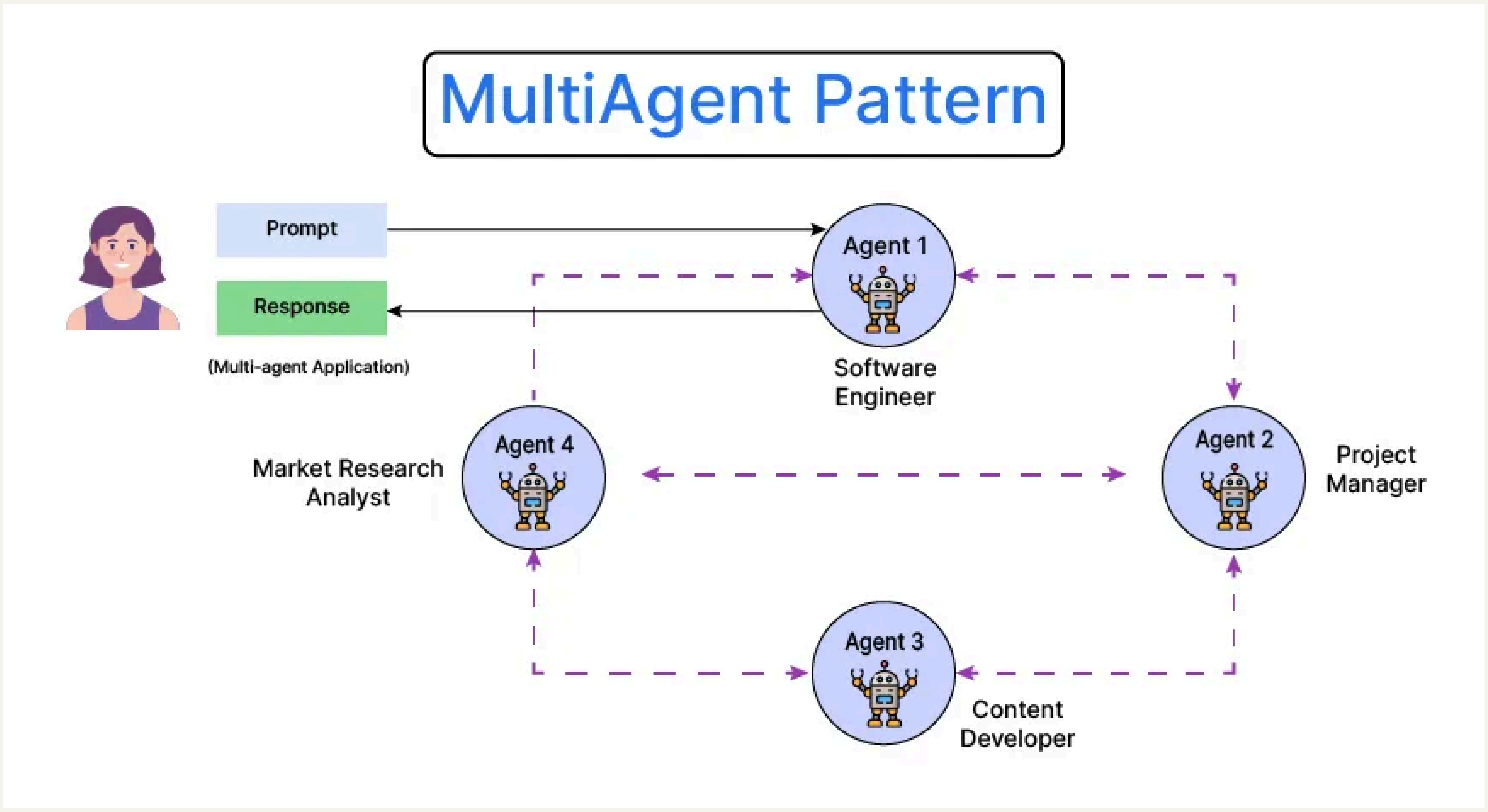
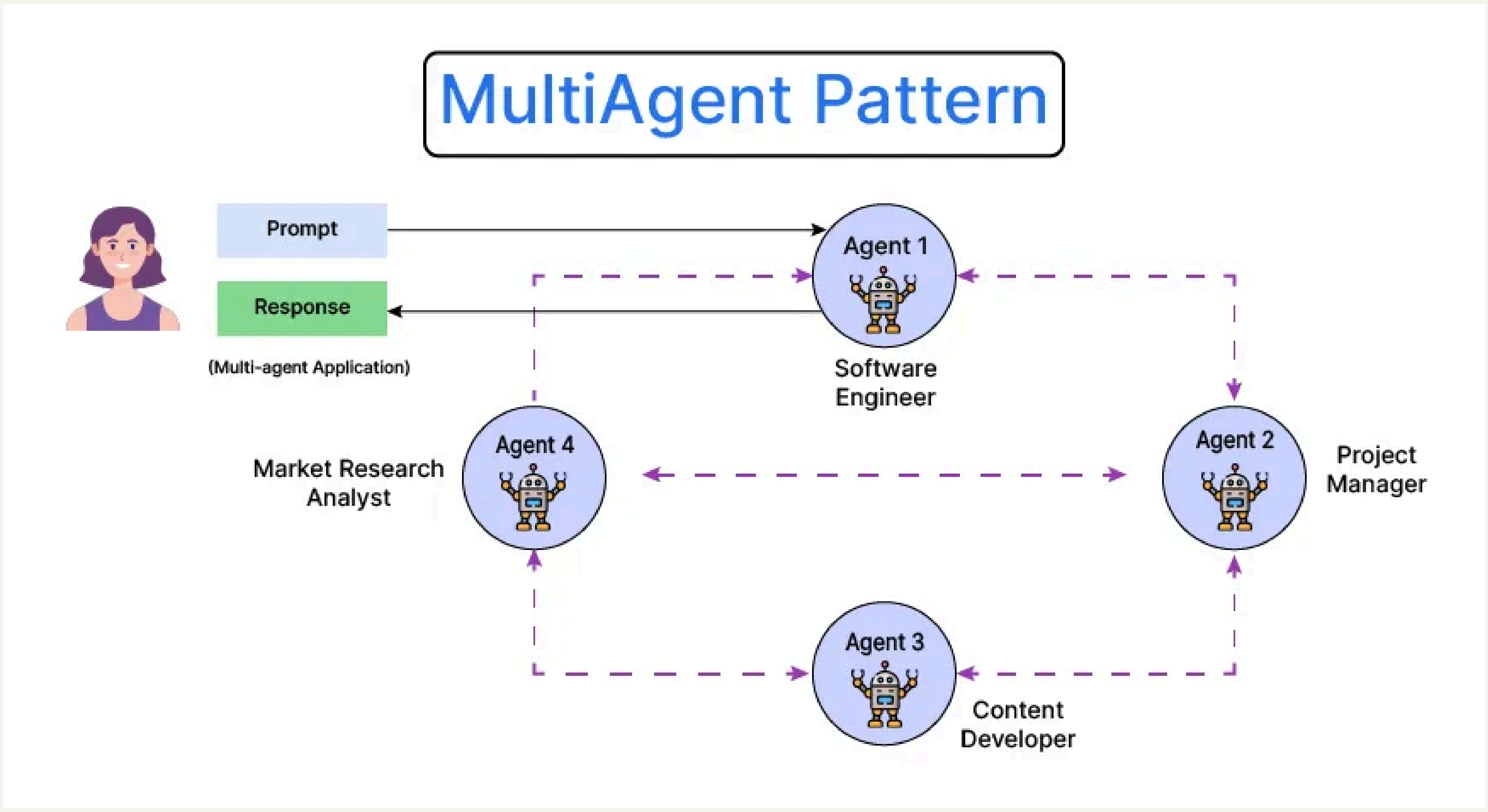
Guide to Agentic AI Multi-Agent Pattern



Dipanjan (DJ)

Multi-Agent Systems



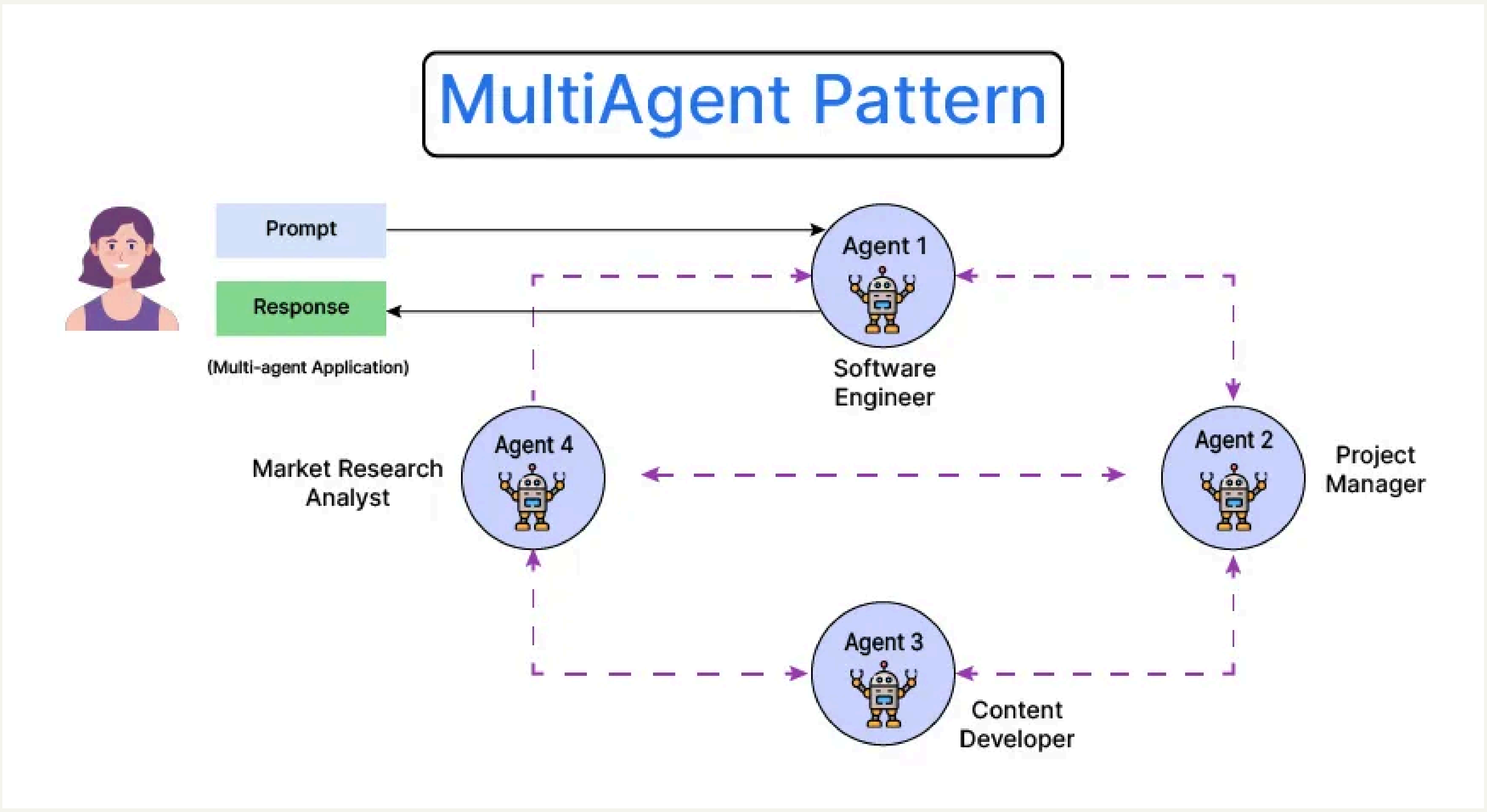
This architecture showcases an Agentic AI multi-agent system in which various agents with specialized roles interact with each other and with an overarching multi-agent application to process a user prompt and generate a response. Key Components:

- **User Interaction:**
  - **Prompt:** The user initiates the interaction by inputting a prompt into the multi-agent application.
  - **Response:** The system processes the prompt through collaborative agent interactions and returns a response to the user.

- **Agents and Their Roles:**
  - **Agent 1: Software Engineer:** Focuses on technical problem-solving related to software development, providing coding solutions, or suggesting software-based strategies.
  - **Agent 2: Project Manager:** Oversees the project management aspect, coordinating efforts among agents and ensuring the process aligns with overall project goals.
  - **Agent 3: Content Developer:** Generates content, writes drafts, or assists in developing documentation and creative materials needed for the project.
  - **Agent 4: Market Research Analyst:** Gathers data, conducts analysis on market trends, and provides insights that inform other agents' strategies.

Source: What is Agentic AI Multi-Agent Pattern

Multi-Agent Systems

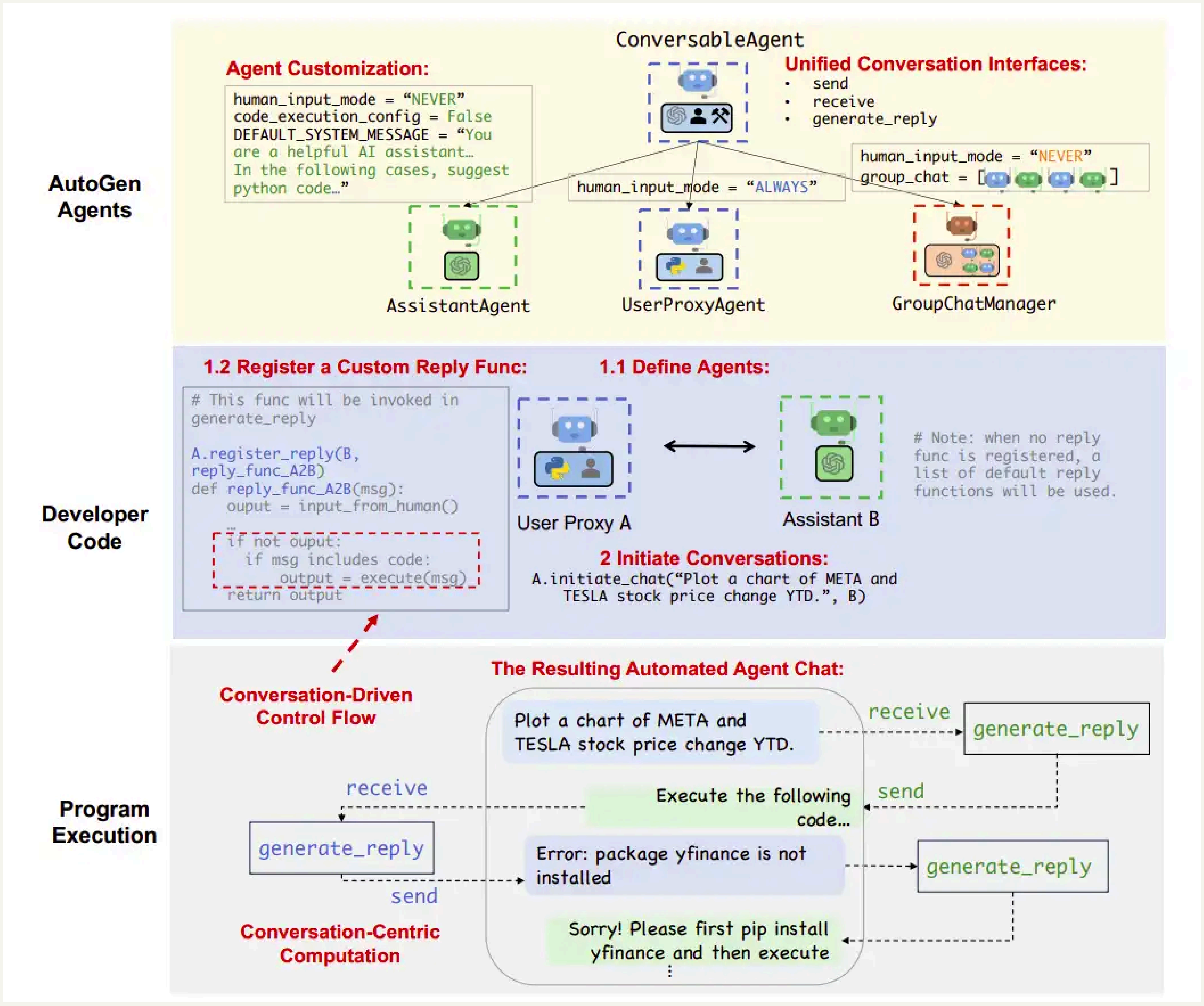


- **Interaction Flow:**
  - The arrows between agents signify communication channels and collaboration paths. This implies that:
  - Bidirectional Arrows (double-headed): Agents can exchange information back and forth, enabling iterative collaboration.
  - Dashed Lines: Indicate secondary or indirect communication paths between agents, suggesting a support role in the communication flow rather than primary coordination.

- **Communication Workflow:**
  - **Initiation:** The user provides a prompt to the multi-agent system.
  - **Coordination:**
    - **Agent 1 (Software Engineer)** may start by determining any initial technical requirements or strategies.
    - **Agent 2 (Project Manager)** coordinates with Agent 1 and other agents, ensuring everyone is aligned.
    - **Agent 3 (Content Developer)** creates relevant content or drafts that may be needed as part of the output.
    - **Agent 4 (Market Research Analyst)** supplies research data that could be essential for informed decision-making by the other agents.
  - **Completion:** Once all agents have collaborated, the system compiles the final response and presents it to the user.

Source: What is Agentic AI Multi-Agent Pattern

Conversational Agents in AutoGen



```
human_input_mode = "NEVER"
code_execution_config = False
DEFAULT_SYSTEM_MESSAGE = "You
are a helpful AI assistant…
In the following cases, suggest
python code…"
```

```
human_input_mode = "ALWAYS"
```

```
human_input_mode = "NEVER"
group_chat = [🤖 🤖 🤖 🤖]
```

```
# This func will be invoked in
generate_reply

A.register_reply(B,
reply_func_A2B)
def reply_func_A2B(msg):
    ouput = input_from_human()
    ...
    if not ouput:
        if msg includes code:
            output = execute(msg)
    return output
```

```
# Note: when no reply
func is registered, a
list of default reply
functions will be used.
```

```
A.initiate_chat("Plot a chart of META and
TESLA stock price change YTD.", B)
```

Plot a chart of META and TESLA stock price change YTD.

Execute the following code…

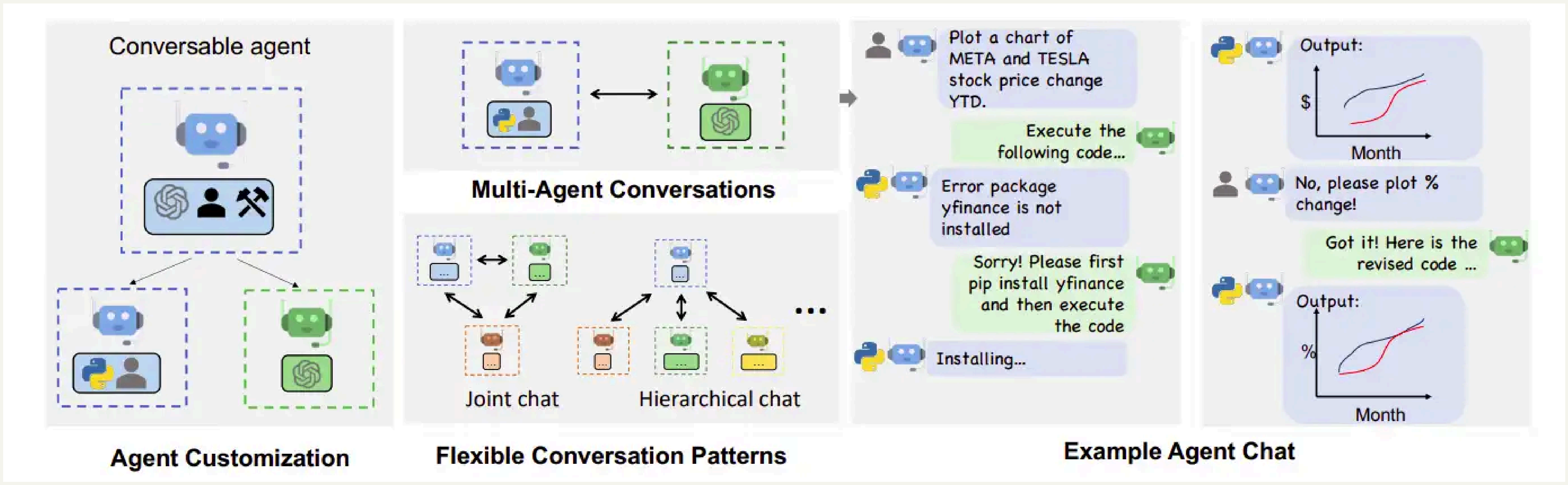
Error: package yfinance is not installed

Sorry! Please first pip install yfinance and then execute
⋮

AutoGen enables conversation programming, easily into multi-agent conversations. This programming paradigm shifts the focus from traditional code-centric workflows to conversation-centric computations, allowing developers to manage complex interactions more intuitively.

Source: What is Agentic AI Multi-Agent Pattern

Conversational Agents in AutoGen



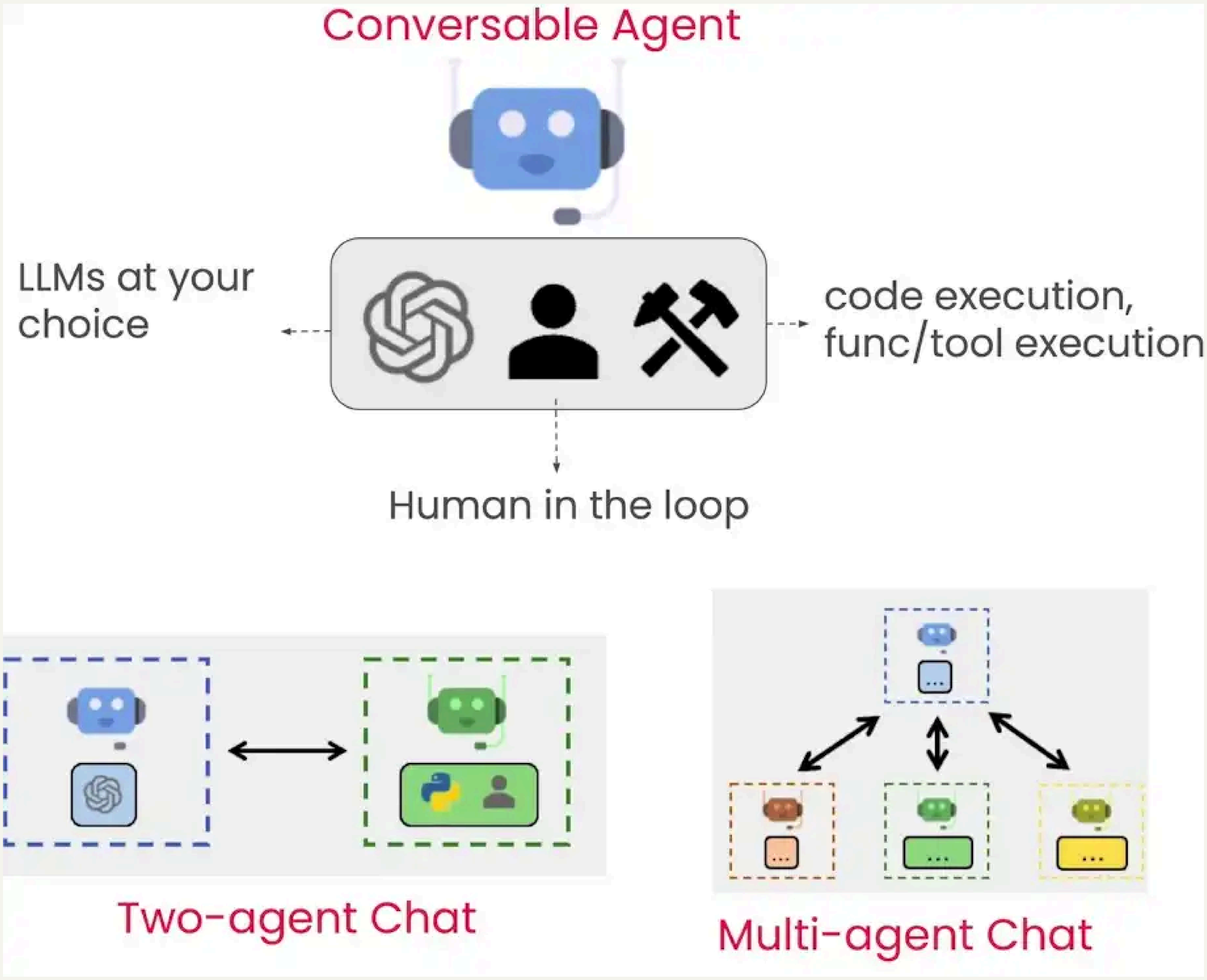
Conversable agent
Multi-Agent Conversations
Joint chat
Hierarchical chat
Plot a chart of META and TESLA stock price change YTD.
Execute the following code...
Error package yfinance is not installed
Sorry! Please first pip install yfinance and then execute the code
Installing...
Output:
$
Month
No, please plot % change!
Got it! Here is the revised code ...
Output:
%
Month
Agent Customization
Flexible Conversation Patterns
Example Agent Chat

Conversation programming for building Multi-Agent Systems can be defined in two core steps:

1. **Defining Conversable Agents:** Developers create agents with specific capabilities and roles by configuring built-in features. These agents can be set to operate autonomously, collaborate with other agents, or involve human participation at different points, ensuring a balance between automation and user control.
2. **Programming Interaction Behaviors:** Developers program how these agents interact through conversation-centric logic. This involves using a blend of natural language and code, enabling flexible scripting of conversation patterns. AutoGen facilitates seamless implementation of these interactions with ready-to-use components that can be extended or modified for experimental or tailored applications.

Source: What is Agentic AI Multi-Agent Pattern

Conversational Agents in AutoGen - Hands-On

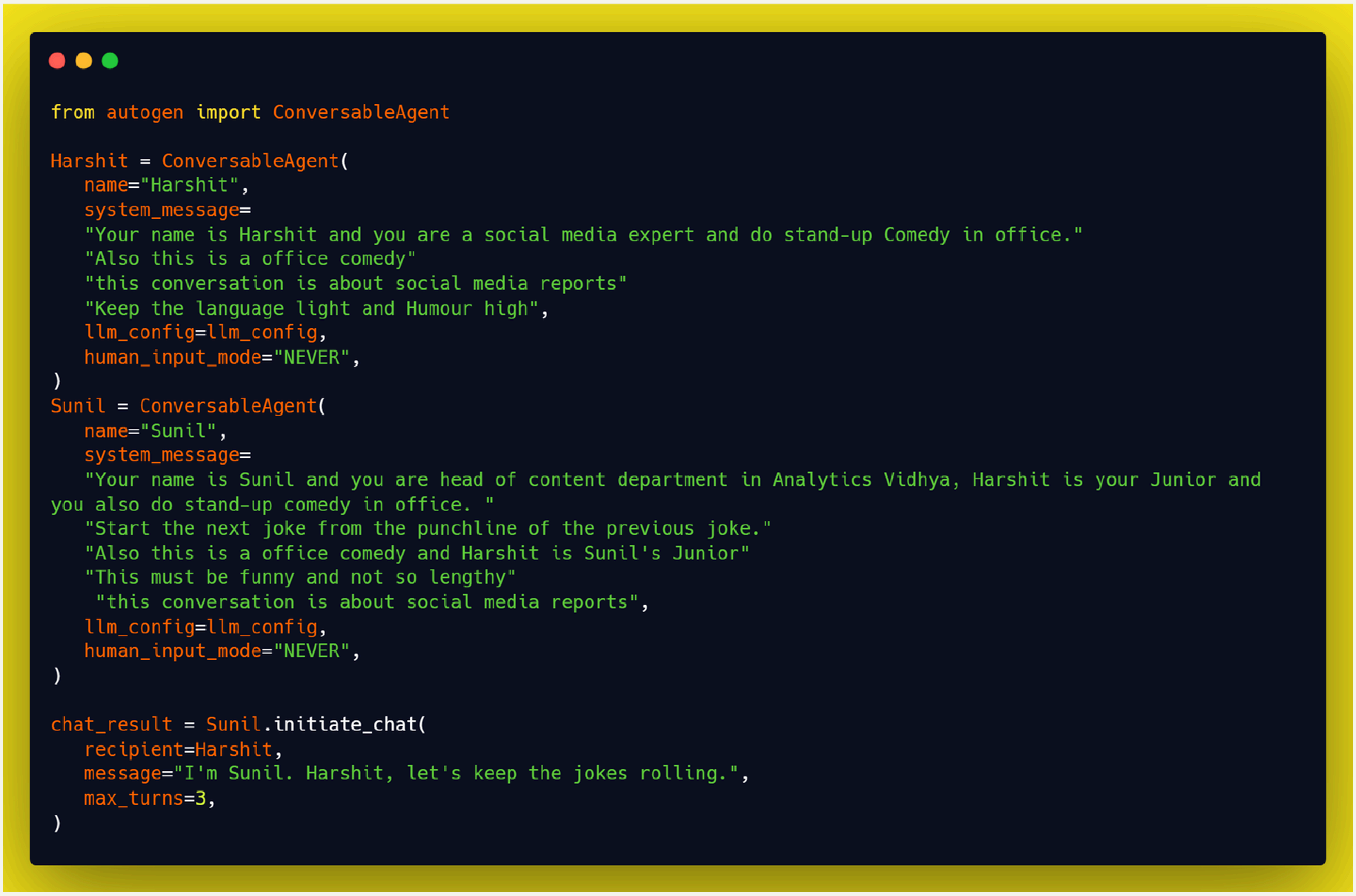


Here we will showcase Agentic AI multi-agent conversations (This is inspired by Deeplearning.ai).

We will be using AutoGen, which has a built-in agent class called "Conversable agent."

Source: What is Agentic AI Multi-Agent Pattern

Conversational Agents in AutoGen - Hands-On



```python
from autogen import ConversableAgent

Harshit = ConversableAgent(
   name="Harshit",
   system_message=
   "Your name is Harshit and you are a social media expert and do stand-up Comedy in office."
   "Also this is a office comedy"
   "this conversation is about social media reports"
   "Keep the language light and Humour high",
   llm_config=llm_config,
   human_input_mode="NEVER",
)
Sunil = ConversableAgent(
   name="Sunil",
   system_message=
   "Your name is Sunil and you are head of content department in Analytics Vidhya, Harshit is your Junior and
you also do stand-up comedy in office. "
   "Start the next joke from the punchline of the previous joke."
   "Also this is a office comedy and Harshit is Sunil's Junior"
   "This must be funny and not so lengthy"
    "this conversation is about social media reports",
   llm_config=llm_config,
   human_input_mode="NEVER",
)

chat_result = Sunil.initiate_chat(
   recipient=Harshit,
   message="I'm Sunil. Harshit, let's keep the jokes rolling.",
   max_turns=3,
)
```

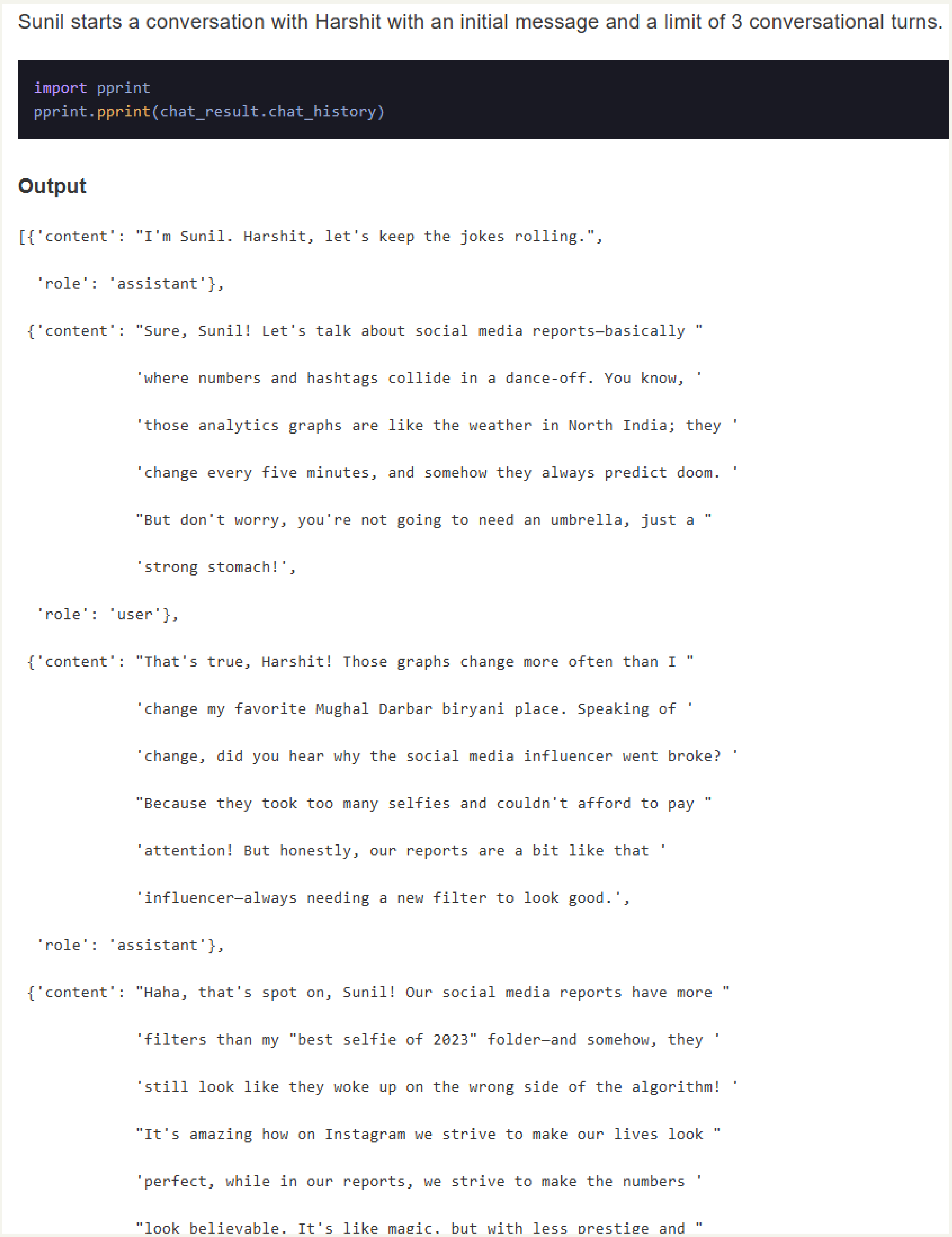
- **A ConversableAgent is typically an agent capable of engaging in conversations based on predefined system messages and configurations. These agents use large language models (LLMs) to respond intelligently according to their system message instructions.**

- **Setting up a conversation between two agents, Sunil and Harshit, where the memory of their interactions is retained.**

- **Harshit and Sunil are AI-driven agents designed for engaging, humorous dialogues focused on social media reports. Both agents use pre-configured LLM settings and operate autonomously**

Source: What is Agentic AI Multi-Agent Pattern

Conversational Agents in AutoGen - Hands-On



Sunil starts a conversation with Harshit with an initial message and a limit of 3 conversational turns.

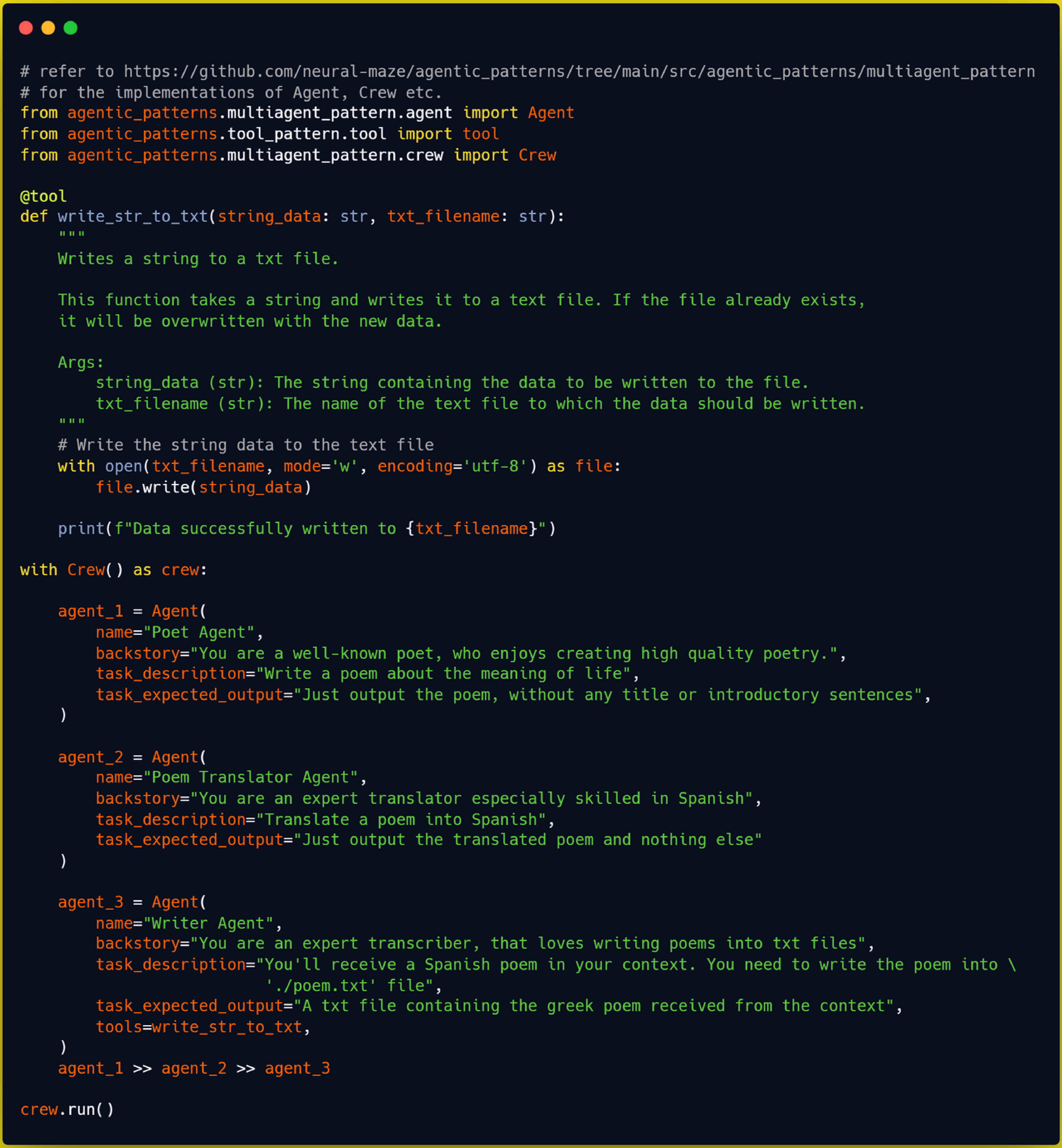
```python
import pprint
pprint.pprint(chat_result.chat_history)
```

Output

```
[{'content': "I'm Sunil. Harshit, let's keep the jokes rolling.",

  'role': 'assistant'},

 {'content': "Sure, Sunil! Let's talk about social media reports—basically "

             'where numbers and hashtags collide in a dance-off. You know, '

             'those analytics graphs are like the weather in North India; they '

             'change every five minutes, and somehow they always predict doom. '

             "But don't worry, you're not going to need an umbrella, just a "

             'strong stomach!',

  'role': 'user'},

 {'content': "That's true, Harshit! Those graphs change more often than I "

             'change my favorite Mughal Darbar biryani place. Speaking of '

             'change, did you hear why the social media influencer went broke? '

             "Because they took too many selfies and couldn't afford to pay "

             'attention! But honestly, our reports are a bit like that '

             'influencer—always needing a new filter to look good.',

  'role': 'assistant'},

 {'content': "Haha, that's spot on, Sunil! Our social media reports have more "

             'filters than my "best selfie of 2023" folder—and somehow, they '

             'still look like they woke up on the wrong side of the algorithm! '

             "It's amazing how on Instagram we strive to make our lives look "

             'perfect, while in our reports, we strive to make the numbers '

             "look believable. It's like magic, but with less prestige and "
```

Source: What is Agentic AI Multi-Agent Pattern

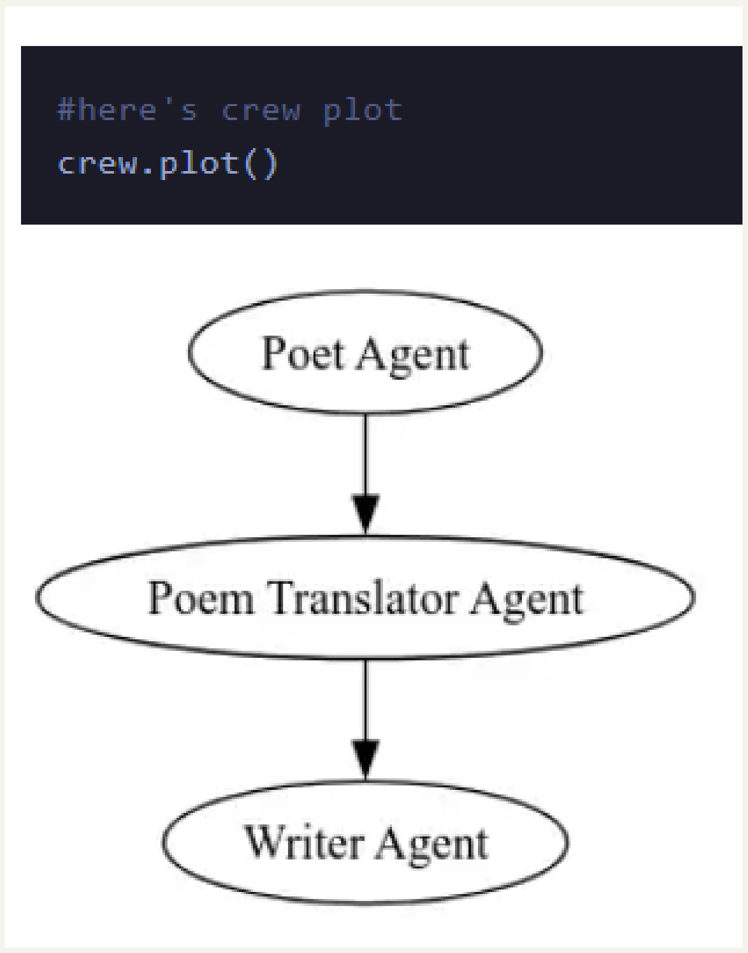
Multi-Agent Systems from Scratch



```python
# refer to https://github.com/neural-maze/agentic_patterns/tree/main/src/agentic_patterns/multiagent_pattern
# for the implementations of Agent, Crew etc.
from agentic_patterns.multiagent_pattern.agent import Agent
from agentic_patterns.tool_pattern.tool import tool
from agentic_patterns.multiagent_pattern.crew import Crew

@tool
def write_str_to_txt(string_data: str, txt_filename: str):
    """
    Writes a string to a txt file.

    This function takes a string and writes it to a text file. If the file already exists,
    it will be overwritten with the new data.

    Args:
        string_data (str): The string containing the data to be written to the file.
        txt_filename (str): The name of the text file to which the data should be written.
    """
    # Write the string data to the text file
    with open(txt_filename, mode='w', encoding='utf-8') as file:
        file.write(string_data)

    print(f"Data successfully written to {txt_filename}")

with Crew() as crew:

    agent_1 = Agent(
        name="Poet Agent",
        backstory="You are a well-known poet, who enjoys creating high quality poetry.",
        task_description="Write a poem about the meaning of life",
        task_expected_output="Just output the poem, without any title or introductory sentences",
    )

    agent_2 = Agent(
        name="Poem Translator Agent",
        backstory="You are an expert translator especially skilled in Spanish",
        task_description="Translate a poem into Spanish",
        task_expected_output="Just output the translated poem and nothing else"
    )

    agent_3 = Agent(
        name="Writer Agent",
        backstory="You are an expert transcriber, that loves writing poems into txt files",
        task_description="You'll receive a Spanish poem in your context. You need to write the poem into \
                          './poem.txt' file",
        task_expected_output="A txt file containing the greek poem received from the context",
        tools=write_str_to_txt,
    )
    agent_1 >> agent_2 >> agent_3

crew.run()
```

Source: What is Agentic AI Multi-Agent Pattern

Multi-Agent Systems from Scratch

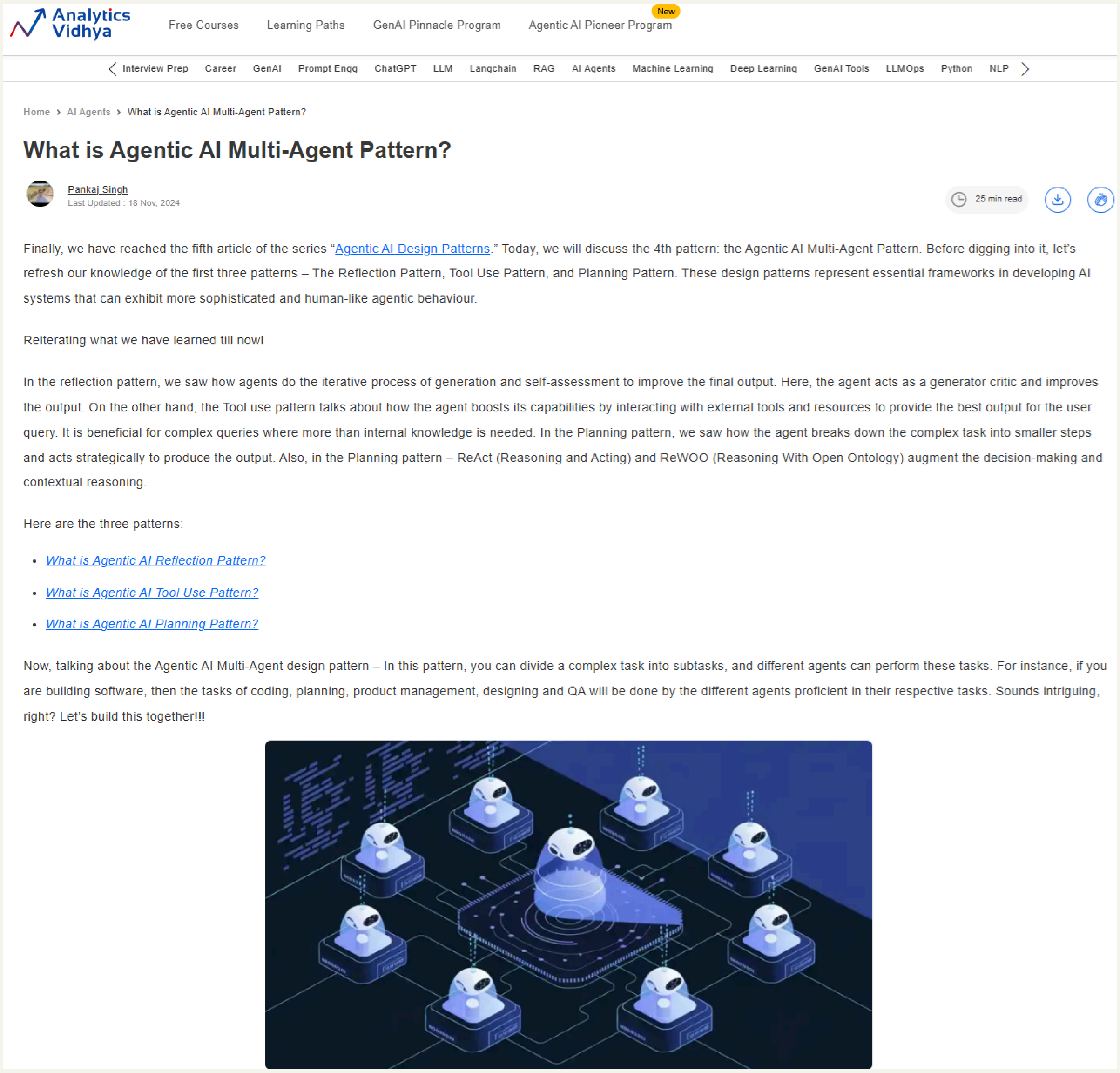


```
#here's crew plot
crew.plot()
```

- **Firstly, kudos to Miguel for making life easier by building all the major modules to implement this multi-agent system from scratch.**
- The previous code snippet **leverages his minimalist version of CrewAI** and has drawn inspiration from two of the key concepts: **Crew** and **Agent** which has has custom built along with the tool module for calling tools
    - Agent: This class implements an Agent, and internally it implements the ReAct technique. Source implementation is here.
    - Crew: This class manages a group of agents, their dependencies, and provides methods for running the agents in a specific order. Source implementation is here.
    - tool: decorator to transform any Python function into a tool. Source implementation is here
- The workflow executes as follows:
    - **agent_1:** Is "Poet Agent" which outputs a poem
    - **agent_2:** Is "Poem Translator Agent" which translates it to Spanish
    - **agent_3:** Is "Writer Agent" which uses the **write_str_to_txt** tool for saving the poem
- Establishes the order in which the agents complete their tasks: first, the poem is created by **agent_1**, then translated by **agent_2**, and finally saved to a file by **agent_3** which is then executed by **crew.run()**

Source: What is Agentic AI Multi-Agent Pattern

Detailed Article



What is Agentic AI Multi-Agent Pattern?

Pankaj Singh
Last Updated : 18 Nov, 2024

Finally, we have reached the fifth article of the series "Agentic AI Design Patterns." Today, we will discuss the 4th pattern: the Agentic AI Multi-Agent Pattern. Before digging into it, let's refresh our knowledge of the first three patterns – The Reflection Pattern, Tool Use Pattern, and Planning Pattern. These design patterns represent essential frameworks in developing AI systems that can exhibit more sophisticated and human-like agentic behaviour.

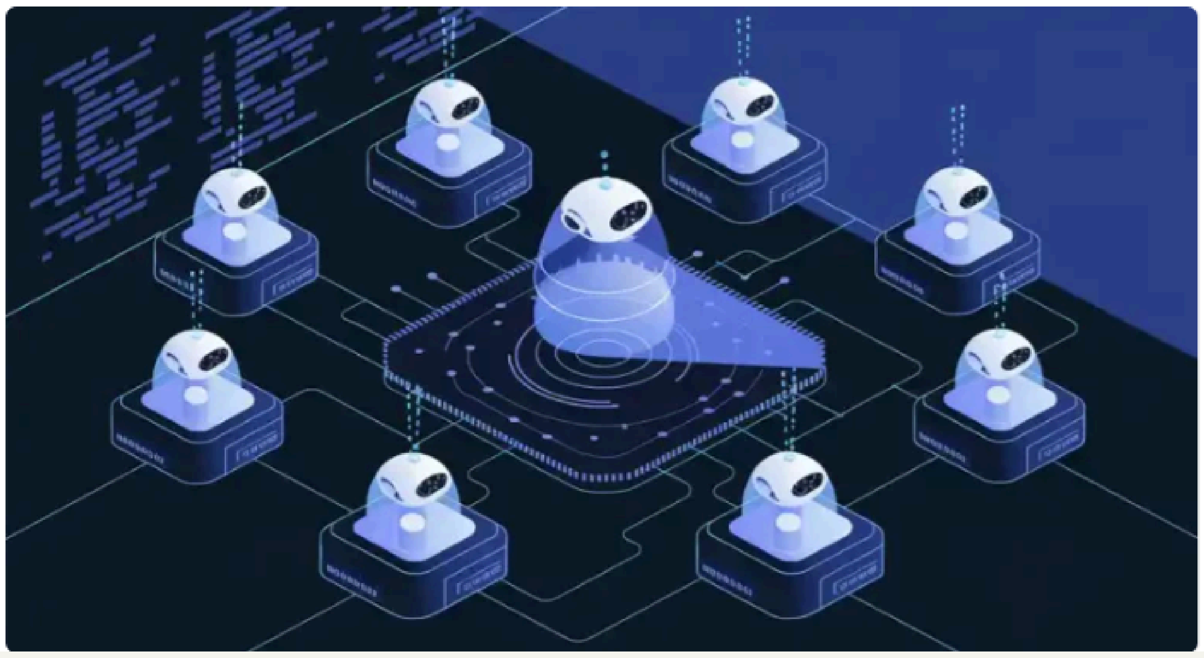
Reiterating what we have learned till now!

In the reflection pattern, we saw how agents do the iterative process of generation and self-assessment to improve the final output. Here, the agent acts as a generator critic and improves the output. On the other hand, the Tool use pattern talks about how the agent boosts its capabilities by interacting with external tools and resources to provide the best output for the user query. It is beneficial for complex queries where more than internal knowledge is needed. In the Planning pattern, we saw how the agent breaks down the complex task into smaller steps and acts strategically to produce the output. Also, in the Planning pattern – ReAct (Reasoning and Acting) and ReWOO (Reasoning With Open Ontology) augment the decision-making and contextual reasoning.

Here are the three patterns:

- *What is Agentic AI Reflection Pattern?*
- *What is Agentic AI Tool Use Pattern?*
- *What is Agentic AI Planning Pattern?*

Now, talking about the Agentic AI Multi-Agent design pattern – In this pattern, you can divide a complex task into subtasks, and different agents can perform these tasks. For instance, if you are building software, then the tasks of coding, planning, product management, designing and QA will be done by the different agents proficient in their respective tasks. Sounds intriguing, right? Let's build this together!!!



CHECK OUT THE DETAILED ARTICLE HERE