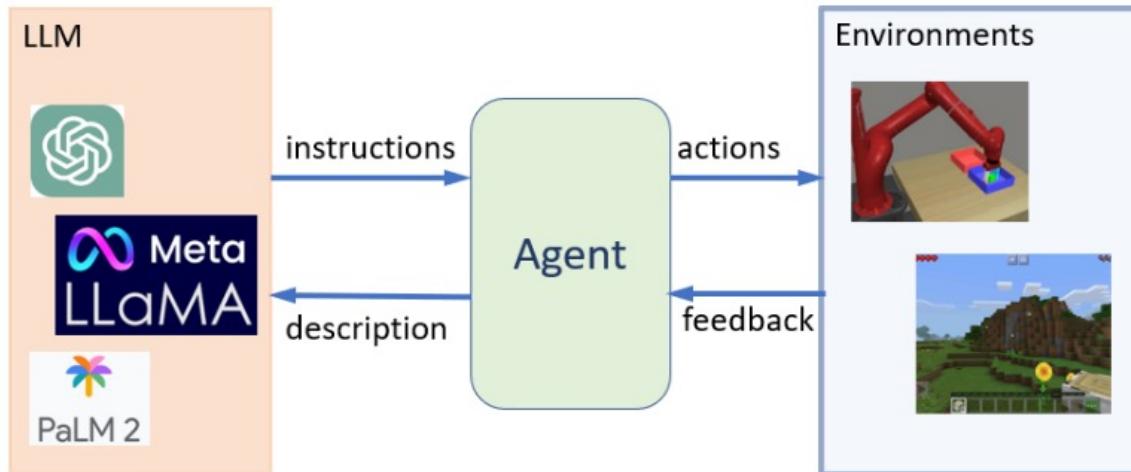
Introduction

❖ Introduction to “Agent”



Introduction

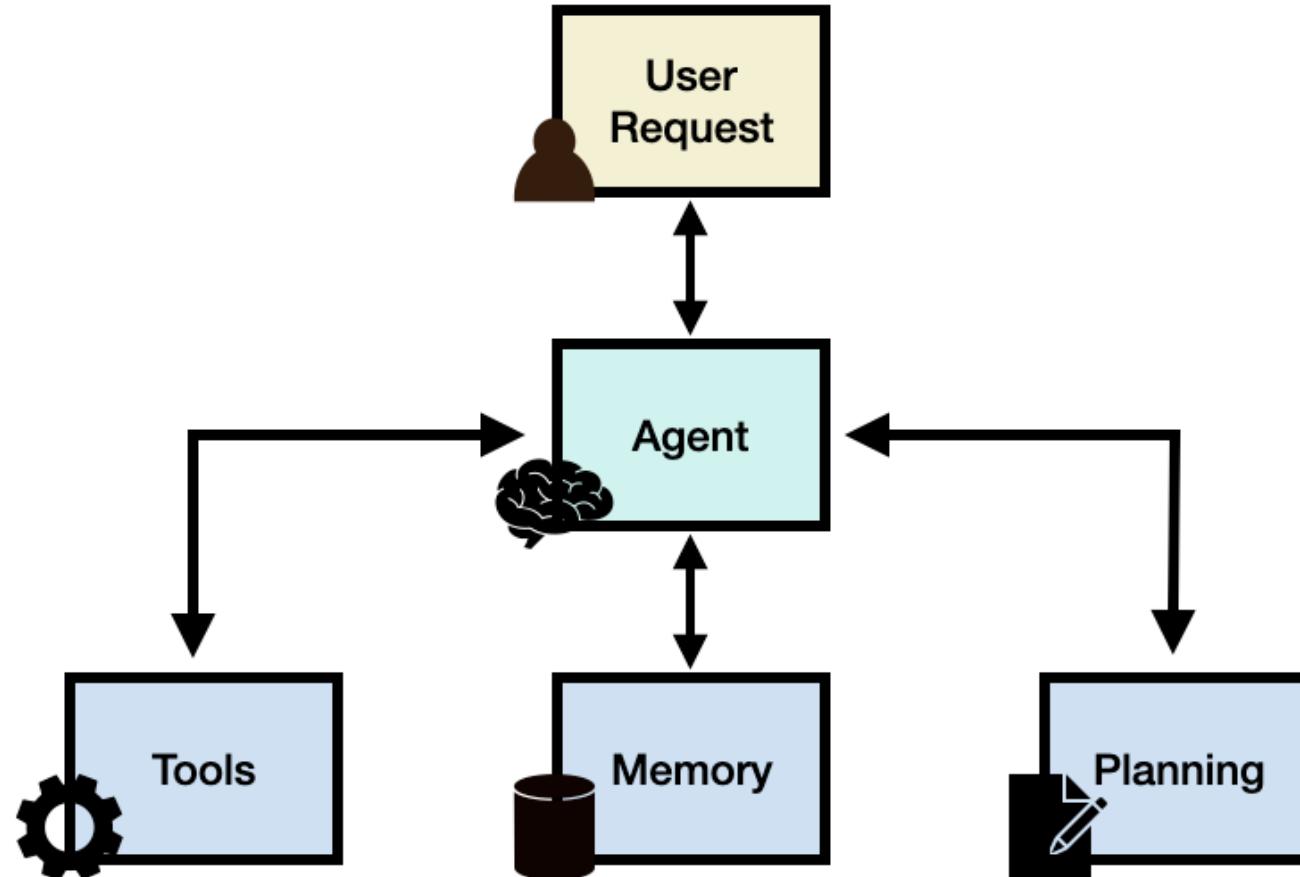
❖ LLMs Agent



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



LLMs Agent framework

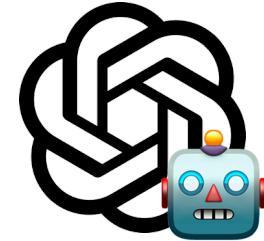
Introduction

❖ General LLMs vs LLMs Agent



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

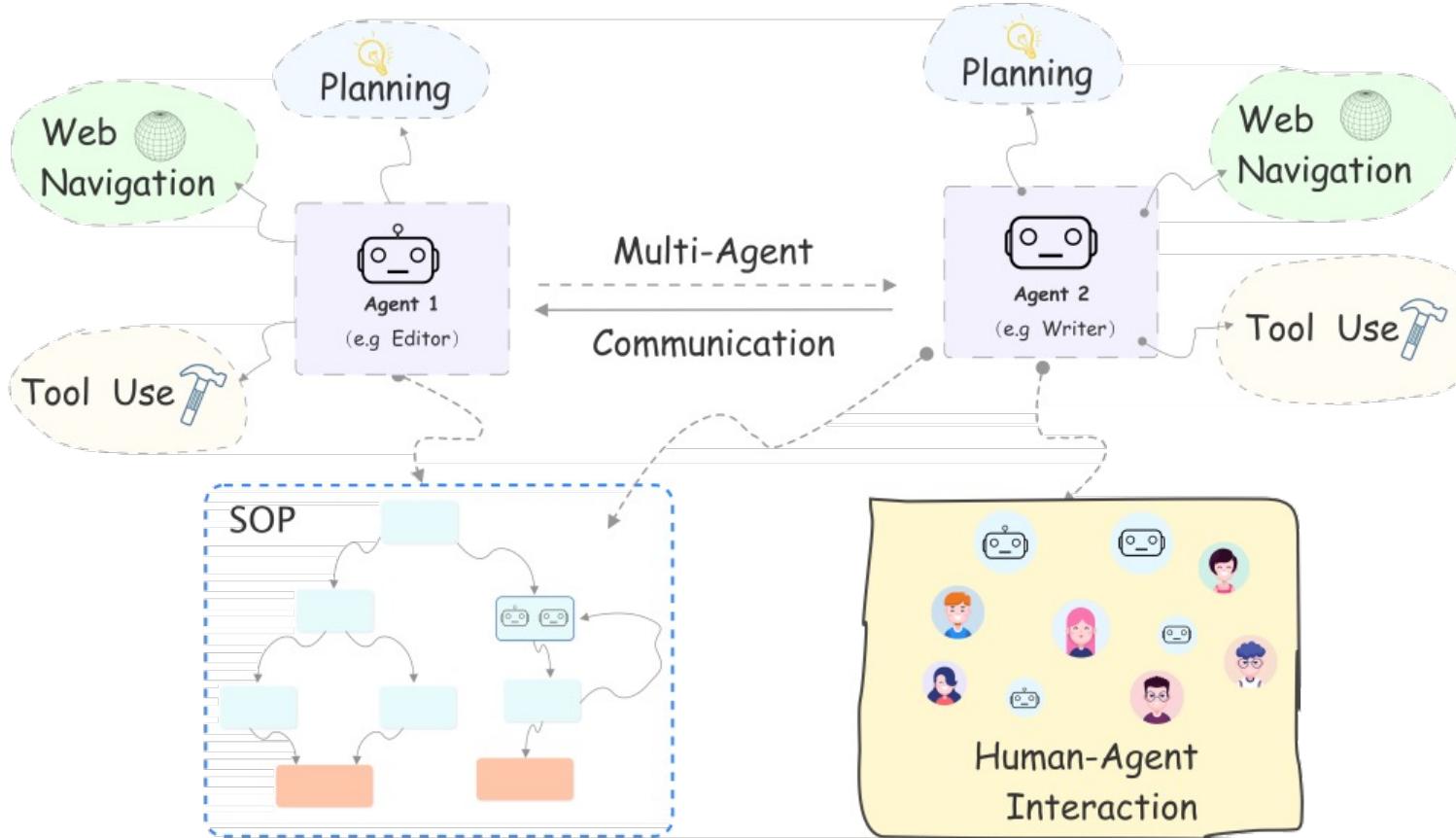
❖ Introduction to “Multi-agent”



How about setting up an environment that involved with multiple agents working together to produce the final result ?

Introduction

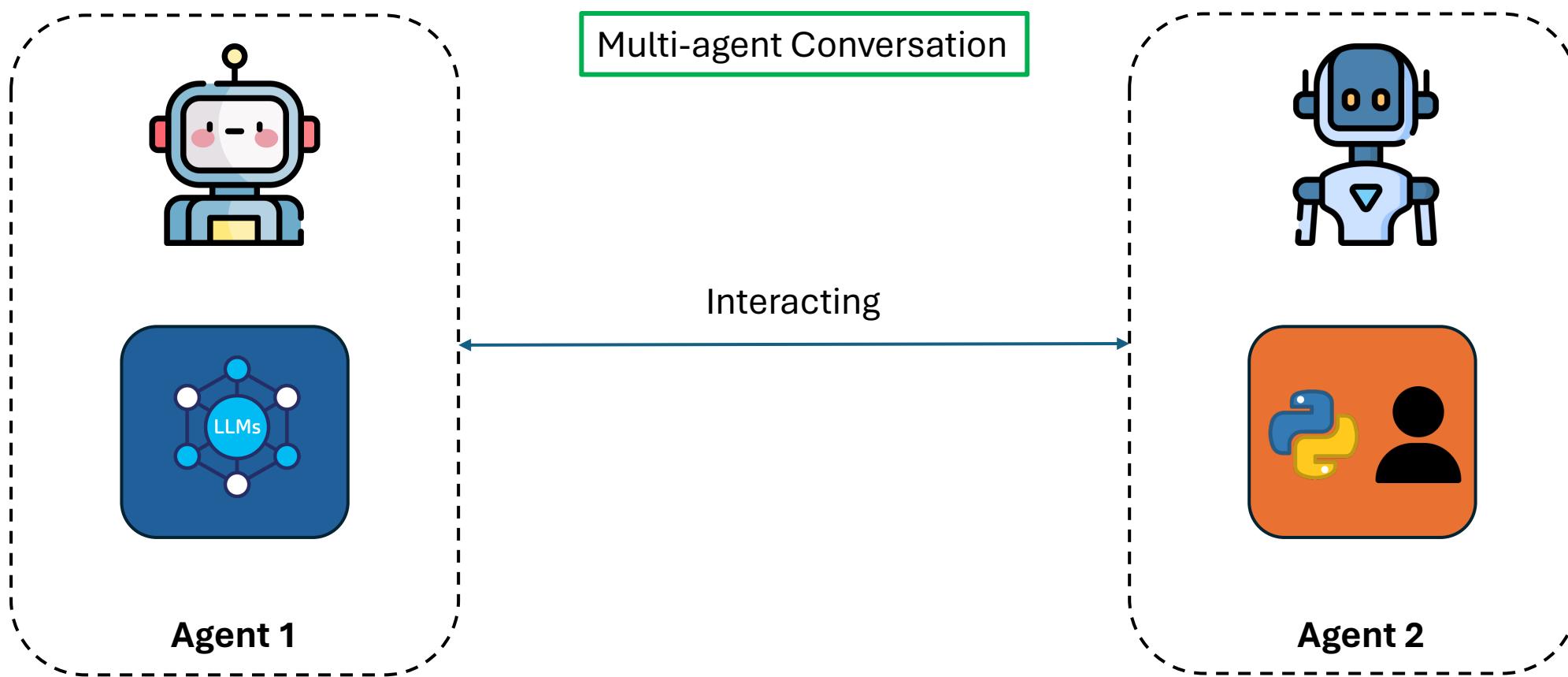
❖ Introduction to “Multi-gent”



In this system, all agents with the ability to use tools, interacting to each other to solve a given task.

Introduction

❖ Multi-agent LLMs



LLMs Agent with LangChain

LLMs Agent with LangChain

❖ Recap

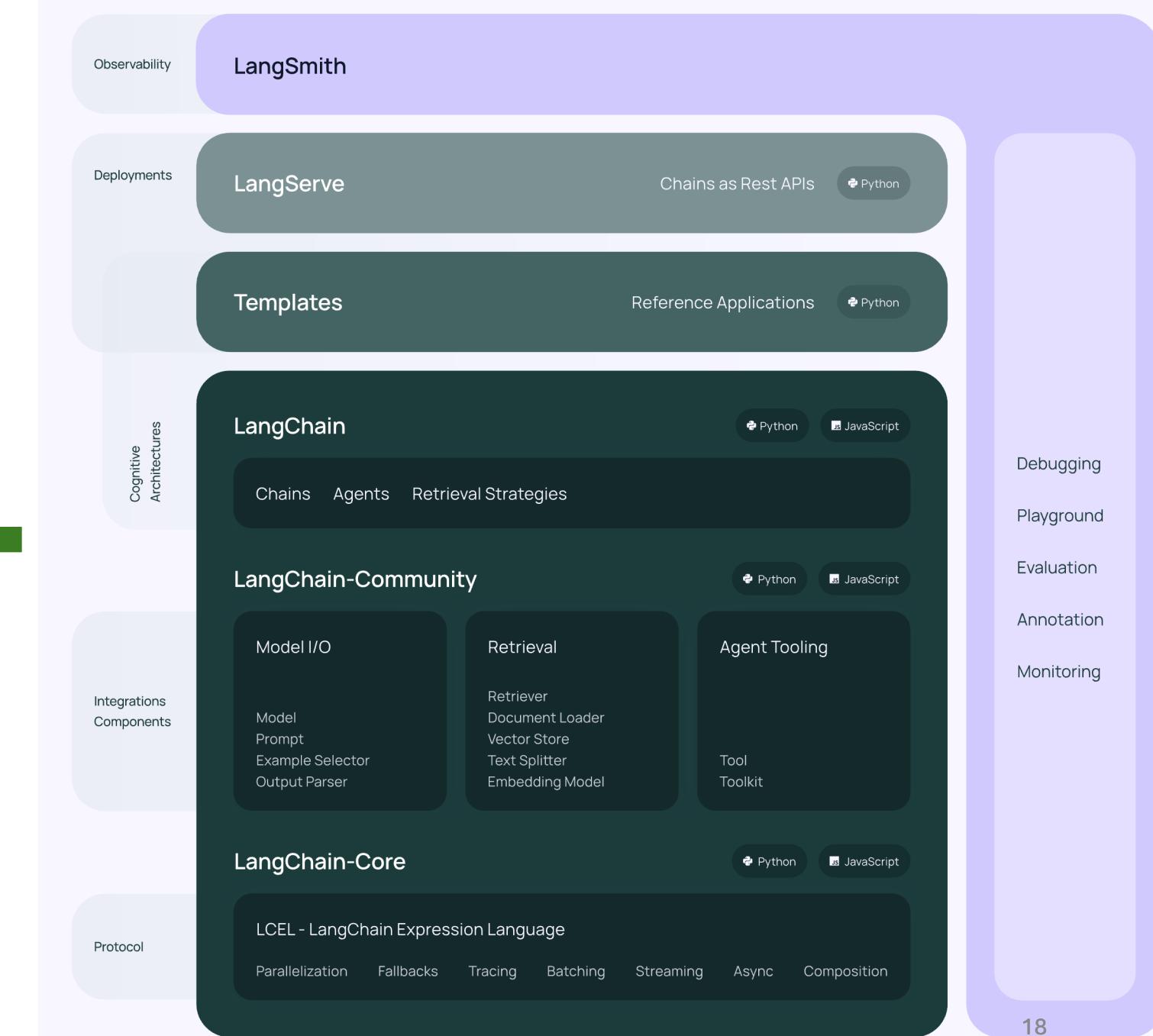


LangChain

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

LLMs Agent with LangChain

❖ Recap



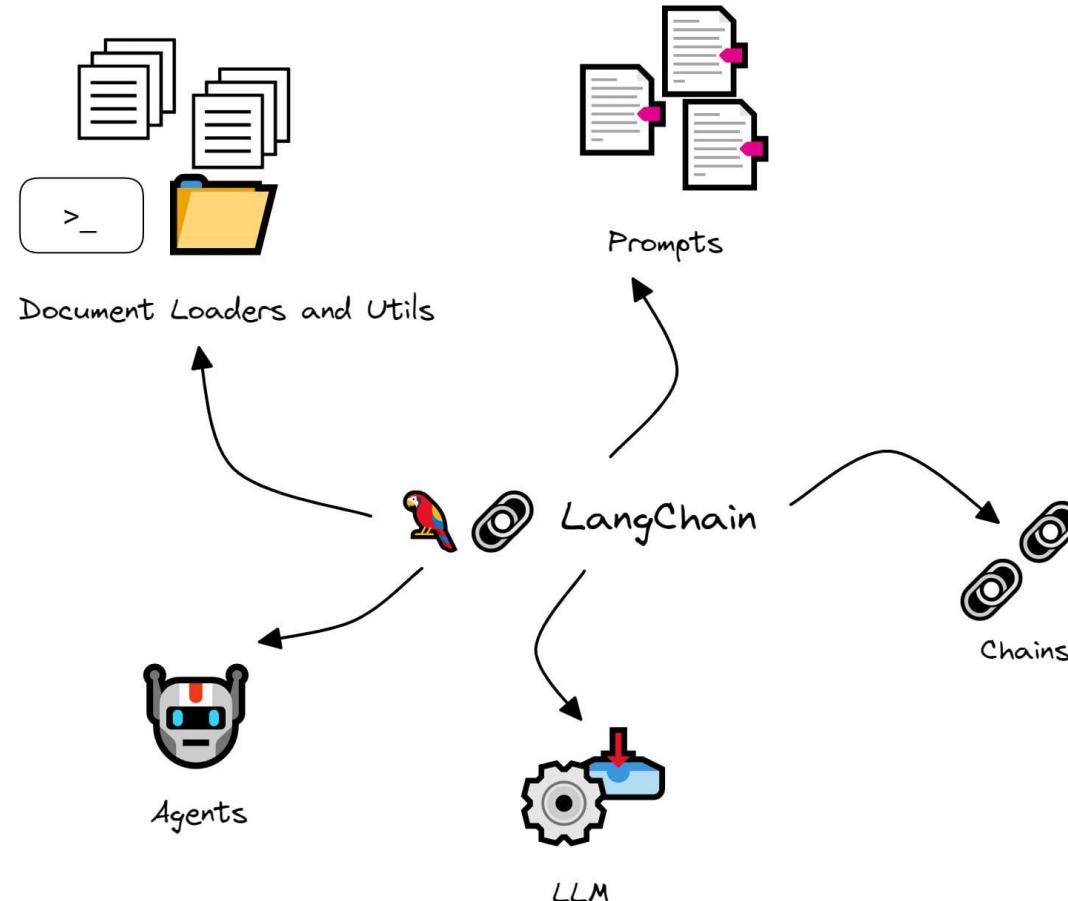
LLMs Agent with LangChain

❖ Recap: LangChain Components



LLMs Agent with LangChain

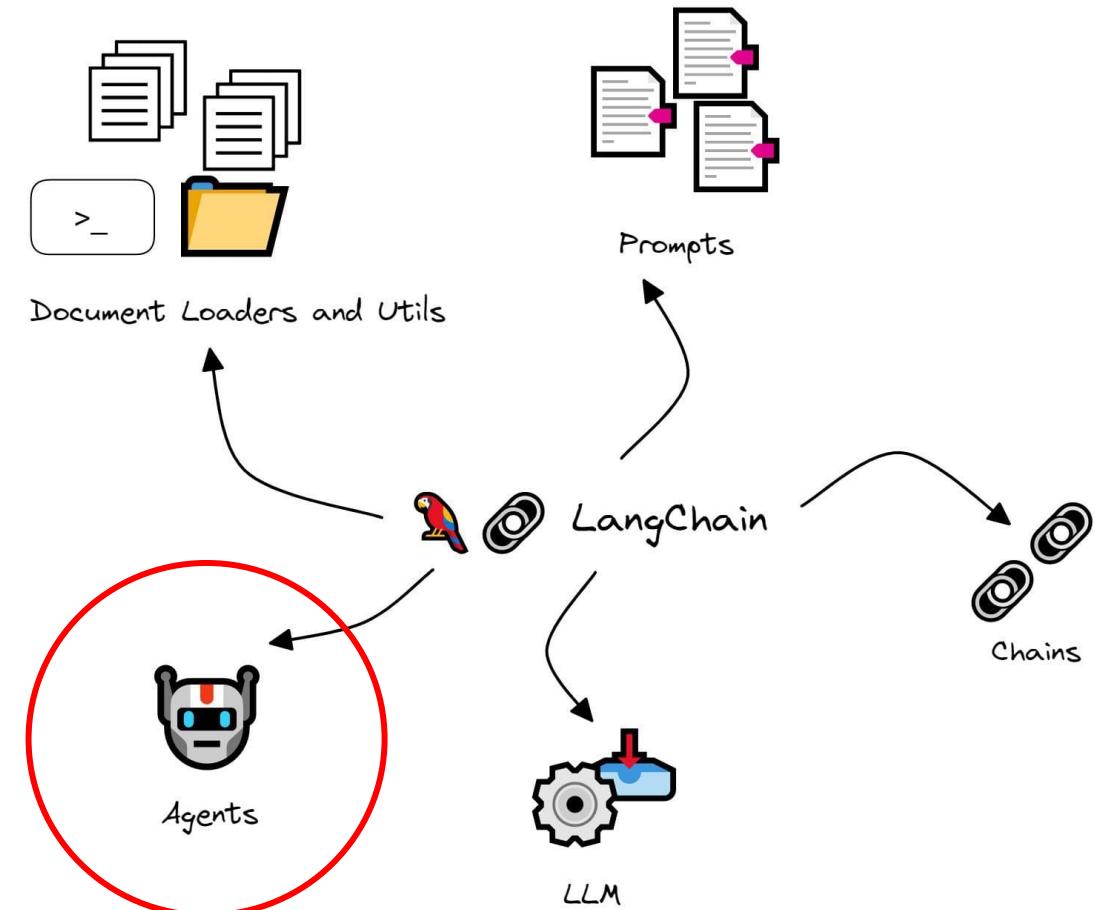
❖ LangChain Components



LLMs Agent with LangChain

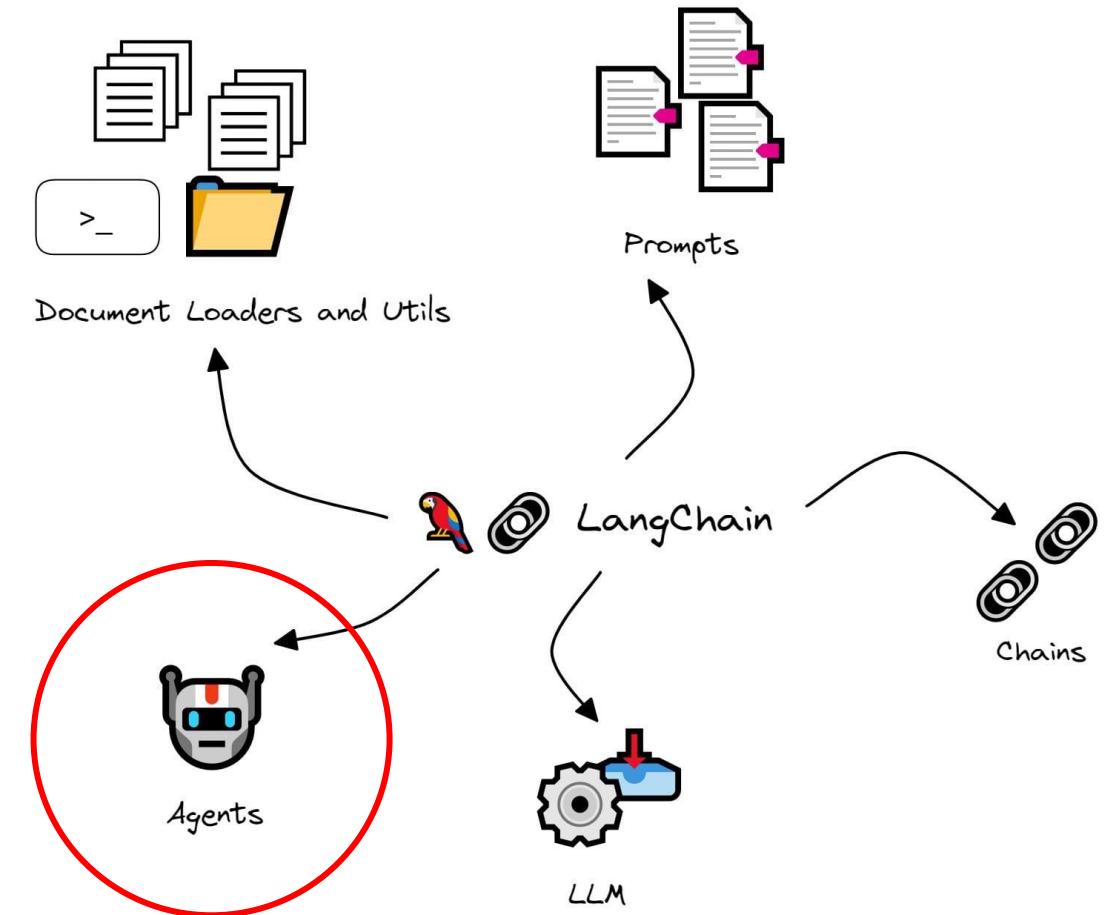
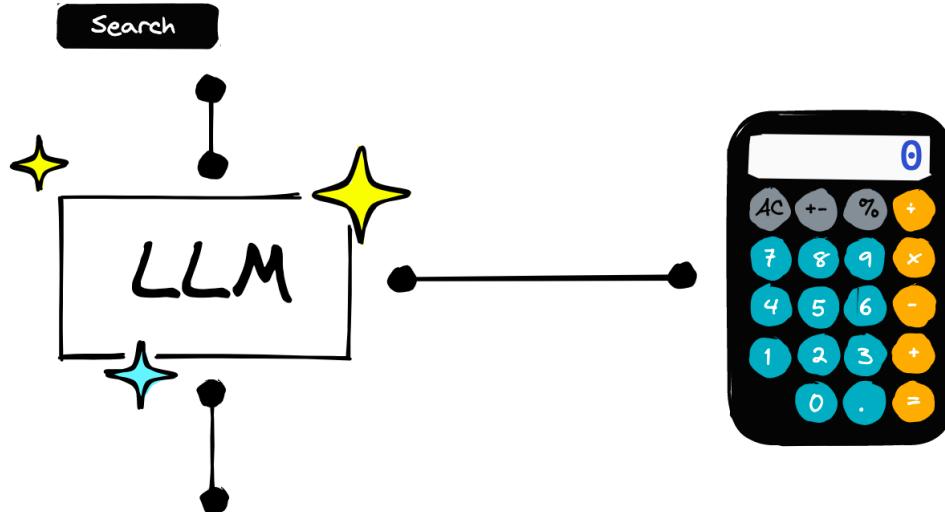
❖ LangChain Agent Component

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



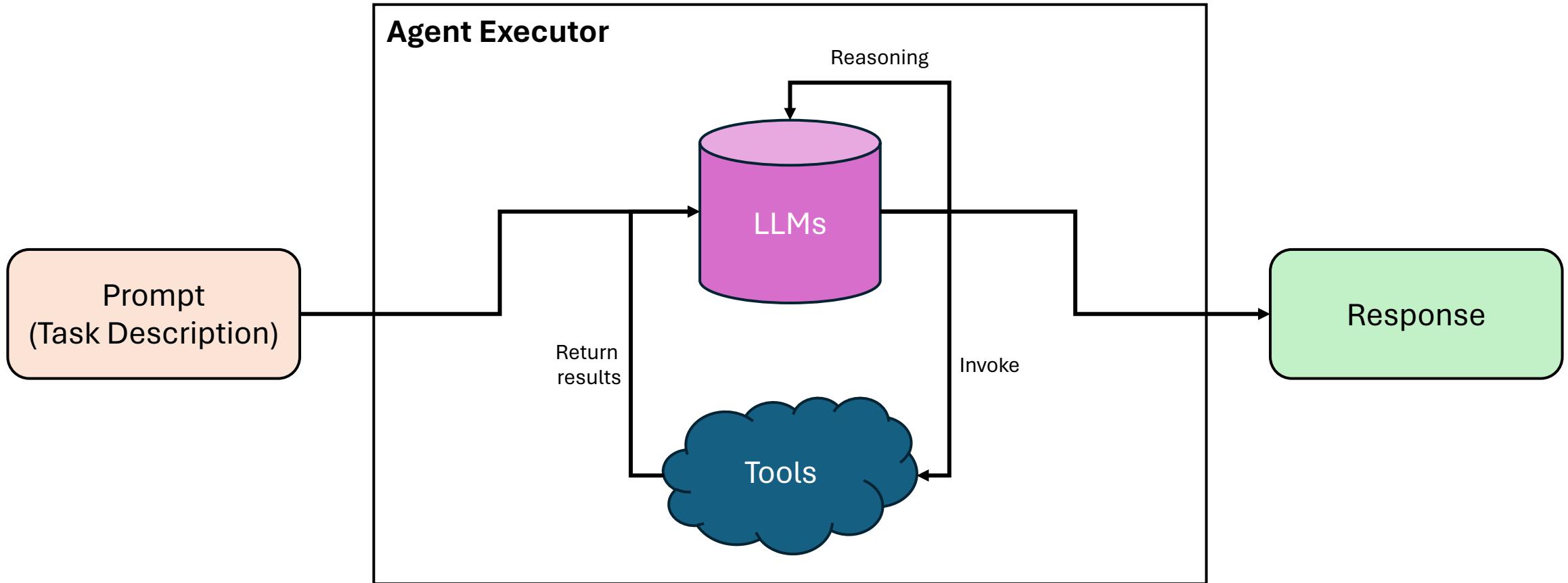
LLMs Agent with LangChain

❖ LangChain Agent Component



LLMs Agent with LangChain

❖ LLMs Agent Pipeline



LLMs Agent with 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()
```



Initialize Tavily Search (or other search API)

LLMs Agent with LangChain

❖ Example: Online Search

The screenshot shows the Tavily AI platform's Overview page. On the left, a sidebar menu includes 'Overview' (selected), 'Research Assistant', 'Research Reports', 'API Playground', and 'Documentation'. The main content area has a header 'Pages / Overview' and 'Overview'. It features a large purple box titled 'CURRENT PLAN' with 'Researcher' and an 'API Limit' of '8 / 1,000 Requests'. Below this is an 'API Key' section with a placeholder 'tvly-*****' and a 'Contact us' button at the bottom.

Tavily API Key

LLMs Agent with 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 = 0, the wavefunction must be zero at the boundaries. This is the most basic principle of quantum mechanics.'},  
 {'url': 'https://www.britannica.com/science/quantum-mechanics',  
 'content': 'Given the wavelength of the wavefunction, the energy levels of the system can be calculated using the formula E = hc/\lambda, 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 matter and energy at the atomic and subatomic level, and it is based on the idea that particles can exist in multiple states simultaneously.'},  
 {'url': 'https://www.livescience.com/35347-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 is based on the idea that particles have definite positions and velocities."}]
```

Search data using Tavily API

LLMs Agent with 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)

LLMs Agent with LangChain

❖ Example: Online Search

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

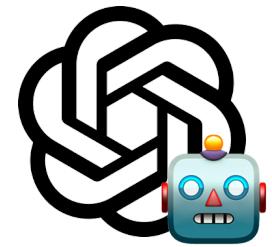
```
[SystemMessagePromptTemplate(prompt=PromptTemplate(input_variables=[], template='You are a helpful assistant')),  
 MessagesPlaceholder(variable_name='chat_history', optional=True),  
 HumanMessagePromptTemplate(prompt=PromptTemplate(input_variables=['input'], template='{input}')),  
 MessagesPlaceholder(variable_name='agent_scratchpad')]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

Create prompt to chat with
OpenAI GPT

LLMs Agent with 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)

LLMs Agent with 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/>) .

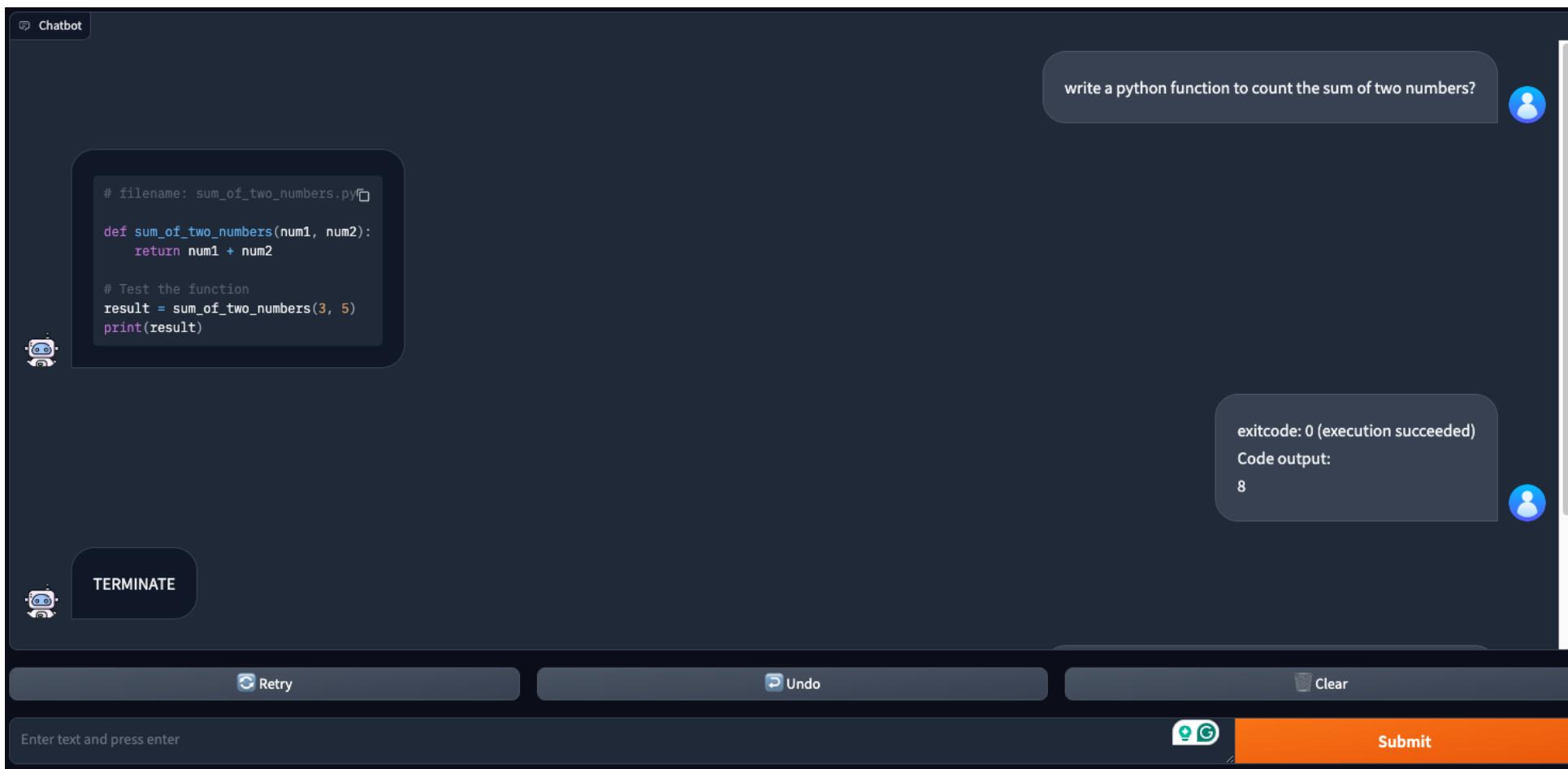
QUIZ

Multi-agent LLMs with Autogen

Multi-agent LLMs with Autogen

❖ Introduction

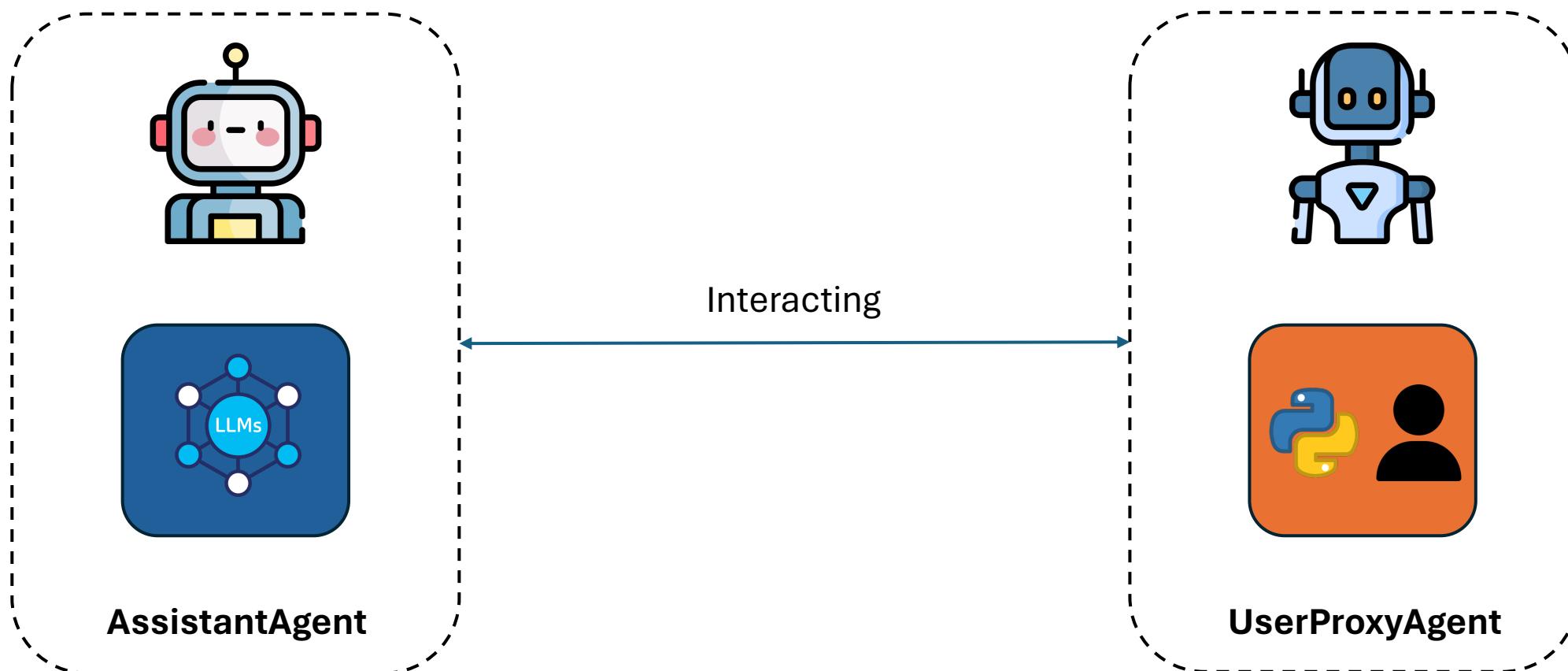
Description: Build a Multi-agent LLMs application that could generate a code snippet related to an input query, then automatically debug the code until completely clean.



Multi-agent LLMs with Autogen

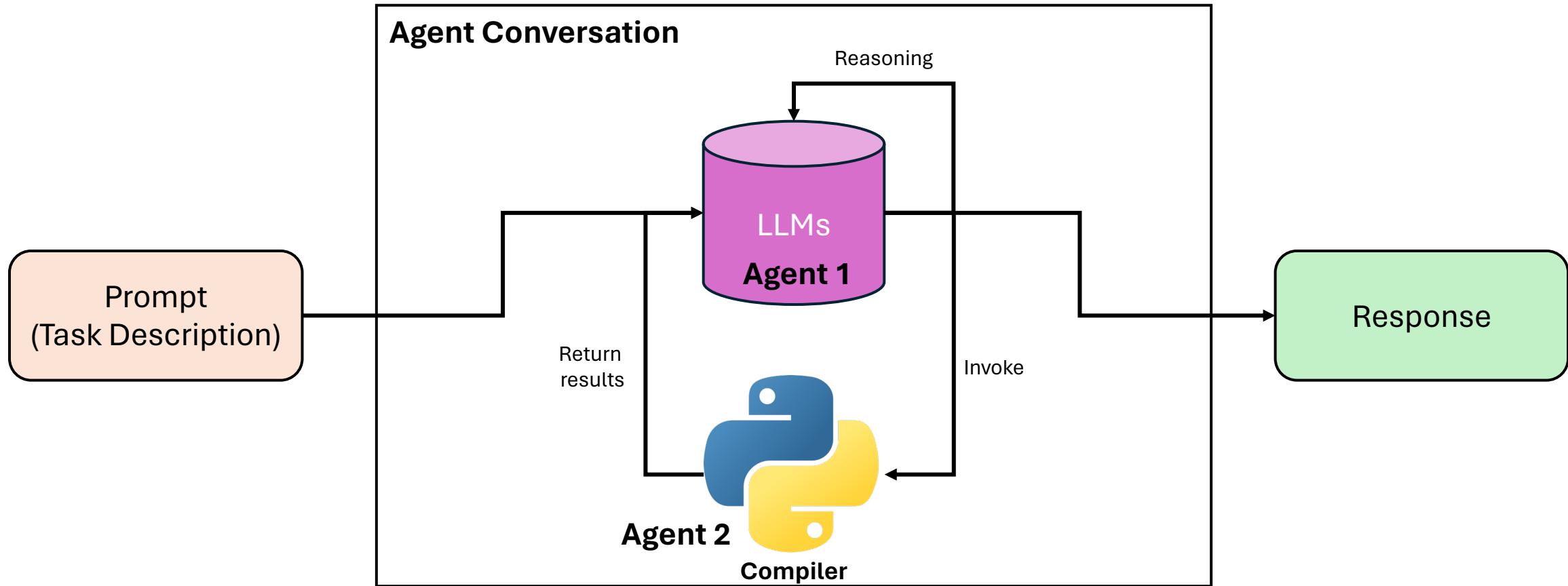
❖ Introduction

Description: Build a Multi-agent LLMs application that could generate a code snippet related to an input query, then automatically debug the code until completely clean.



Multi-agent LLMs with Autogen

❖ Pipeline



Multi-agent LLMs with Autogen

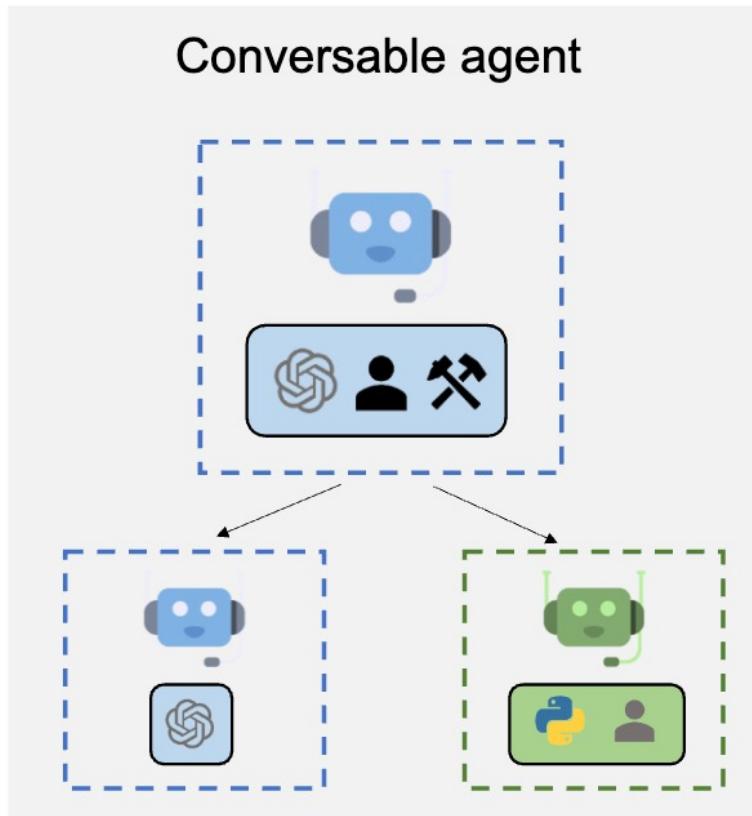
❖ Autogen Introduction



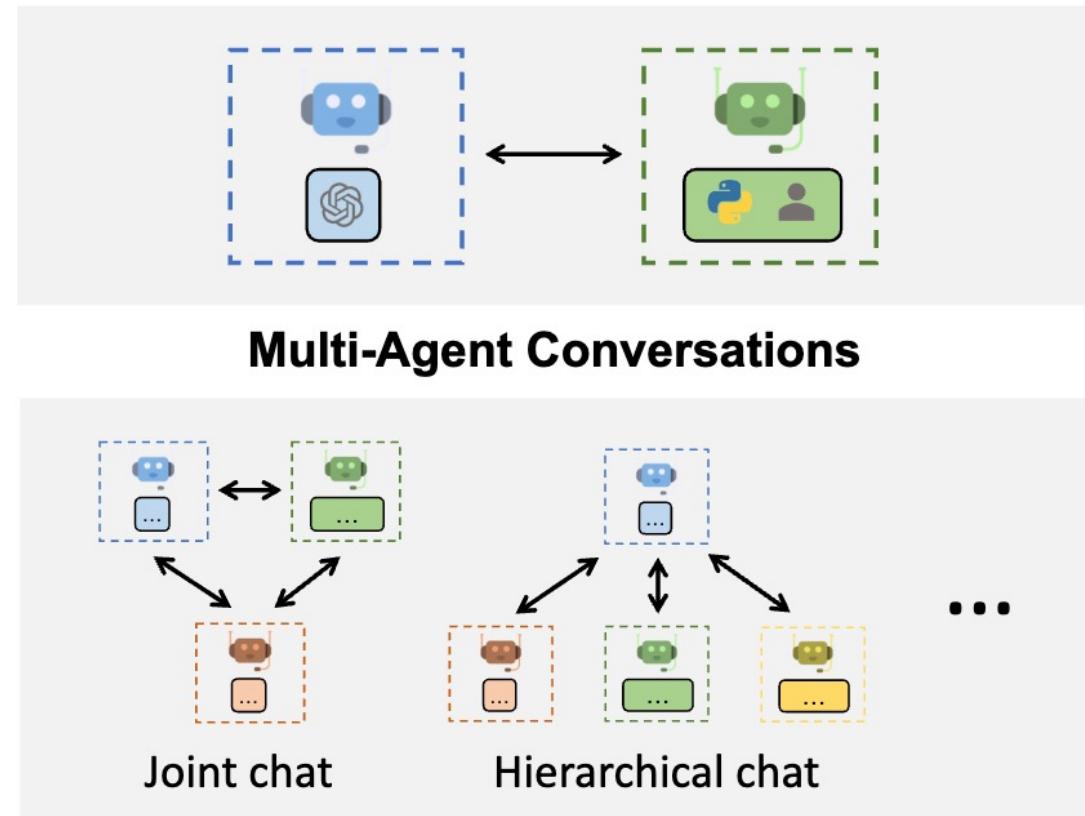
Microsoft Autogen: AutoGen is a framework that enables development of LLM applications using multiple agents that can converse with each other to solve tasks. They can operate in various modes that employ combinations of LLMs, human inputs, and tools.

Multi-agent LLMs with Autogen

❖ Autogen Introduction



Agent Customization



Flexible Conversation Patterns

Multi-agent LLMs with Autogen

❖ Autogen Agent

An Autogen Agent may is:

- LLM (eg. chatGPT)
- Code Executor (Local for Docker)
- Tool (function)
- Input of human.
- or Custom

Autogen ConversationAgent

LLM
(chatGPT)

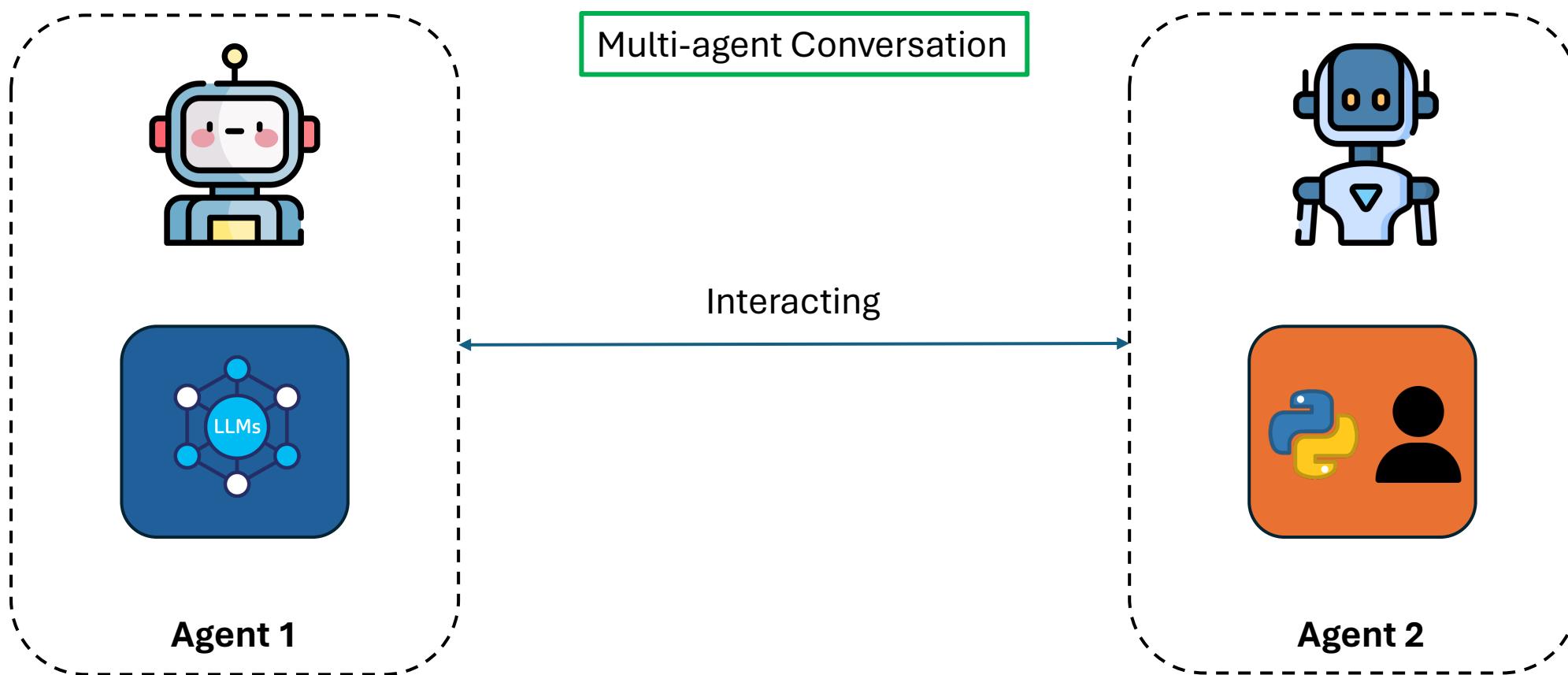
Code
Executor

Tool
(function)

Human
input

Multi-agent LLMs with Autogen

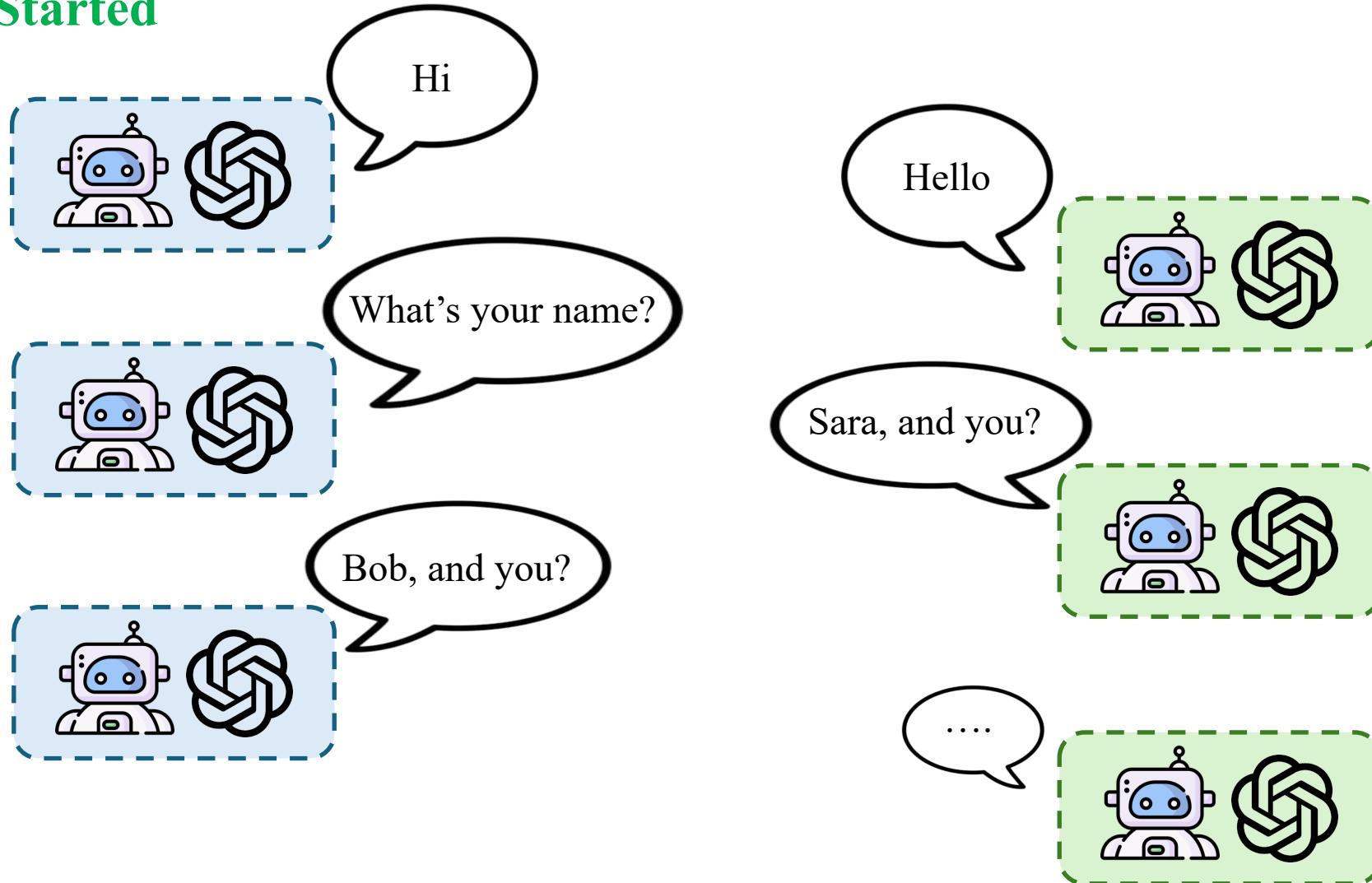
❖ Recap: Multi-agent LLMs



How can we make it with Autogen?

Multi-agent LLMs with Autogen

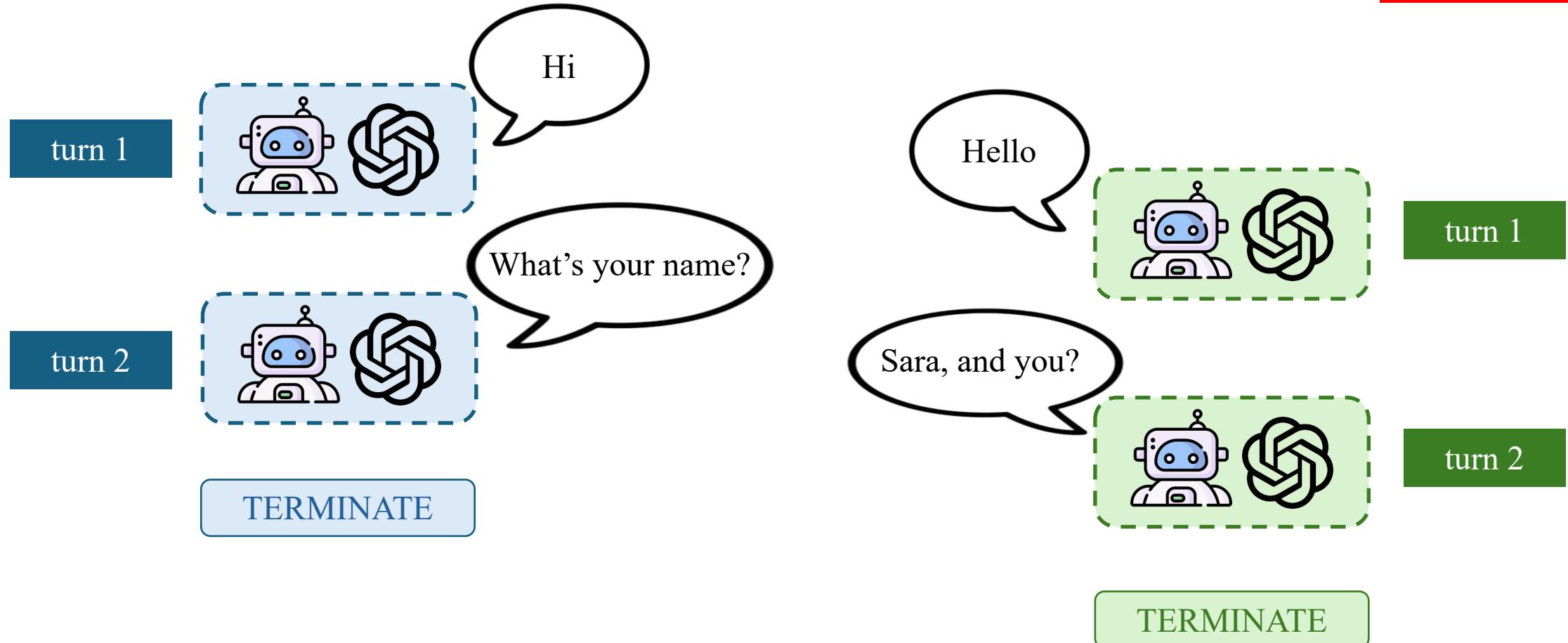
❖ Getting Started



Multi-agent LLMs with Autogen

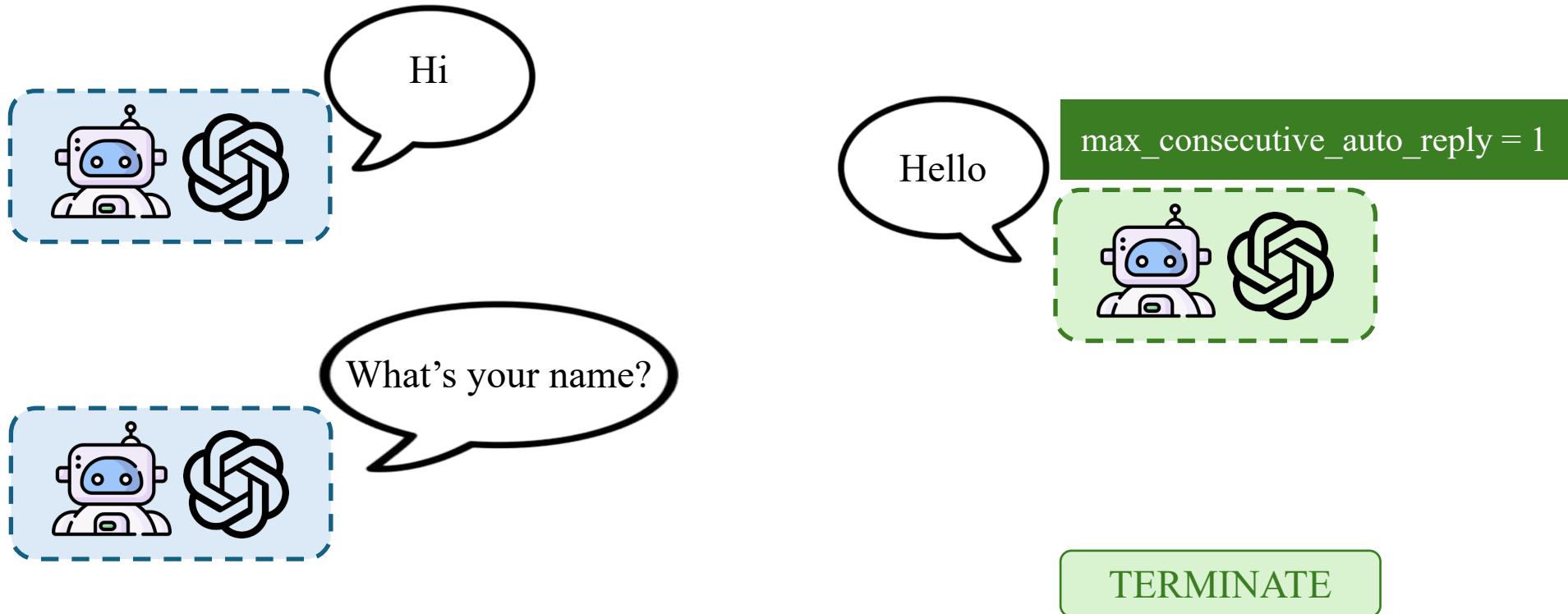
❖ max_turn parameter

max_turn = 2



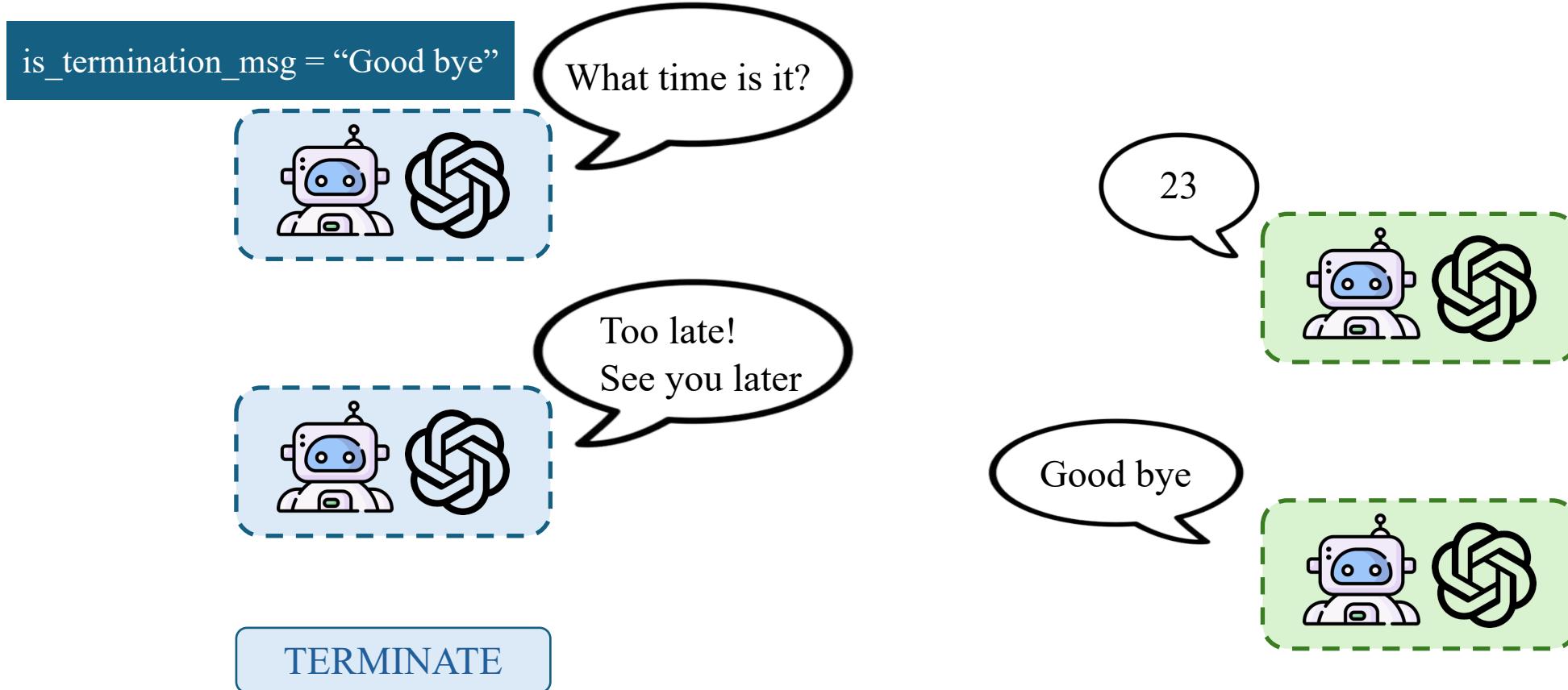
Multi-agent LLMs with Autogen

❖ max_consecutive_auto_reply parameter



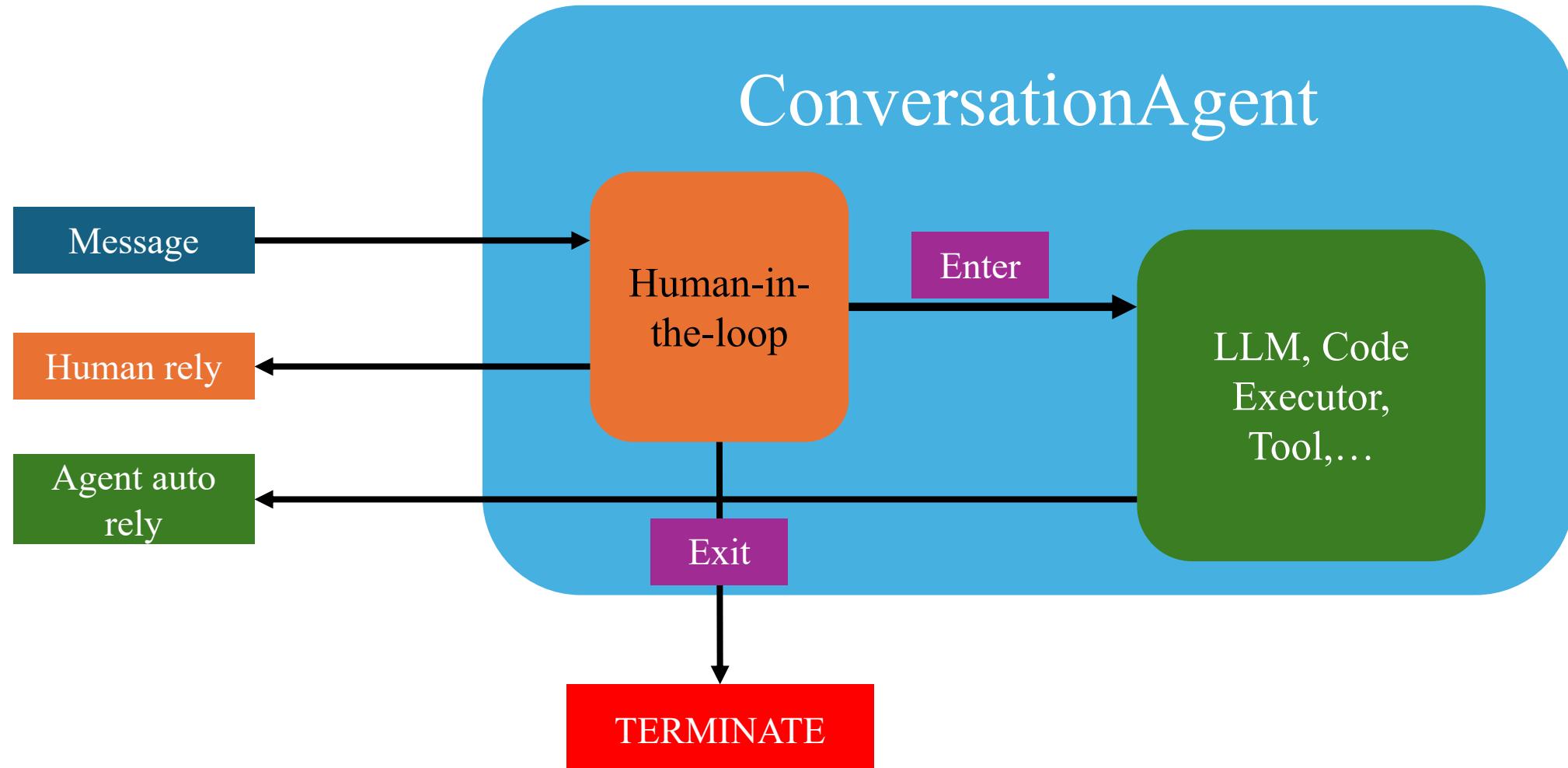
Multi-agent LLMs with Autogen

❖ is_termination_msg parameter



Multi-agent LLMs with Autogen

❖ Getting Started

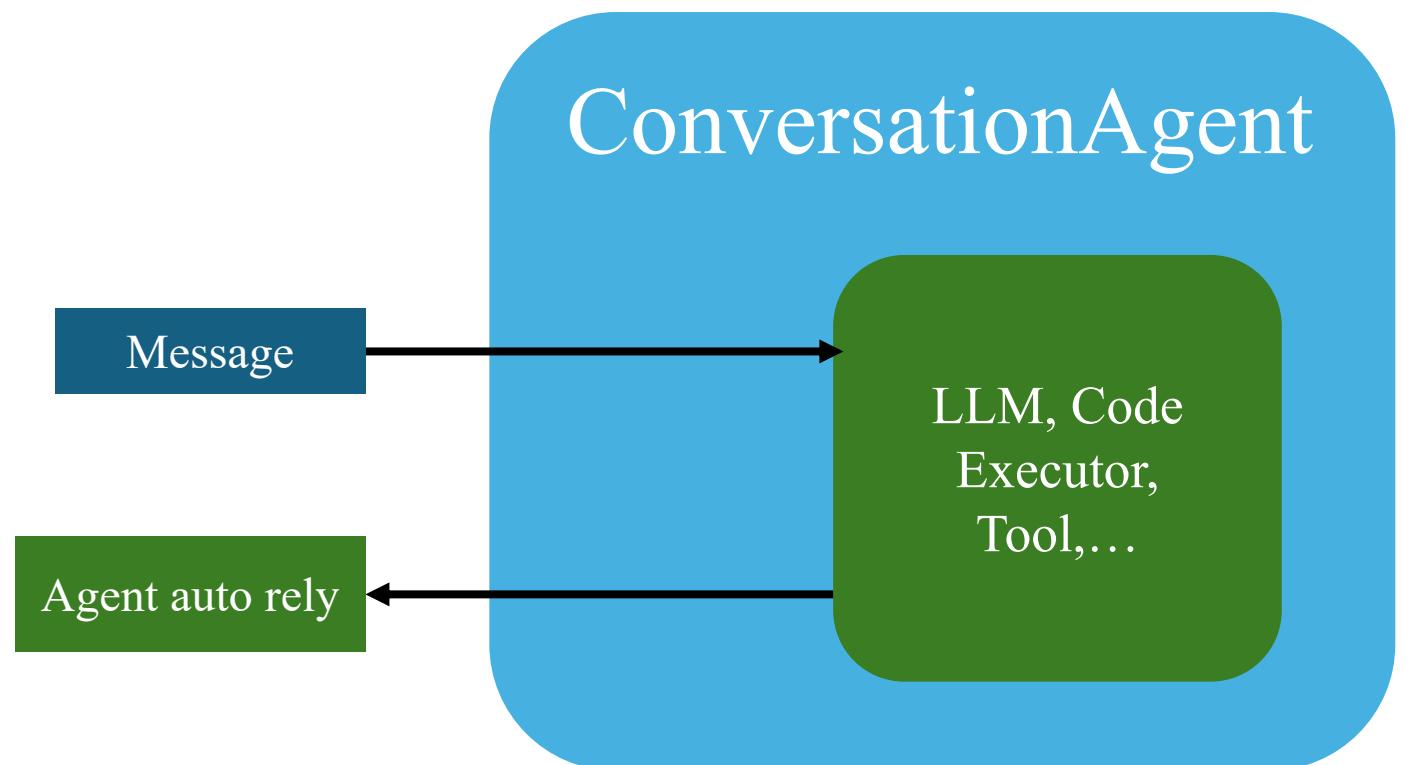


Multi-agent LLMs with Autogen

❖ Human input: NEVER

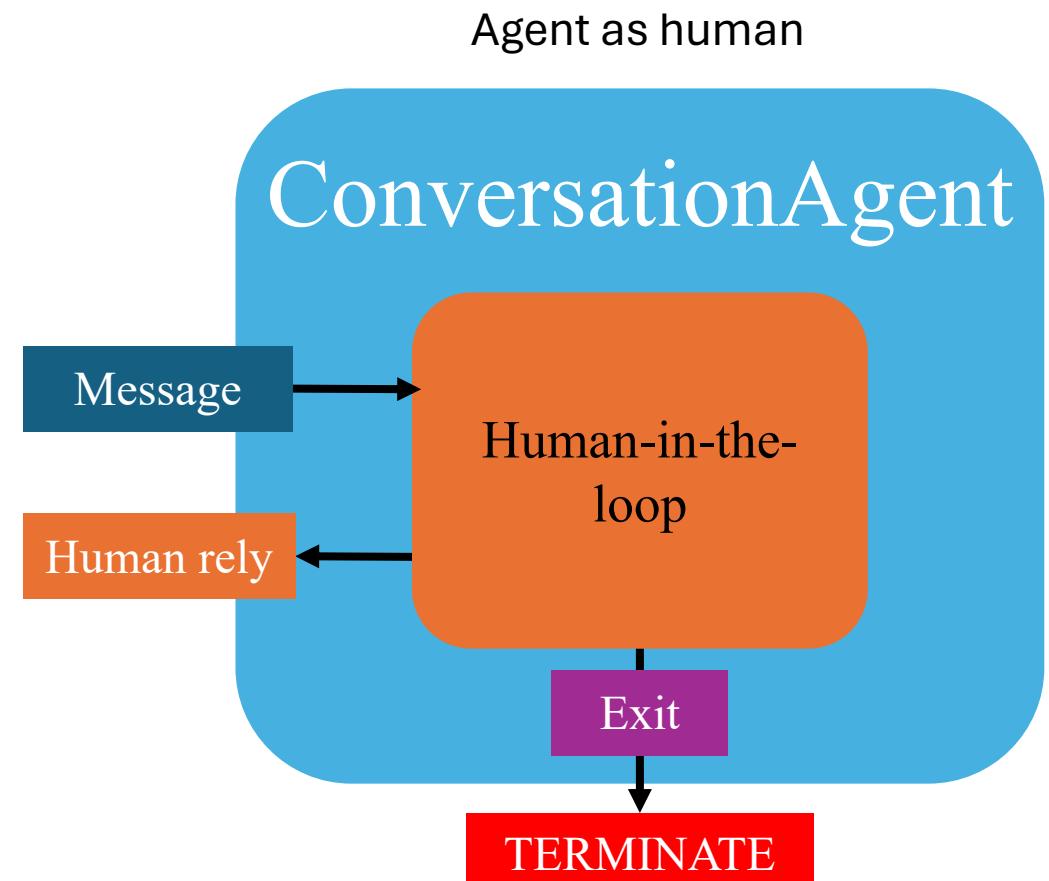
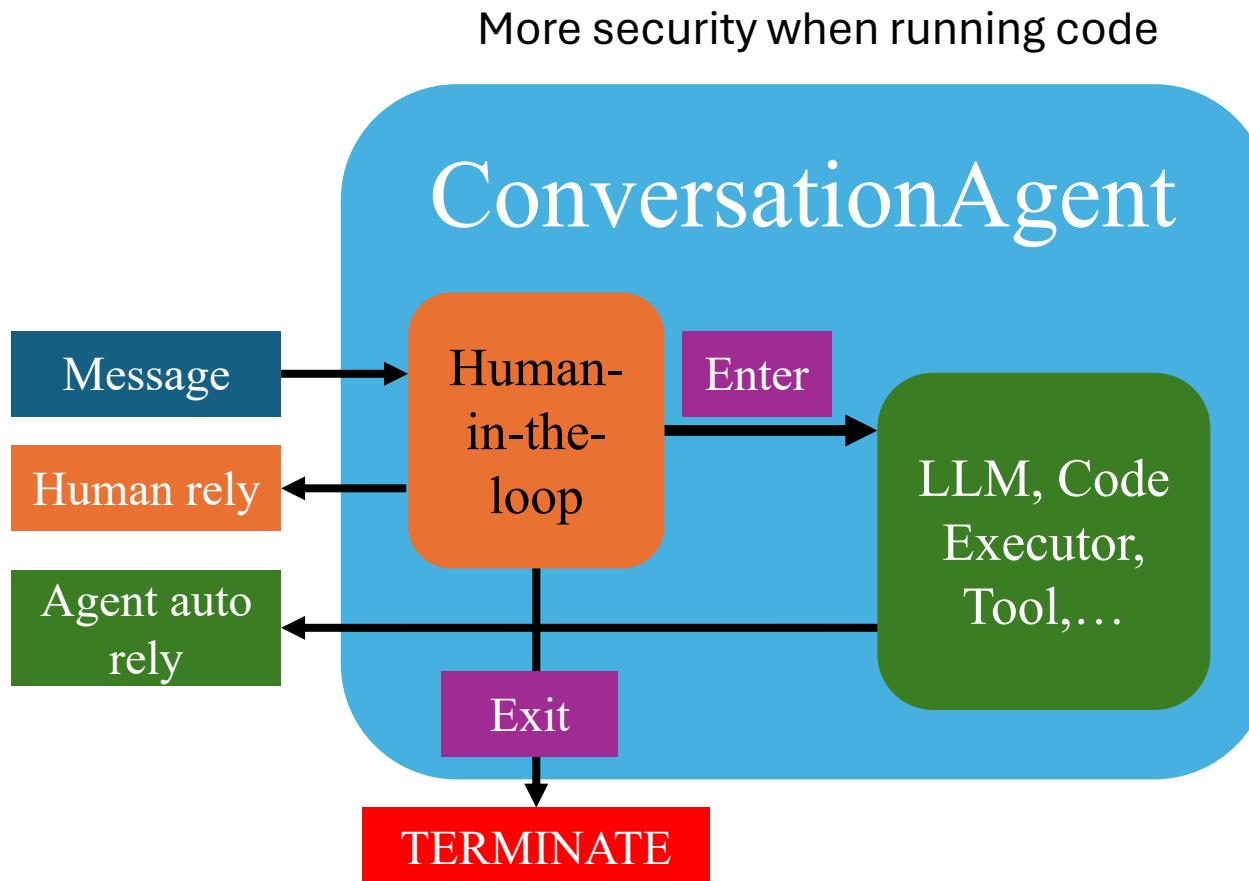
Never ask for human input, use chat termination to stop the conversation.

High risk when the agent is a code executor.



Multi-agent LLMs with Autogen

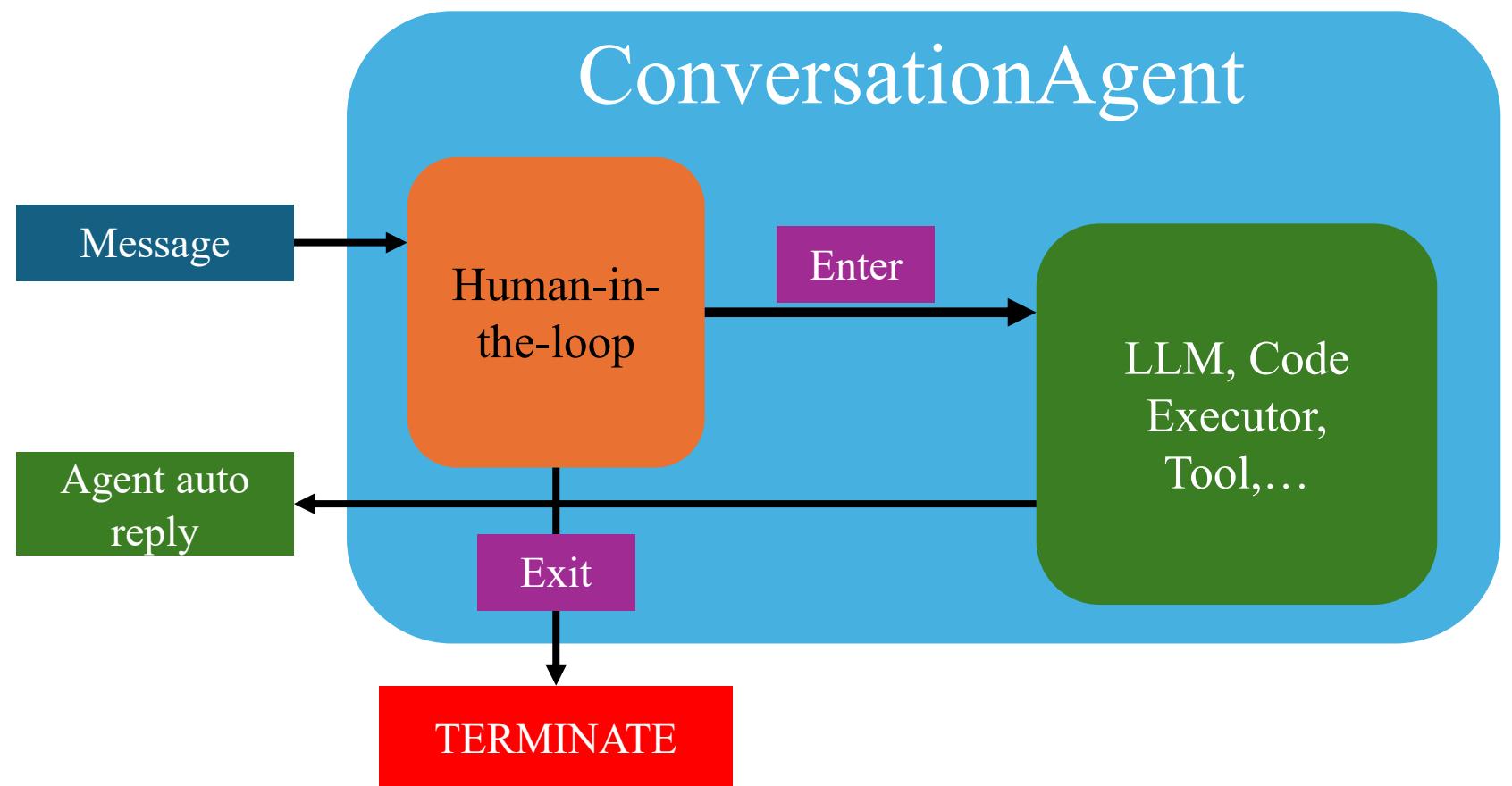
❖ Human input: ALWAYS



Multi-agent LLMs with Autogen

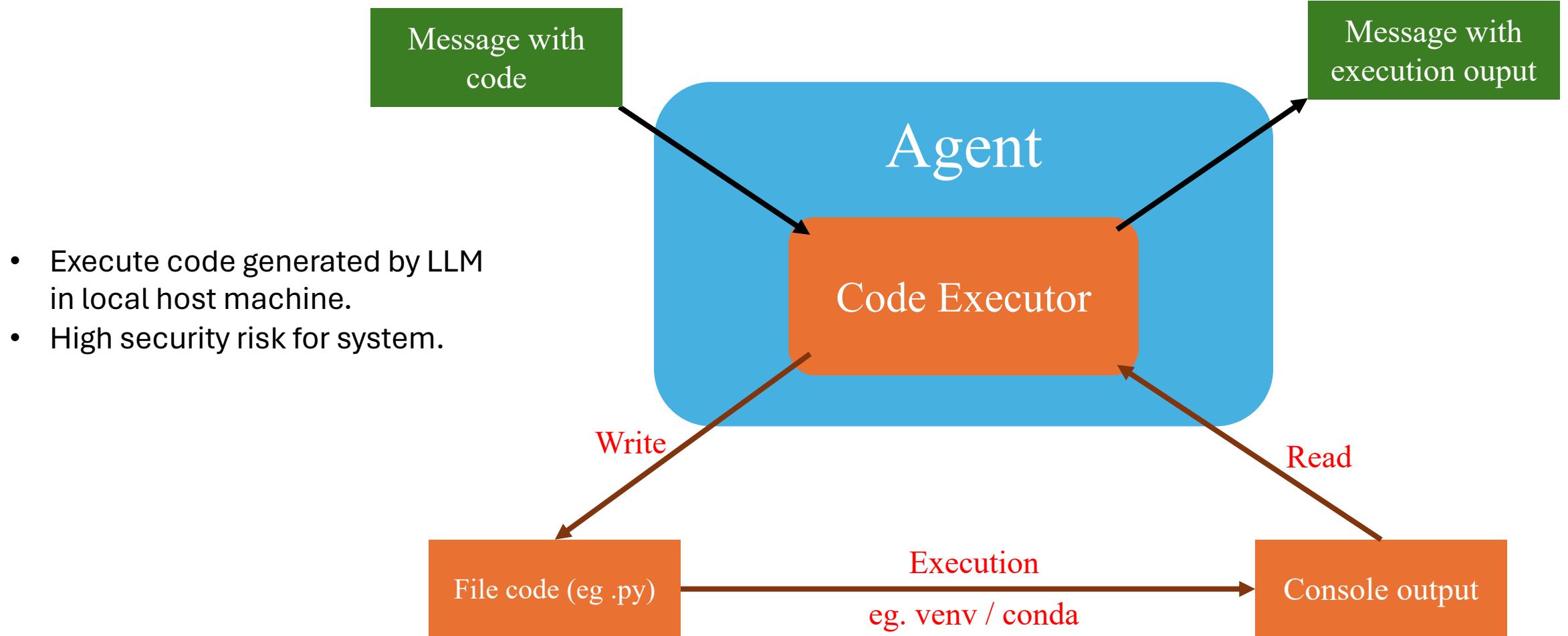
❖ Human input: TERMINATE

Ask the human when the conversation should stop.



Multi-agent LLMs with Autogen

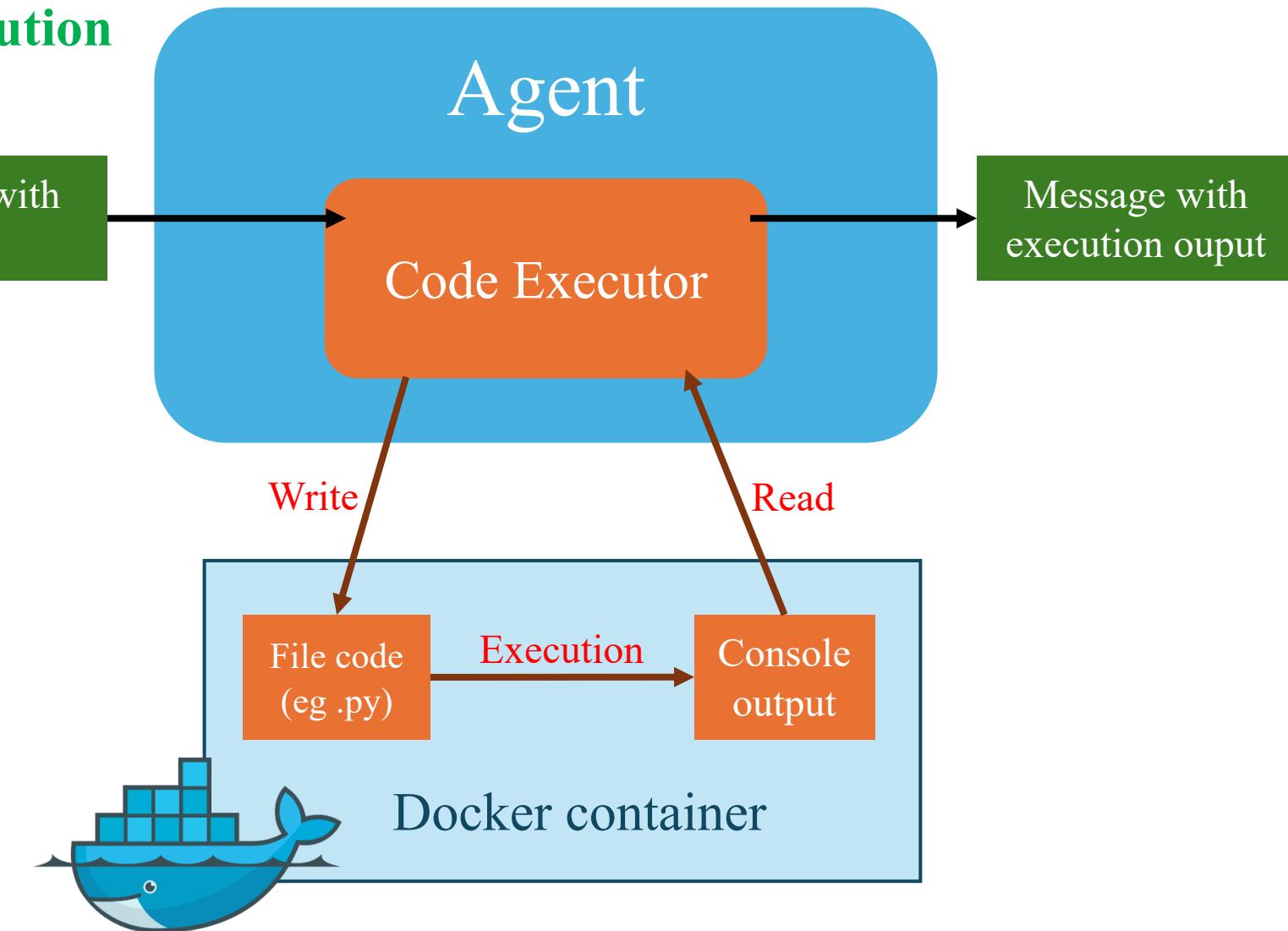
❖ Code Executor: Local Execution



Multi-agent LLMs with Autogen

❖ Code Executor: Docker Execution

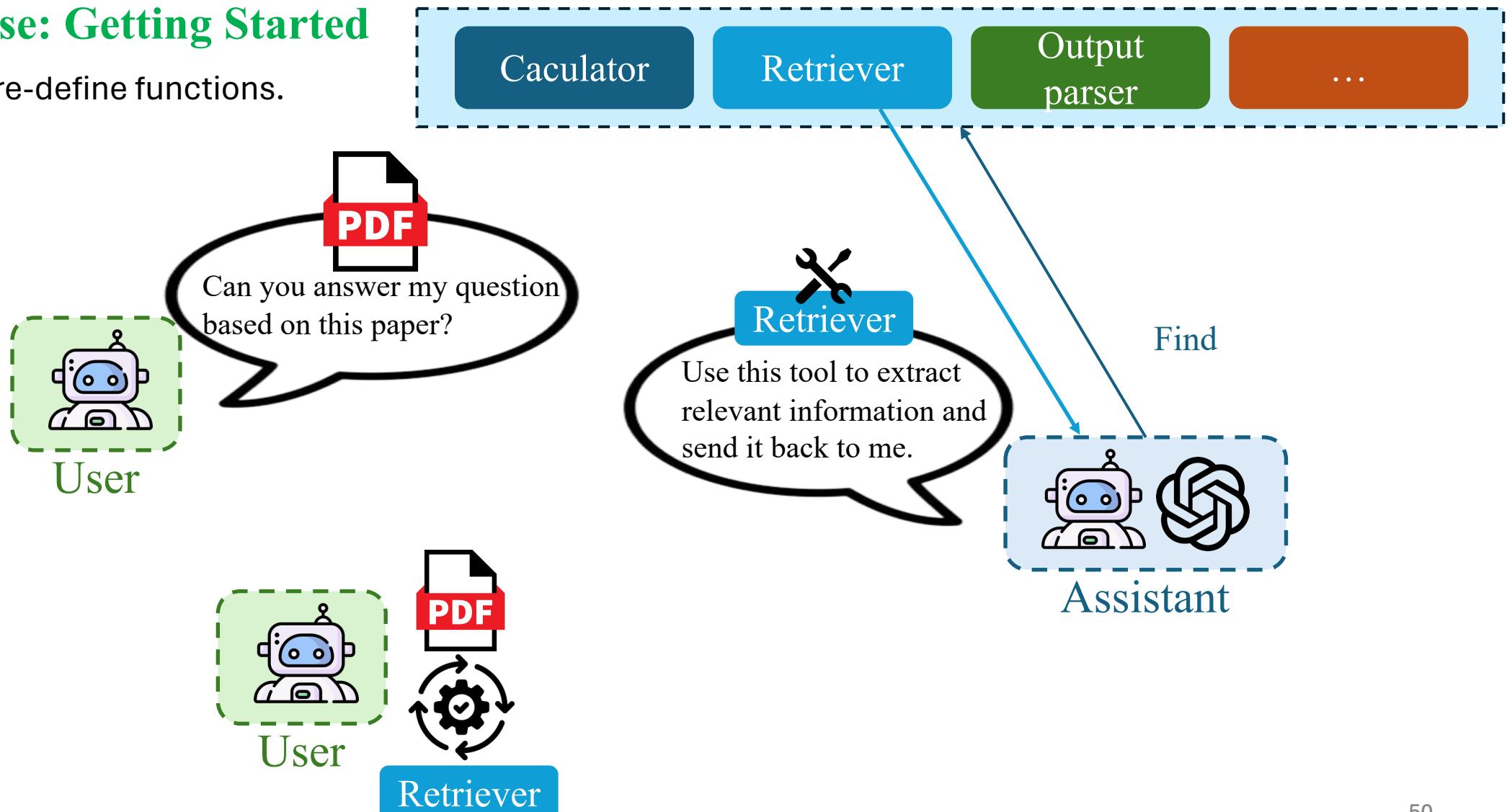
- Execute code generated by LLM in docker container.



Multi-agent LLMs with Autogen

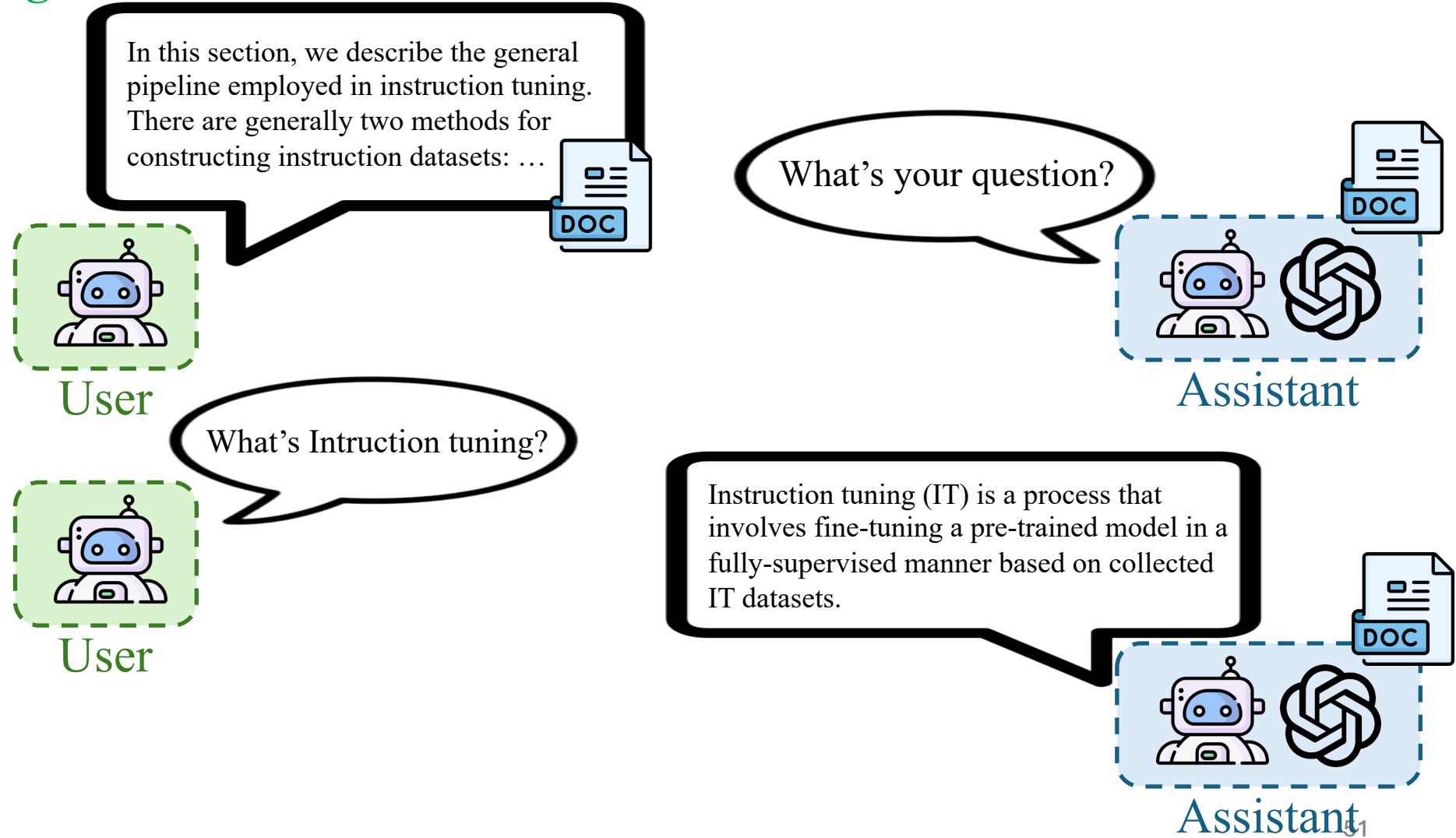
❖ Tool use: Getting Started

Tools are pre-define functions.



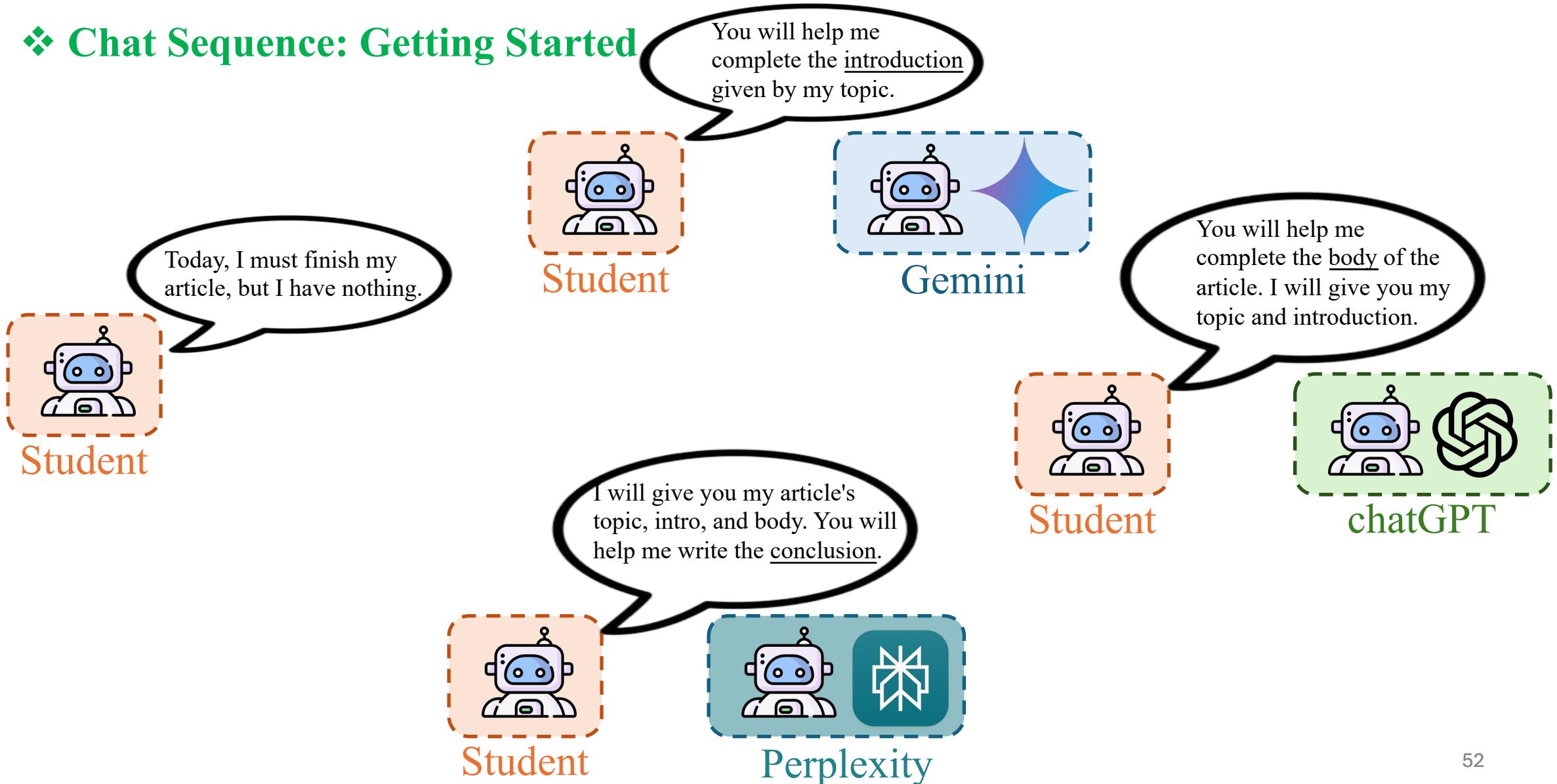
Multi-agent LLMs with Autogen

❖ Tool use: Getting Started



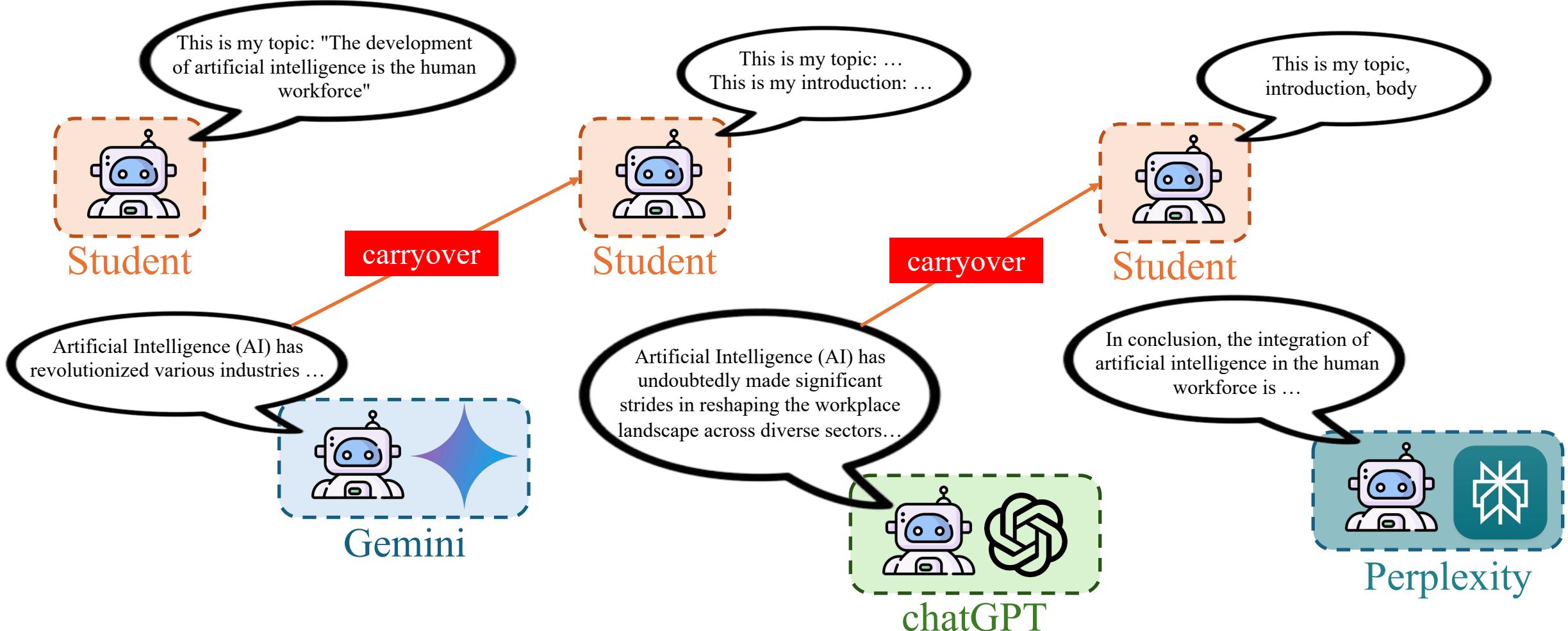
Multi-agent LLMs with Autogen

❖ Chat Sequence: Getting Started



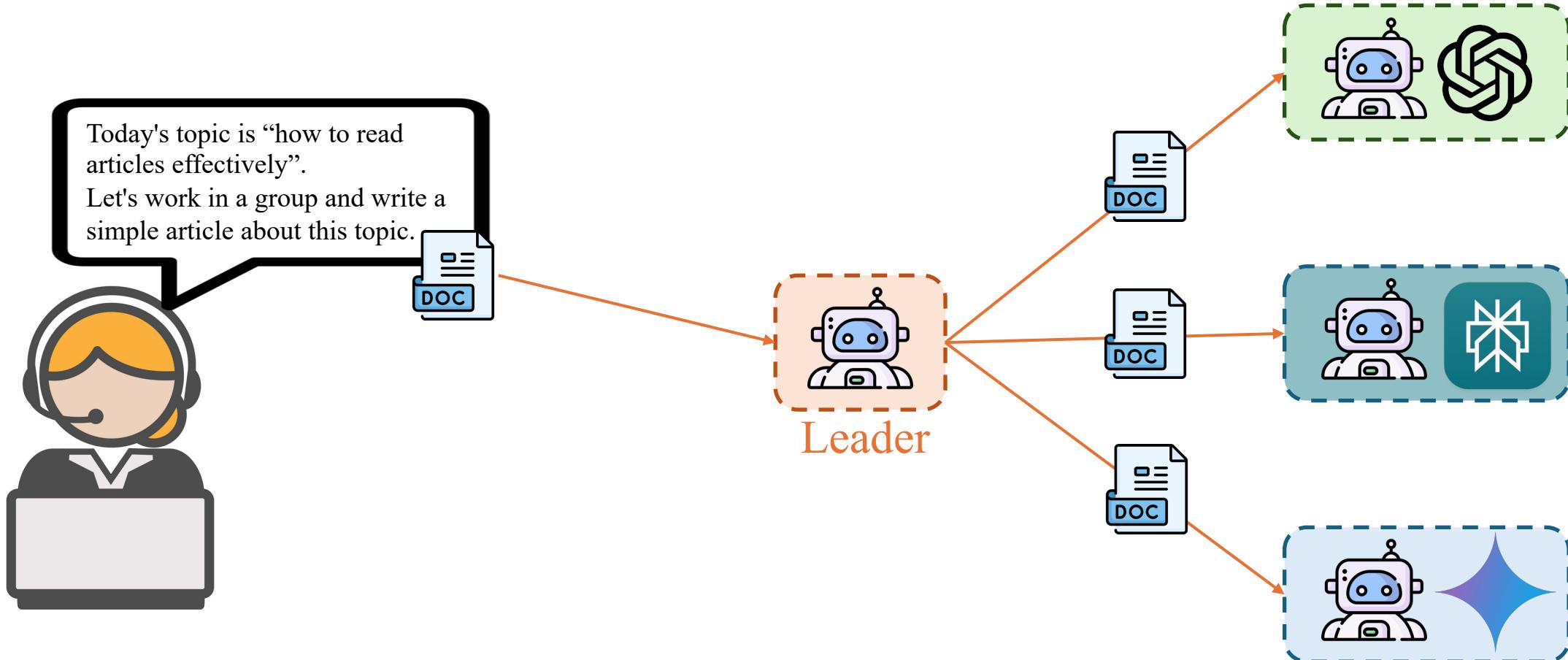
Multi-agent LLMs with Autogen

❖ Chat Sequence: Getting Started



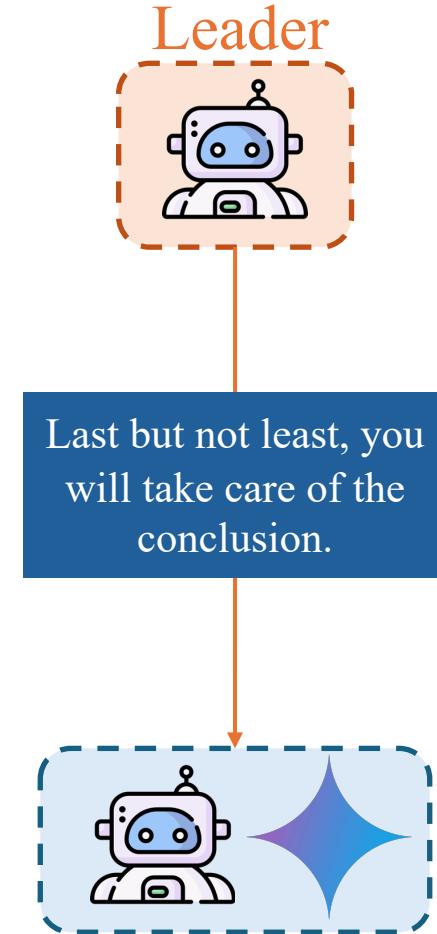
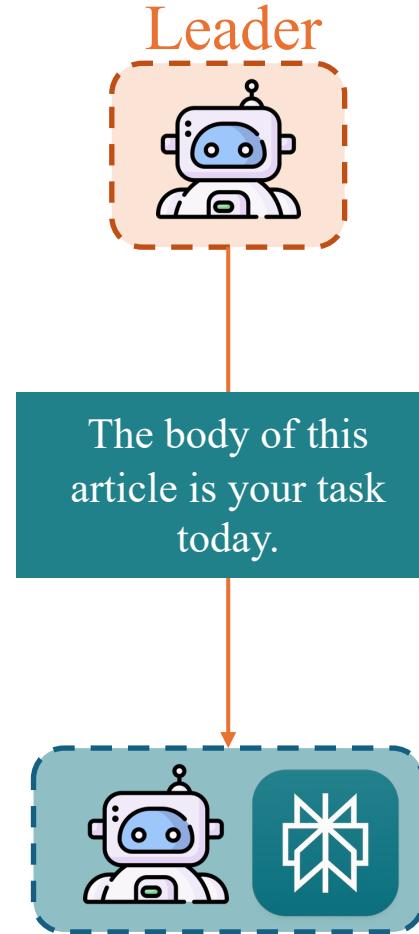
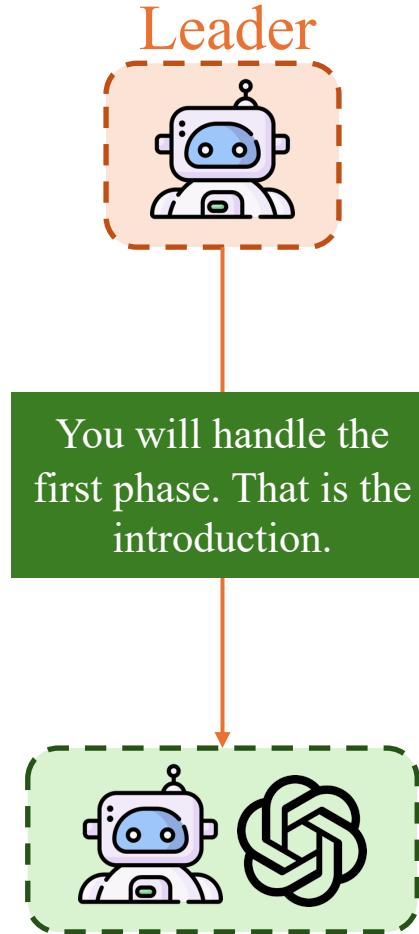
Multi-agent LLMs with Autogen

❖ Group Chat: Getting Started



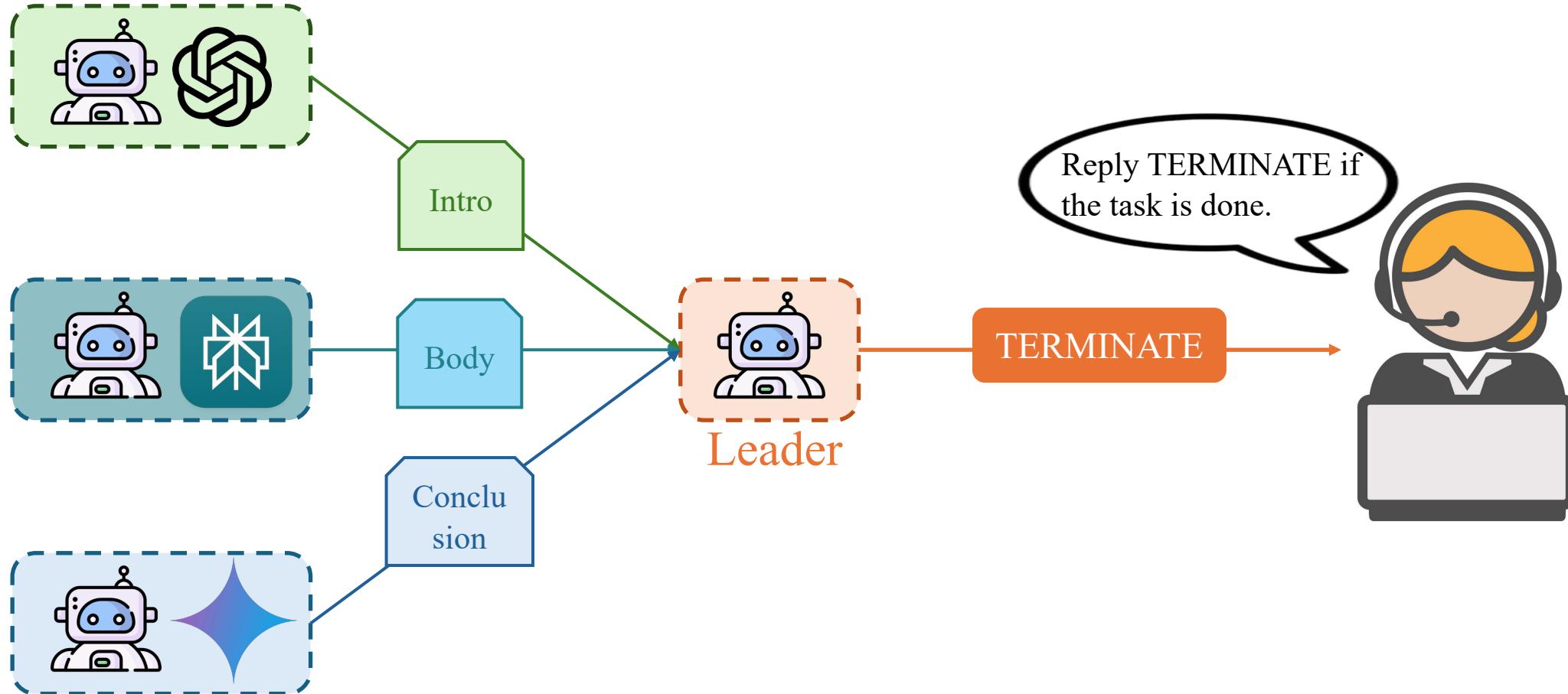
Multi-agent LLMs with Autogen

❖ Group Chat



Multi-agent LLMs with Autogen

❖ Group Chat



Multi-agent LLMs with Autogen

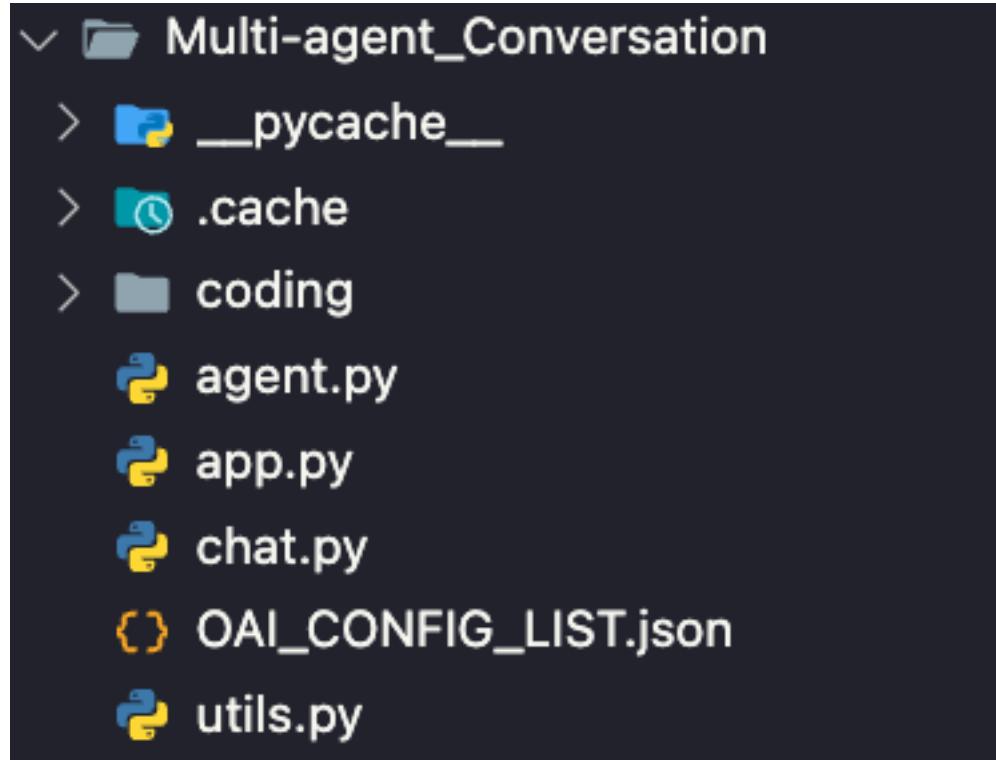
❖ Build application step 1: Organize source code

```
multiagent_llms/
  ├── agent.py
  ├── app.py
  ├── chat.py
  ├── utils.py
  └── OAI_CONFIG_LIST.json
  └── requirements.txt
```

- **requirements.txt**: Text file containing required libraries to run the source code.
- **agent.py**: Python file containing code for defining function that can create agents.
- **OAI_CONFIG_LIST.json**: Json file containing API KEY of the OpenAI GPT.
- **utils.py**: Python file containing code for threading.
- **chat.py**: Python file containing code for defining chatbox between two agents.
- **app.py**: Python file containing code for deploying.

Multi-agent LLMs with Autogen

❖ Build application step 1: Organize source code



We build this API in local computer with the following source code structure

Multi-agent LLMs with Autogen

❖ Build application step 2: Define requirements.txt



```
1 pyautogen==0.2.27  
2 gradio==4.29.0
```



```
1 pip install -r requirements.txt
```

Multi-agent LLMs with Autogen

❖ Build application step 3: Update agent.py

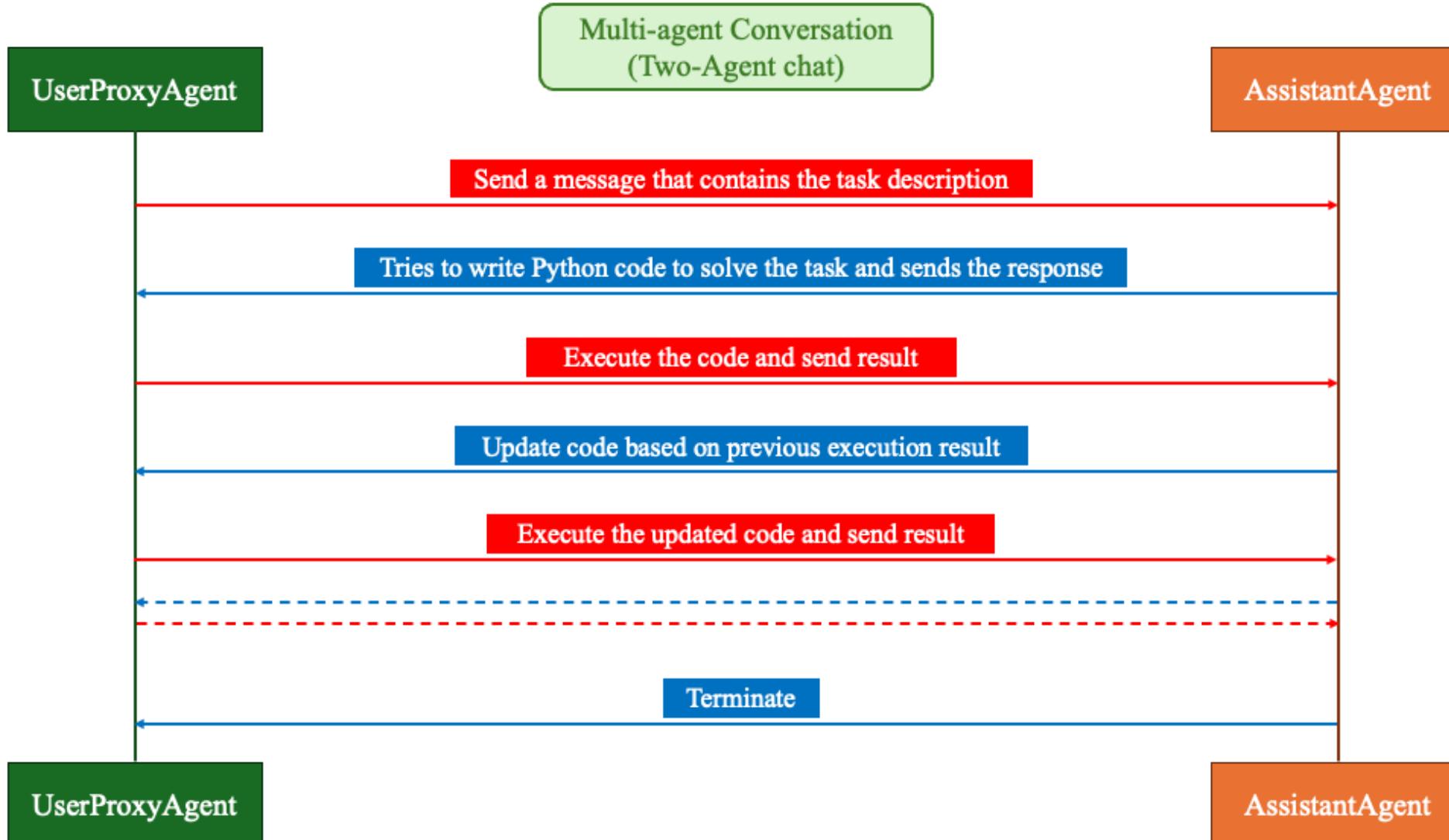
```
1 import autogen
2 from autogen import AssistantAgent, UserProxyAgent
3 from autogen.code_utils import extract_code
4
5 TIMEOUT = 60
6
7 config_list = autogen.config_list_from_json(
8     "./OAI_CONFIG_LIST.json"
9 )
```

```
{} OAI_CONFIG_LIST.json ×
ThangDuongTeam > thuannnd > Multi-Agents-LLM >
1 [
2   {
3     "model": "gpt-3.5-turbo",
4     "api_key": "API_KEY"
5   }
6 ]
```

Load OpenAI model configuration

Multi-agent LLMs with Autogen

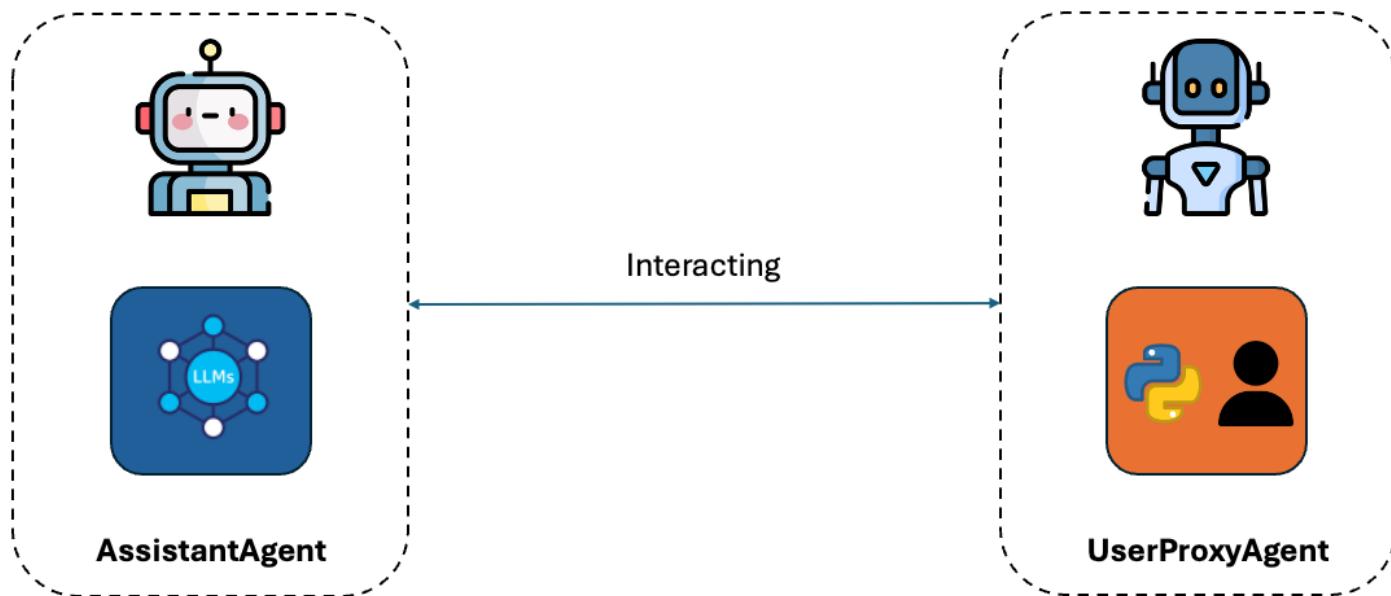
❖ Build application step 3: Update agent.py



Multi-agent LLMs with Autogen

❖ Build application step 3: Update agent.py

```
●●●  
10 def initialize_agents():  
11     assistant = AssistantAgent(  
12         name="assistant",  
13         max_consecutive_auto_reply=5,  
14         llm_config={  
15             "timeout": TIMEOUT,  
16             "config_list": config_list,  
17         },  
18     )  
19     userproxy = UserProxyAgent(  
20         name="userproxy",  
21         human_input_mode="NEVER",  
22         is_termination_msg=_is_termination_msg,  
23         max_consecutive_auto_reply=5,  
24         code_execution_config={  
25             "work_dir": "coding",  
26             "use_docker": False,  
27         },  
28     ),  
29     )  
30  
31     return assistant, userproxy
```

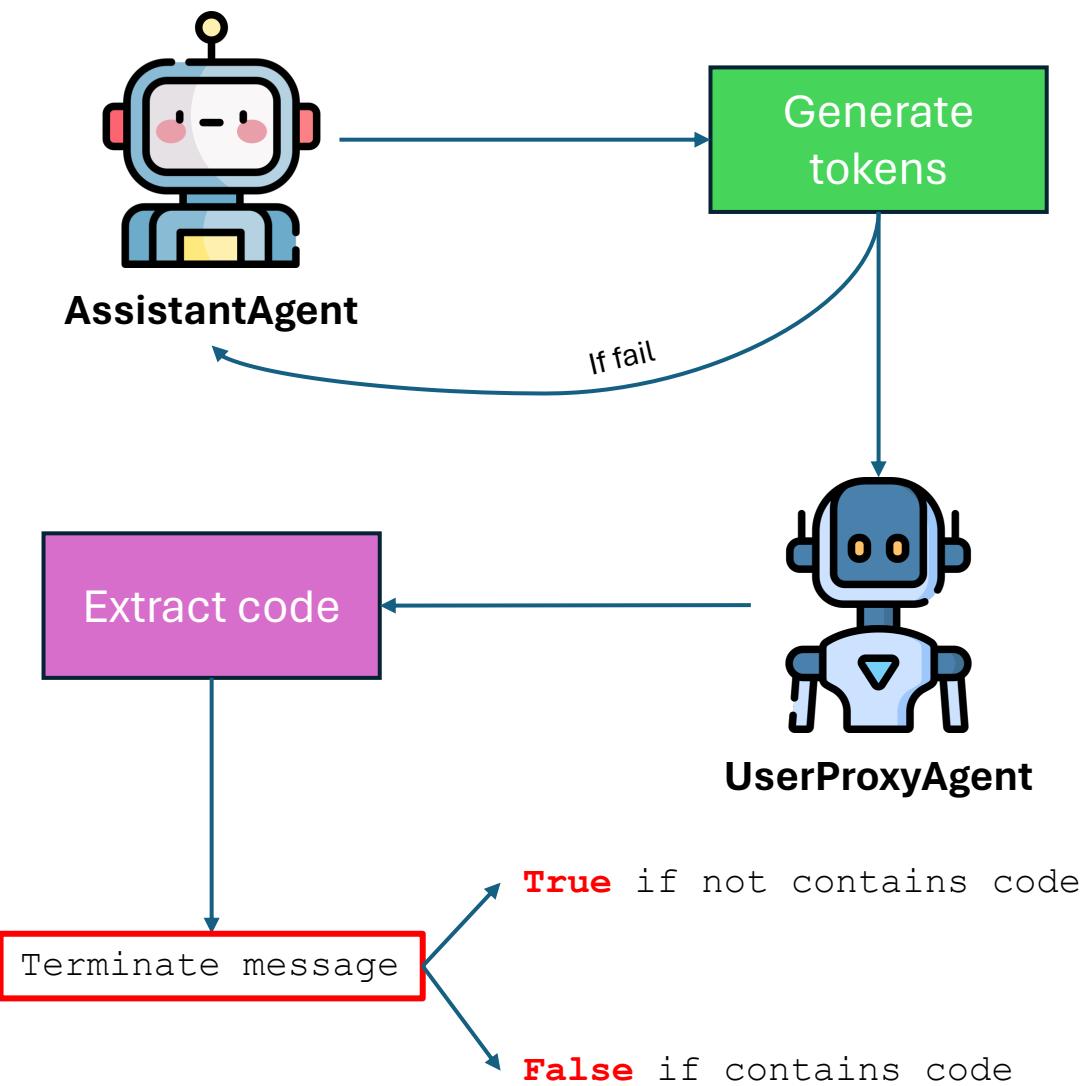


Multi-agent LLMs with Autogen

❖ Build application step 3: Update agent.py

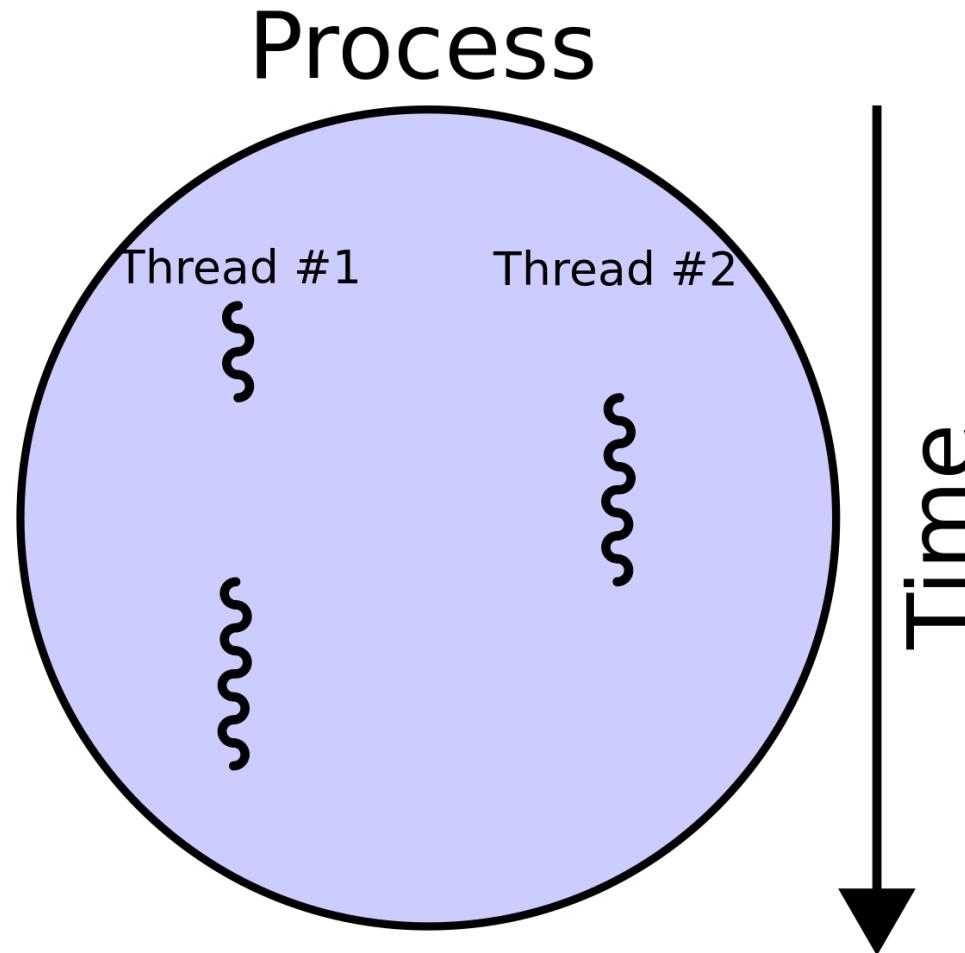


```
33 def _is_termination_msg(message):  
34     if isinstance(message, dict):  
35         message = message.get("content")  
36     if message is None:  
37         return False  
38     cb = extract_code(message)  
39     contain_code = False  
40     for c in cb:  
41         if c[0] == "python":  
42             contain_code = True  
43             break  
44     return not contain_code
```



Multi-agent LLMs with Autogen

- ❖ Build application step 4: Update utils.py



Multi-agent LLMs with Autogen

❖ Build application step 4: Update utils.py



```
1 import sys
2 import threading
3
4 LOG_LEVEL = "INFO"
5 TIMEOUT = 60
6
7 class thread_with_trace(threading.Thread):
8     def __init__(self, *args, **keywords):
9         threading.Thread.__init__(self, *args, **keywords)
10        self.killed = False
11        self._return = None
12
13    def start(self):
14        self.__run_backup = self.run
15        self.run = self.__run
16        threading.Thread.start(self)
17
18    def __run(self):
19        sys.settrace(self.globaltrace)
20        self.__run_backup()
21        self.run = self.__run_backup
```

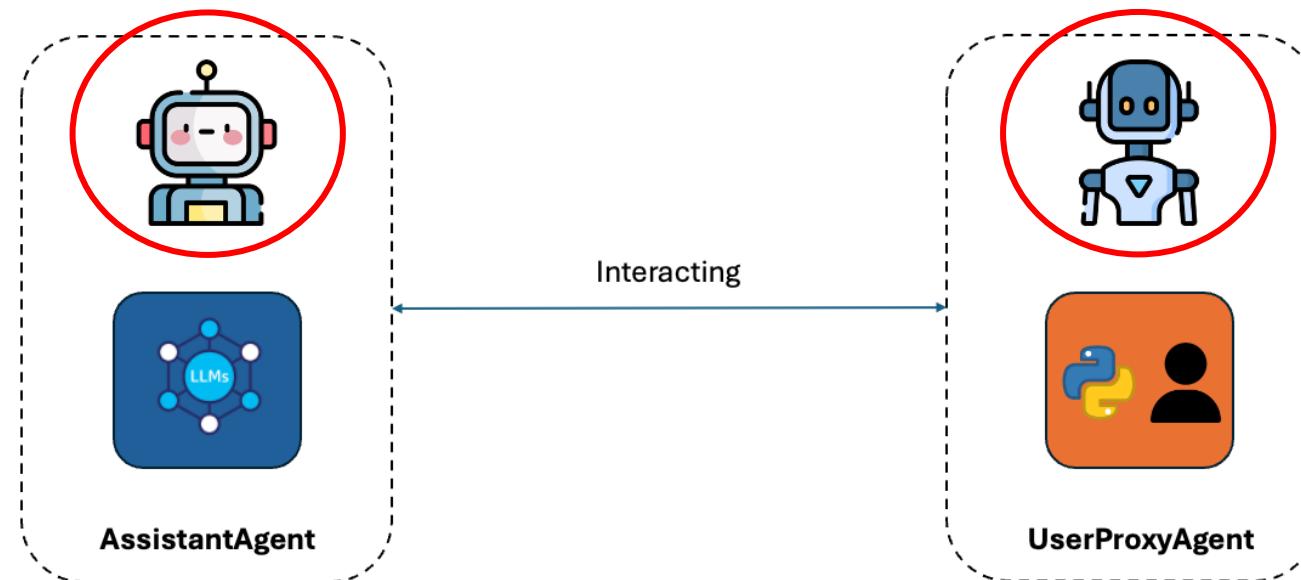


```
22    def run(self):
23        if self._target is not None:
24            self._return = self._target(*self._args, **self._kwargs)
25
26    def globaltrace(self, frame, event, arg):
27        if event == "call":
28            return self.localtrace
29        else:
30            return None
31
32    def localtrace(self, frame, event, arg):
33        if self.killed:
34            if event == "line":
35                raise SystemExit()
36            return self.localtrace
37
38    def kill(self):
39        self.killed = True
40
41    def join(self, timeout=0):
42        threading.Thread.join(self, timeout)
43        return self._return
```

Multi-agent LLMs with Autogen

❖ Build application step 5: Update chat.py

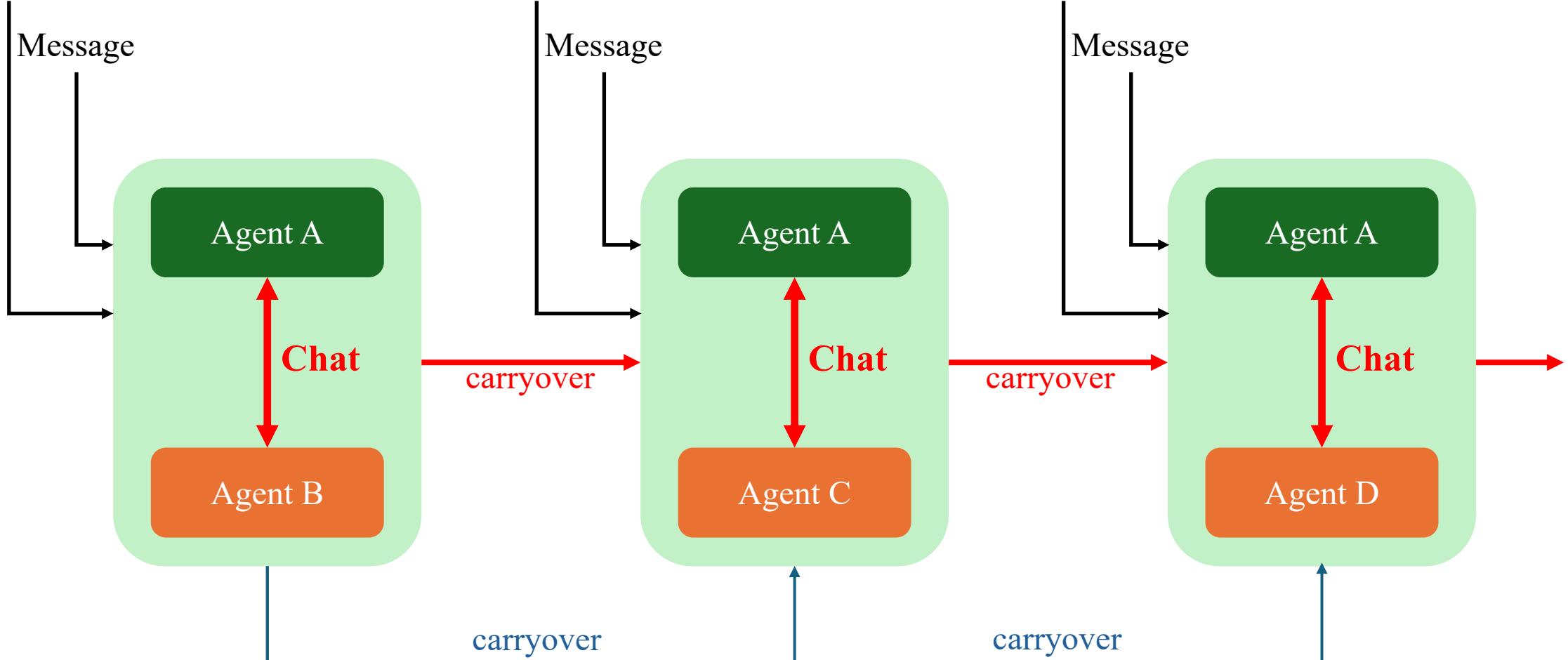
```
 1 import os
 2 from autogen import OpenAIWrapper
 3 from agent import initialize_agents, config_list
 4 from utils import *
 5
 6 assistant, userproxy = initialize_agents()
```



Multi-agent LLMs with Autogen

❖ Build application step 5: Update chat.py

Context



Multi-agent LLMs with Autogen

❖ Build application step 5: Update chat.py



```
6 def chat_to_oai_message(chat_history):
7     """Convert chat history to OpenAI message format."""
8     messages = []
9     if LOG_LEVEL == "DEBUG":
10         print(f"chat_to_oai_message: {chat_history}")
11     for msg in chat_history:
12         messages.append(
13             {
14                 "content": msg[0].split()[0] if
15                 msg[0].startswith("exitcode") else msg[0],
16                 "role": "user",
17             }
18         messages.append({"content": msg[1], "role": "assistant"})
19     return messages
```

Since we are using OpenAI GPT model, we need to convert into its format

SYSTEM

Based on the chat history, please rephrase the final user's question into a standalone question.

USER What's the Japanese name of a process of premixing yeast into a warmed slurry for soft bread?

ASSISTANT The process is called "Yudane".

USER What about it, chemically, causes softer bread?

ASSISTANT The Yudane method gelatinizes the starches in the flour when it's mixed with boiling water. This gelatinization retains more water during baking, resulting in a moister, softer bread with a longer shelf life.

USER Is adding yeast at the first stage important?

ASSISTANT No, yeast is not added in the first stage of the Yudane method. The initial step involves mixing flour with boiling water. Yeast is typically added later in the process when the dough is being prepared.

USER What is the English word for a similar breadmaking technique?

ASSISTANT The English term for a similar breadmaking technique is "Tangzhong" or "water roux".

USER How long should the yudane premix rest before being incorporated?

ASSISTANT How long should the Yudane premix rest before being incorporated into the bread dough?

Multi-agent LLMs with Autogen

❖ Build application step 5: Update chat.py

The screenshot shows a dark-themed chatbot interface. At the top, a user message reads: "write a python function that print \"Hello \" + user_name parameter". Below it, the bot responds with a Python code snippet:

```
def greet_user(user_name):
    print("Hello " + user_name)

# Example usage:
user_name = "Alice"
greet_user(user_name)
```

Below the code, the bot provides instructions: "You can call this function with the desired user name as the argument. This will print "Hello Alice" in this case. If you want to use a different name, you can replace "Alice" with the desired name when calling the function." A small robot icon is next to this text.

Further down, the bot says: "You can save this code in a Python script and run it to see the greeting message. If you have any specific user name you want to use, please provide it for demonstration." A small robot icon is next to this text.

At the bottom right, there is a terminal-like window showing the output of running the code: "exitcode: 0 (execution succeeded)", "Code output:", and "Hello Alice". A small person icon is next to this window.

We also need to get back the chat history to show the conversation in the chatbox

Multi-agent LLMs with Autogen

❖ Build application step 5: Update chat.py

```
●●●

20 def oai_message_to_chat(oai_messages, sender):
21     """Convert OpenAI message format to chat history."""
22     chat_history = []
23     messages = oai_messages[sender]
24     if LOG_LEVEL == "DEBUG":
25         print(f"oai_message_to_chat: {messages}")
26     for i in range(0, len(messages), 2):
27         chat_history.append(
28             [
29                 messages[i]["content"],
30                 messages[i + 1]["content"] if i + 1 < len(messages)
31             else "",
32         )
33     return chat_history
```

We also need to get back the chat history to show the conversation in the chatbox

Multi-agent LLMs with Autogen

❖ Build application step 5: Update chat.py

```
●●●  
34 def initiate_chat(config_list, user_message, chat_history):  
35     if LOG_LEVEL == "DEBUG":  
36         print(f"chat_history_init: {chat_history}")  
37  
38     if len(config_list[0].get("api_key", "")) < 2:  
39         chat_history.append(  
40             [  
41                 user_message,  
42             ]  
43         )  
44     return chat_history  
45 else:  
46     llm_config = {  
47         "timeout": TIMEOUT,  
48         "config_list": config_list,  
49     }  
50     assistant.llm_config.update(llm_config)  
51     assistant.client = OpenAIWrapper(**assistant.llm_config)  
52  
53     if user_message.strip().lower().startswith("show file:"):   
54         filename = user_message.strip().lower().replace("show file:", "").strip()  
55         filepath = os.path.join("coding", filename)  
56         if os.path.exists(filepath):  
57             chat_history.append([user_message, (filepath,)])  
58         else:  
59             chat_history.append([user_message, f"File {filename} not found."])  
60     return chat_history
```

```
●●●  
61     assistant.reset()  
62     oai_messages = chat_to_oai_message(chat_history)  
63     assistant._oai_system_message_origin = assistant._oai_system_message.copy()  
64     assistant._oai_system_message += oai_messages  
65  
66     try:  
67         userproxy.initiate_chat(assistant, message=user_message)  
68         messages = userproxy.chat_messages  
69         chat_history += oai_message_to_chat(messages, assistant)  
70     except Exception as e:  
71         chat_history.append([user_message, str(e)])  
72  
73     assistant._oai_system_message = assistant._oai_system_message_origin.copy()  
74     if LOG_LEVEL == "DEBUG":  
75         print(f"chat_history: {chat_history}")  
76     return chat_history
```

Function to initialize a chat between agents

Multi-agent LLMs with Autogen

❖ Build application step 5: Update chat.py

```
● ● ●

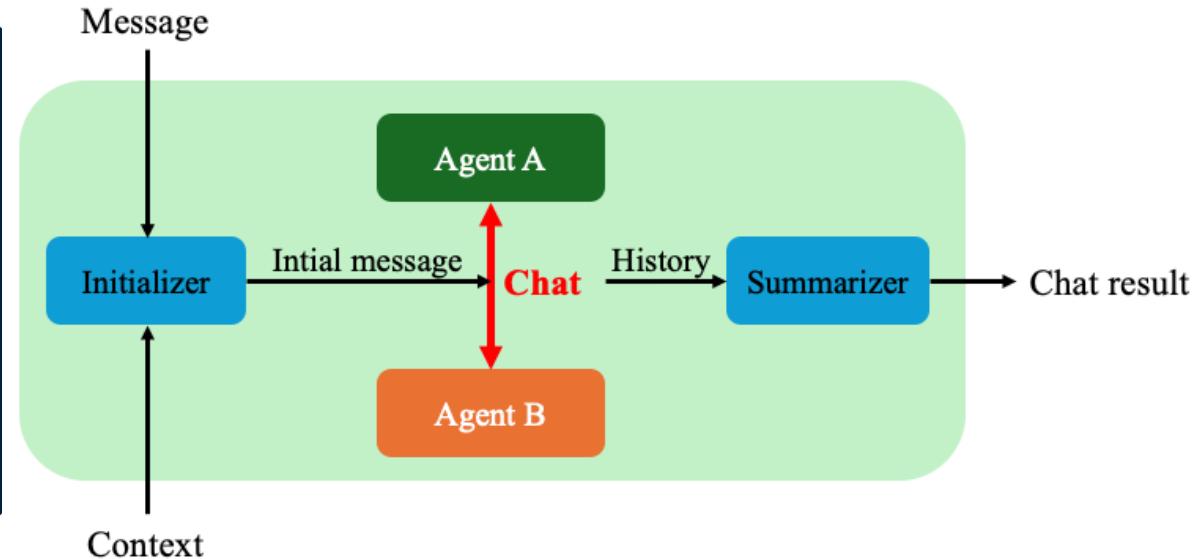
77 def chatbot_reply_thread(input_text, chat_history, config_list):
78     """Chat with the agent through terminal."""
79     thread = thread_with_trace(target=initiate_chat, args=(config_list, input_text, chat_history))
80     thread.start()
81     try:
82         messages = thread.join(timeout=TIMEOUT)
83         if thread.is_alive():
84             thread.kill()
85             thread.join()
86         messages = [
87             input_text,
88             "Timeout Error: Please check your API keys and try again later.",
89         ]
90     except Exception as e:
91         messages = [
92             [
93                 input_text,
94                 str(e) if len(str(e)) > 0 else "Invalid Request to OpenAI, please check your API keys.",
95             ]
96         ]
97     return messages
```

Multi-agent LLMs with Autogen

❖ Build application step 5: Update chat.py

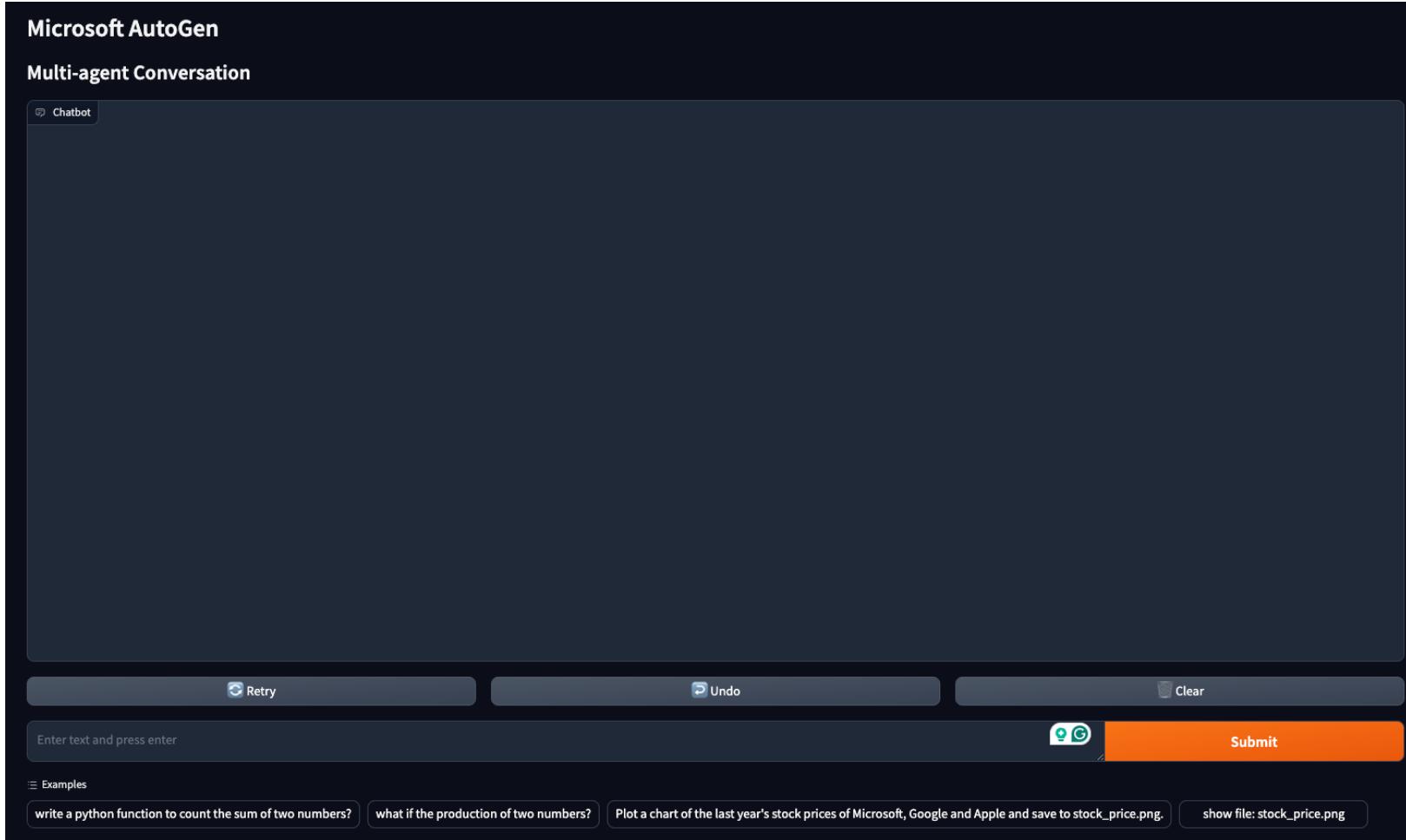
• • •

```
98 def chatbot_reply(input_text, chat_history, config_list):
99     """Chat with the agent through terminal."""
100    return chatbot_reply_thread(input_text, chat_history, config_list)
101
102 def chat_respond(message, chat_history):
103     chat_history[:] = chatbot_reply(message, chat_history, config_list)
104     if LOG_LEVEL == "DEBUG":
105         print(f"return chat_history: {chat_history}")
106     return "
```



Multi-agent LLMs with Autogen

❖ Build application step 6: Update app.py



Create a Chatbox Interface

Multi-agent LLMs with Autogen

❖ Build application step 6: Update app.py



```
1 import gradio as gr
2 from gradio import ChatInterface
3
4 from chat import chat_respond
5
6 LOG_LEVEL = "INFO"
7
8 with gr.Blocks() as demo:
9
10     description = gr.Markdown("""
11         # Microsoft AutoGen
12         ## Multi-agent Conversation
13     """)
14
15     chatbot = gr.Chatbot(
16         [],
17         elem_id="chatbot",
18         bubble_full_width=False,
19         avatar_images=(
20             "../images/human.png",
21             "../images/autogen.png",
22         ),
23         render=False,
24         height=600,
25     )
```



```
26     txt_input = gr.Textbox(
27         scale=4,
28         show_label=False,
29         placeholder="Enter text and press enter",
30         container=False,
31         render=False,
32         autofocus=True,
33     )
34
35     chatiface = ChatInterface(
36         chat_respond,
37         chatbot=chatbot,
38         textbox=txt_input,
39         examples=[
40             ["write a python function to count the sum of two numbers?"],
41             ["what if the production of two numbers?"],
42             ["Plot a chart of the last year's stock prices of Microsoft,
43             Google and Apple and save to stock_price.png."],
44             ["show file: stock_price.png"],
45         ],
46     )
47 if __name__ == "__main__":
48     demo.launch(share=False, server_name="0.0.0.0", server_port=7868)
```

Multi-agent LLMs with Autogen

❖ Build application results

Microsoft AutoGen

Multi-agent Conversation

Chatbot

write a python function to count the sum of two numbers?

```
# filename: sum_of_two_numbers.py
def sum_of_two_numbers(num1, num2):
    return num1 + num2

# Test the function
result = sum_of_two_numbers(3, 5)
print(result)
```

exitcode: 0 (execution succeeded)
Code output:
8

TERMINATE

Retry Undo Clear

Enter text and press enter

Submit

Examples

write a python function to count the sum of two numbers? what if the production of two numbers? Plot a chart of the last year's stock prices of Microsoft, Google and Apple and save to stock_price.png. show file: stock_price.png

The screenshot shows the Microsoft AutoGen interface. At the top, it says "Microsoft AutoGen" and "Multi-agent Conversation". A "Chatbot" icon is on the left. A user message bubble contains the text "write a python function to count the sum of two numbers?". Below it, a code editor window shows a Python script with a function definition and a test call. To the right, another message bubble displays the result: "exitcode: 0 (execution succeeded)" and "Code output: 8". At the bottom, there are buttons for "TERMINATE", "Retry", "Undo", "Clear", and "Submit". There's also a text input field and a "Examples" section with several other prompts.

Summary

In this lecture, we have discussed:

1. What is LLMs agent?

1. How is it differ from general LLMs?
2. What are some challenges when deploying LLMs in production?

2. What is Multi-agent LLMs?

1. How is it differ from LLMs agent?

3. How to implement an LLMs agent application with LangChain?

1. Using retriever tool to retrieve necessary information for LLMs.
2. Using calculator tool to compute input math equation.

4. How to implement an Multi-agent LLMs application with Autogen?

1. Generate code with the help of code compiler for double check.

Question

