

Getting Started with AutoGen

By Lucas Soares

Lucas Soares

- AI Engineer



Lucas Soares

- ML Engineer
- Instructor at O'Reilly Media



Lucas Soares

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



Table of Contents

1. Agents as Thought + Action

Table of Contents

1. Agents as Thought + Action

2. Defining Agents

Table of Contents

- 1. Agents as Thought + Action**
- 2. Defining Agents**
- 3. Agents in 3 Levels of Complexity**

Table of Contents

- 1. Agents as Thought + Action**
- 2. Defining Agents**
- 3. Agents in 3 Levels of Complexity**
- 4. OpenAI's Function API**

Table of Contents

- 1. Agents as Thought + Action**
- 2. Defining Agents**
- 3. Agents in 3 Levels of Complexity**
- 4. OpenAI's Function API**
- 5. AutoGen Framework**

Table of Contents

- 1. Agents as Thought + Action**
- 2. Defining Agents**
- 3. Agents in 3 Levels of Complexity**
- 4. OpenAI's Function API**
- 5. AutoGen Framework**
- 6. Design Patterns in AutoGen**

Table of Contents

- 1. Agents as Thought + Action**
- 2. Defining Agents**
- 3. Agents in 3 Levels of Complexity**
- 4. OpenAI's Function API**
- 5. AutoGen Framework**
- 6. Design Patterns in AutoGen**
- 7. Building Agents with AutoGen**

Table of Contents

- 1. Agents as Thought + Action**
- 2. Defining Agents**
- 3. Agents in 3 Levels of Complexity**
- 4. OpenAI's Function API**
- 5. AutoGen Framework**
- 6. Design Patterns in AutoGen**
- 7. Building Agents with AutoGen**
- 8. Concluding Remarks**

Table of Contents

- 1. Agents as Thought + Action**
- 2. Defining Agents**
- 3. Agents in 3 Levels of Complexity**
- 4. OpenAI's Function API**
- 5. AutoGen Framework**
- 6. Design Patterns in AutoGen**
- 7. Building Agents with AutoGen**
- 8. Concluding Remarks**
- 9. References**

Thought + Action

Thought + Action

- How do we do stuff?

Thought + Action

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

Thought + Action

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

Thought + Action

- How do we do stuff? We **think** and we **act**
- Example: Decision-making process for attending a live-training
- **Thought:** "I want to learn about agents"

Thought + Action

- How do we do stuff? We **think** and we **act**
- Example: Decision-making process for attending a live-training
- **Thought:** "I want to learn about agents"
- **Action:** "Go to the internet and research cool platforms where I can learn about agents"

Thought + Action

- How do we do stuff? We **think** and we **act**
- Example: Decision-making process for attending a live-training
- **Thought:** "I want to learn about agents"
- **Action:** "Go to the internet and research cool platforms where I can learn about agents"
- **Thought:** "O'Reilly has some awesome courses and live-trainings"

Thought + Action

- How do we do stuff? We **think** and we **act**
- Example: Decision-making process for attending a live-training
- **Thought:** "I want to learn about agents"
- **Action:** "Go to the internet and research cool platforms where I can learn about agents"
- **Thought:** "O'Reilly has some awesome courses and live-trainings"
- **Action:** "Look up O'Reilly courses"

Thought + Action

- How do we do stuff? We **think** and we **act**
- Example: Decision-making process for attending a live-training
- **Thought:** "I want to learn about agents"
- **Action:** "Go to the internet and research cool platforms where I can learn about agents"
- **Thought:** "O'Reilly has some awesome courses and live-trainings"
- **Action:** "Look up O'Reilly courses"
- **Thought:** "Live-trainings by instructor Lucas are awesome"

Thought + Action

- How do we do stuff? We **think** and we **act**
- Example: Decision-making process for attending a live-training
 - **Thought:** "I want to learn about agents"
 - **Action:** "Go to the internet and research cool platforms where I can learn about agents"
 - **Thought:** "O'Reilly has some awesome courses and live-trainings"
 - **Action:** "Look up O'Reilly courses"
 - **Thought:** "Live-trainings by instructor Lucas are awesome"
 - **Action:** "Schedule live-training about agents with instructor Lucas Soares"

Thinking:

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

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

What is an Agent?

What is an Agent?

LLM

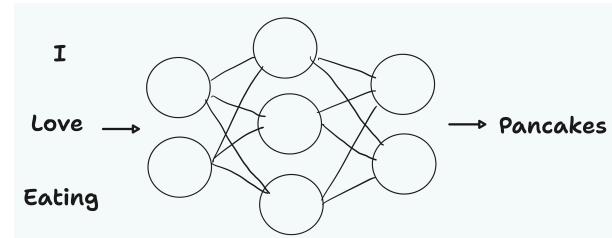
What is an Agent?

LLM + Tools

What is an Agent?

LLM + Tools

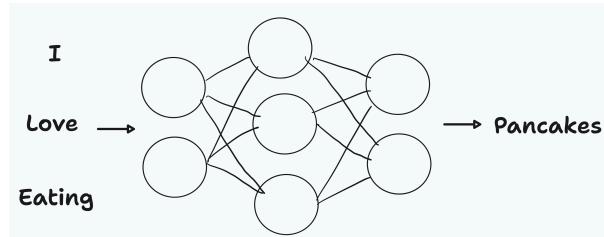
LLM= predicts next word/sentence



What is an Agent?

LLM + Tools

LLM= predicts next word/sentence



Tool= perform actions in the real-world



LLMs can use tools!

- 'Toolformer'

LLMs can use tools!

- 'Toolformer'

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

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

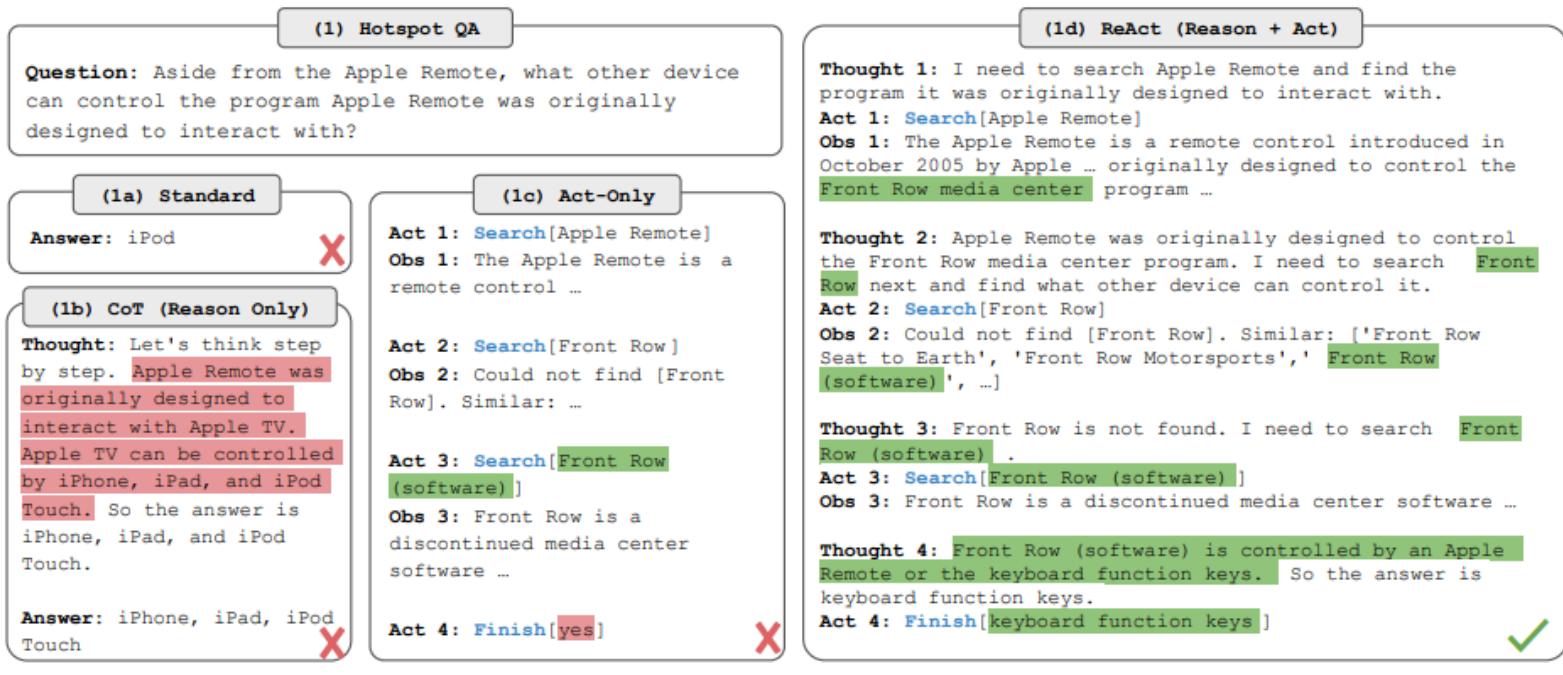
The name derives from "la tortuga", the Spanish word for [MT("tortuga") → turtle] turtle.

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

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

Interleaving Thoughts and Actions

- ReACT: LLMs for REasoning & ACTION.



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

Agents Are Getting Popular

Agents Are Getting Popular

- A Survey on Large Language Model based Autonomous Agents

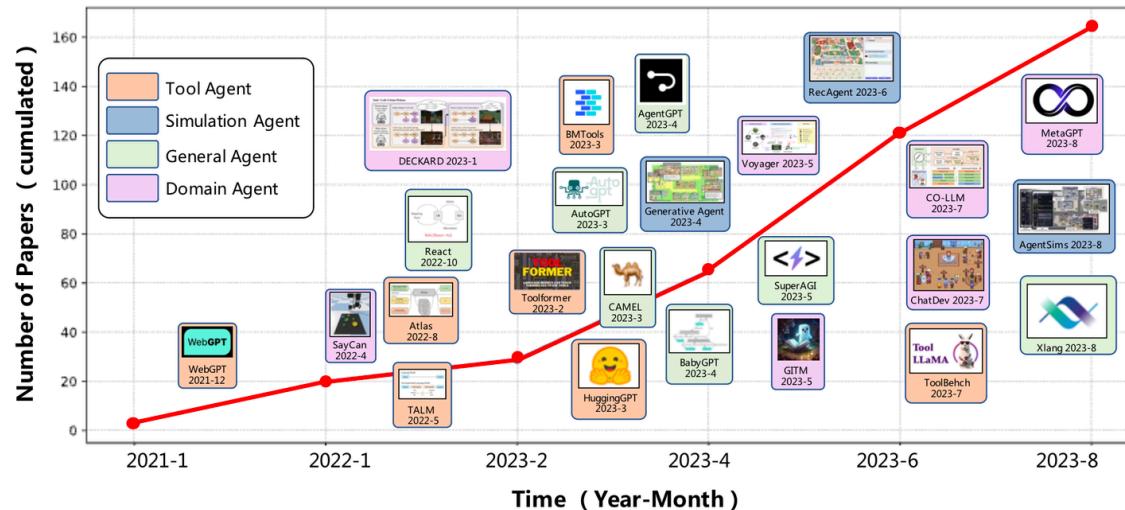


Figure 1: Illustration of the growth trend on the field of LLM-based autonomous agents.

[3] [\(Wang et al. 2023\)](#)

Popular Agent Implementations

Popular Agent Implementations

- BabyAGI: separate planning and execution steps

[4] BabyAGI

Popular Agent Implementations

- [AutoGPT](#): created for long-running, open-ended goals

AutoGPT: build & use AI agents

AutoGPT 50691 members Follow @Auto_GPT License MIT

AutoGPT is the vision of the power of AI accessible to everyone, to use and to build on. Our mission is to provide the tools, so that you can focus on what matters:

- 🏗️ **Building** - Lay the foundation for something amazing.
- ✎ **Testing** - Fine-tune your agent to perfection.
- 🤝 **Delegating** - Let AI work for you, and have your ideas come to life.

Be part of the revolution! AutoGPT is here to stay, at the forefront of AI innovation.

[Documentation](#) | [Contributing](#) | [Build your own Agent - Quickstart](#)

[5] [AutoGPT](#)

Popular Agent Implementations

- GPT-Researcher: produce detailed, factual and unbiased research reports

[6] GPT-Researcher

Popular Agent Implementations

- Custom GPTs for specific tasks

[7] [Custom GPT OpenAI Blog Post](#)

Agents in 3 Levels of Complexity

Level 1: LLM + functions inside the prompt

Level 1: LLM + functions inside the prompt

- Inspired by 'Toolformer'

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

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

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

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

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

Level 1: LLM + functions inside the prompt

```
from openai import OpenAI
client = OpenAI()

def get_response(prompt_question, model="gpt-3.5-turbo-16k"):
    response = client.chat.completions.create(
        model=model,
        messages=[{"role": "system", "content": "You are a helpful research and pr
            {"role": "user", "content": prompt_question}]
    )

    return response.choices[0].message.content

def create_directory(directory_name):
    subprocess.run(["mkdir", directory_name])

def create_file(file_name):
    subprocess.run(["touch", file_name])

def list_files():
    subprocess.run(["ls"])
```

Level 1: LLM + functions inside the prompt

```
task_description = "Create a folder called 'lucas-the-agent-master'. Inside that
output = get_response(f"""Given this task: {task_description}, \n
                        Consider you have access to the following functions:

def create_directory(directory_name):
    '''Function that creates a directory given a directory name.'''
    subprocess.run(["mkdir", directory_name])

def create_file(file_name):
    '''Function that creates a file given a file name.'''
    subprocess.run(["touch", file_name])

def list_files():
    '''Function that lists all files in the current directory.'''
    subprocess.run(["ls"])

Your output should be the first function to be executed to complete the task con
The OUTPUT SHOULD ONLY BE THE PYTHON FUNCTION CALL and NOTHING ELSE.
""")

Markdown(output)

# Output:
# create_directory('lucas-the-agent-master')
```

Level 1: LLM + functions inside the prompt

- Now, all we need is to find a way to execute this function.

Level 1: LLM + functions inside the prompt

- Now, all we need is to find a way to execute this function.
- We can use Python's built in `exec` method for that:

Level 1: LLM + functions inside the prompt

- Now, all we need is to find a way to execute this function.
- We can use Python's built in `exec` method for that:

```
# Output:  
# create_directory('lucas-the-agent-master')  
exec(output)  
!ls -d */ | grep lucas  
# Output:  
# lucas-the-agent-master/
```

Limitations

Limitations

- **Probabilistic outputs** make function calls unreliable

Limitations

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

Limitations

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

Limitations

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

OpenAI Function Calling

OpenAI Function Calling

- Standard way to connect models to outside tools.

OpenAI Function Calling

- Standard way to connect models to outside tools.

Process Simplified:

OpenAI Function Calling

- Standard way to connect models to outside tools.

Process Simplified:

- **Initial Query_**: Start by sending a query to the model.

OpenAI Function Calling

- Standard way to connect models to outside tools.

Process Simplified:

- **Initial Query**: Start by sending a query to the model.
- **Model Response**: The model may request additional information, packaged in a straightforward format.

OpenAI Function Calling

- Standard way to connect models to outside tools.

Process Simplified:

- **Initial Query**: Start by sending a query to the model.
- **Model Response**: The model may request additional information, packaged in a straightforward format.
- **Action**: Execute the necessary actions based on the model's request.

OpenAI Function Calling

- Standard way to connect models to outside tools.

Process Simplified:

- **Initial Query**: Start by sending a query to the model.
- **Model Response**: The model may request additional information, packaged in a straightforward format.
- **Action**: Execute the necessary actions based on the model's request.
- **Feedback Loop**: Share the outcomes with the model for a cohesive summary.

OpenAI Function Calling

- Standard way to connect models to outside tools.

Process Simplified:

- **Initial Query**: Start by sending a query to the model.
- **Model Response**: The model may request additional information, packaged in a straightforward format.
- **Action**: Execute the necessary actions based on the model's request.
- **Feedback Loop**: Share the outcomes with the model for a cohesive summary.
- **Takeaway**: This streamlined interaction sets the stage for the more advanced capabilities provided by the AutoGen framework, focusing on creating efficient, self-improving agents that can communicate with each other.

OpenAI Function Calling

- Standard way to connect models to outside tools.

Process Simplified:

- **Initial Query**: Start by sending a query to the model.
- **Model Response**: The model may request additional information, packaged in a straightforward format.
- **Action**: Execute the necessary actions based on the model's request.
- **Feedback Loop**: Share the outcomes with the model for a cohesive summary.
- **Takeaway**: This streamlined interaction sets the stage for the more advanced capabilities provided by the AutoGen framework, focusing on creating efficient, self-improving agents that can communicate with each other.

[8] [OpenAI Function Calling Docs](#)

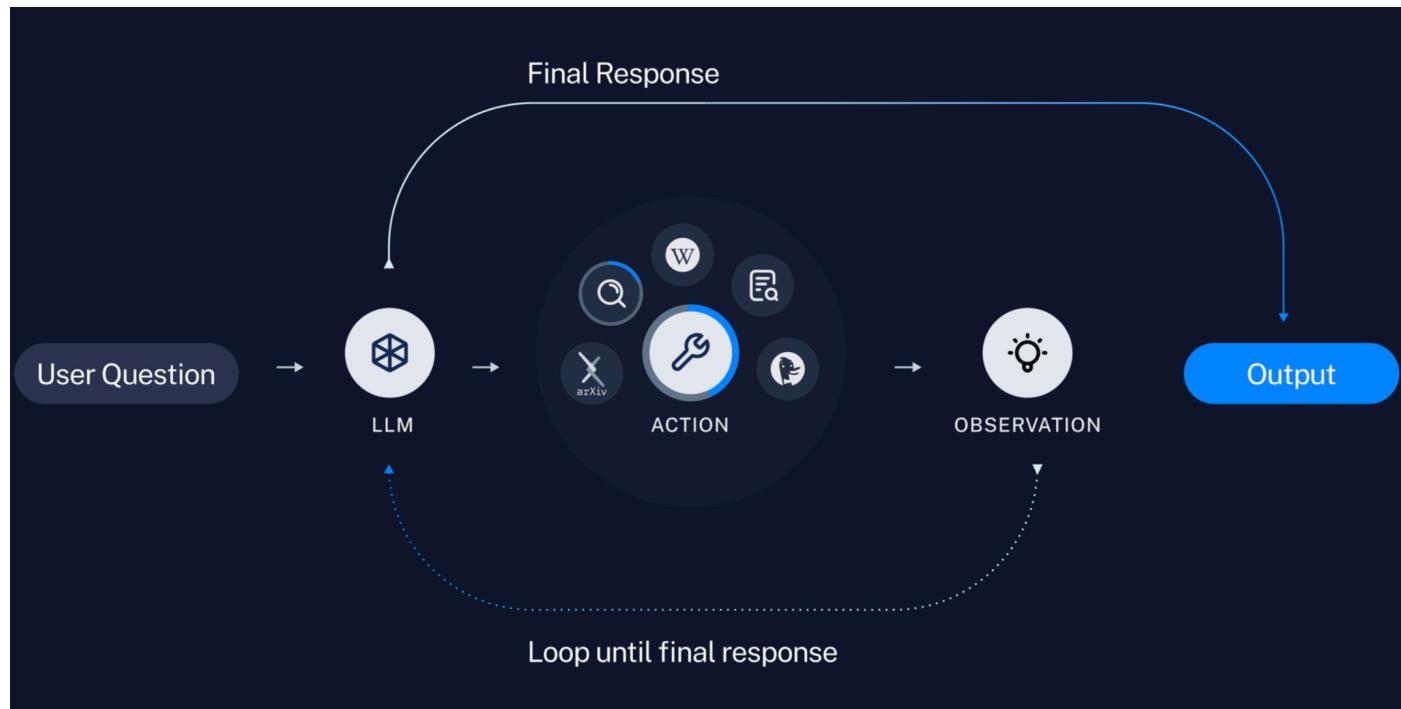
Q&A

Break 10 Minutes

Agents and AutoGen

How Can We Effectively Perform Tasks with Agents?

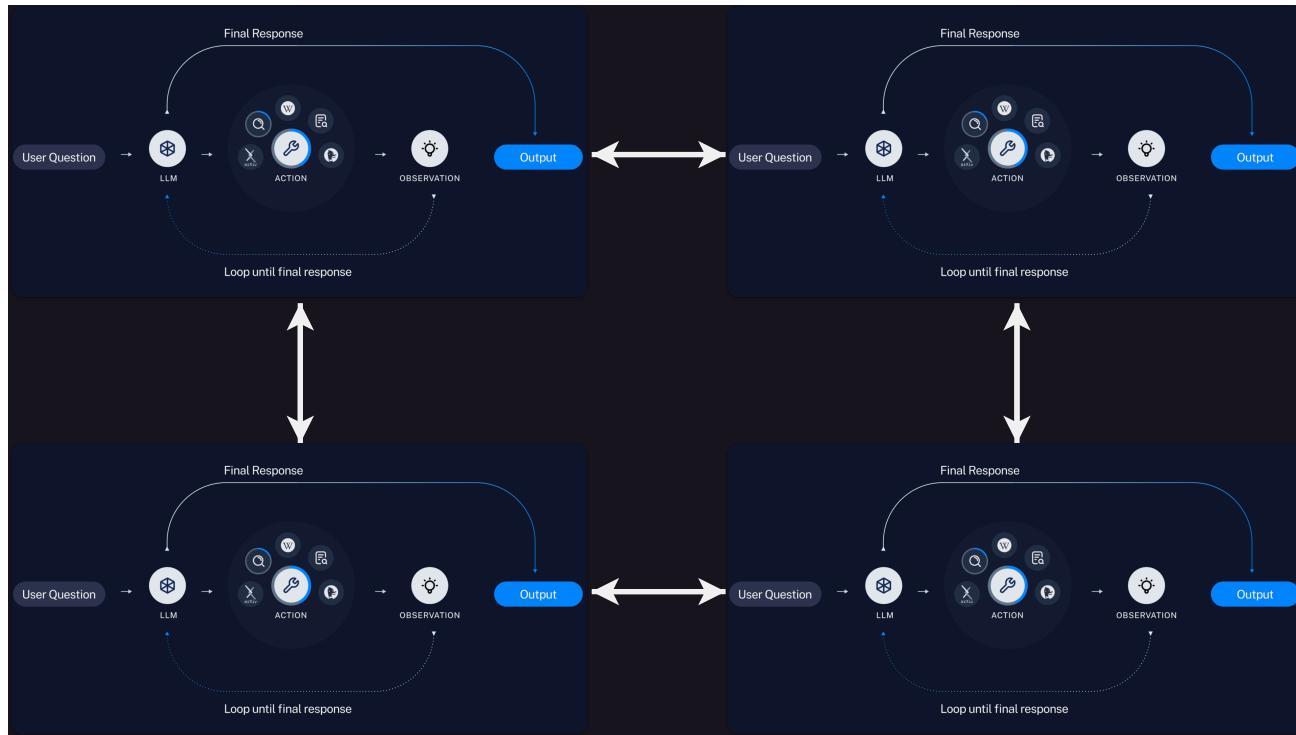
How Can We Effectively Perform Tasks with Agents?



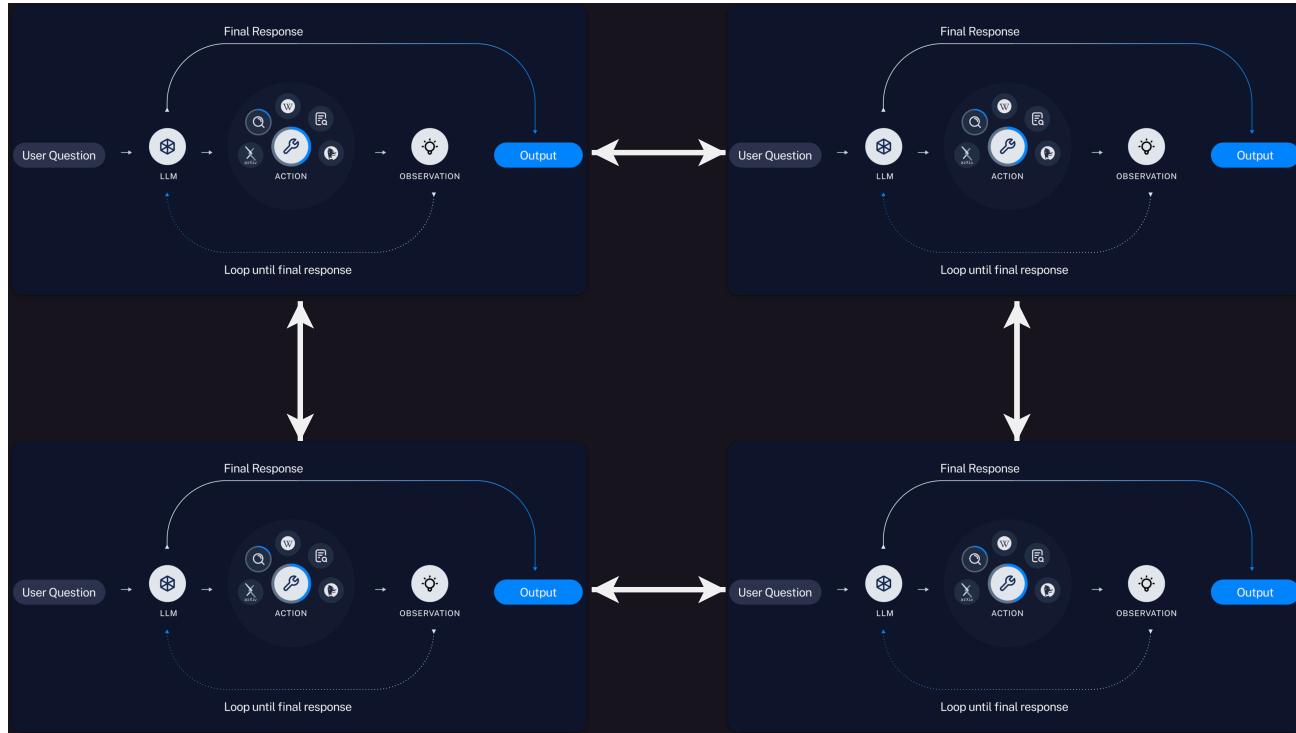
- The Agent Loop

[OpenAI's Bet on a Cognitive Architecture](#)

Good Agents Collaborate

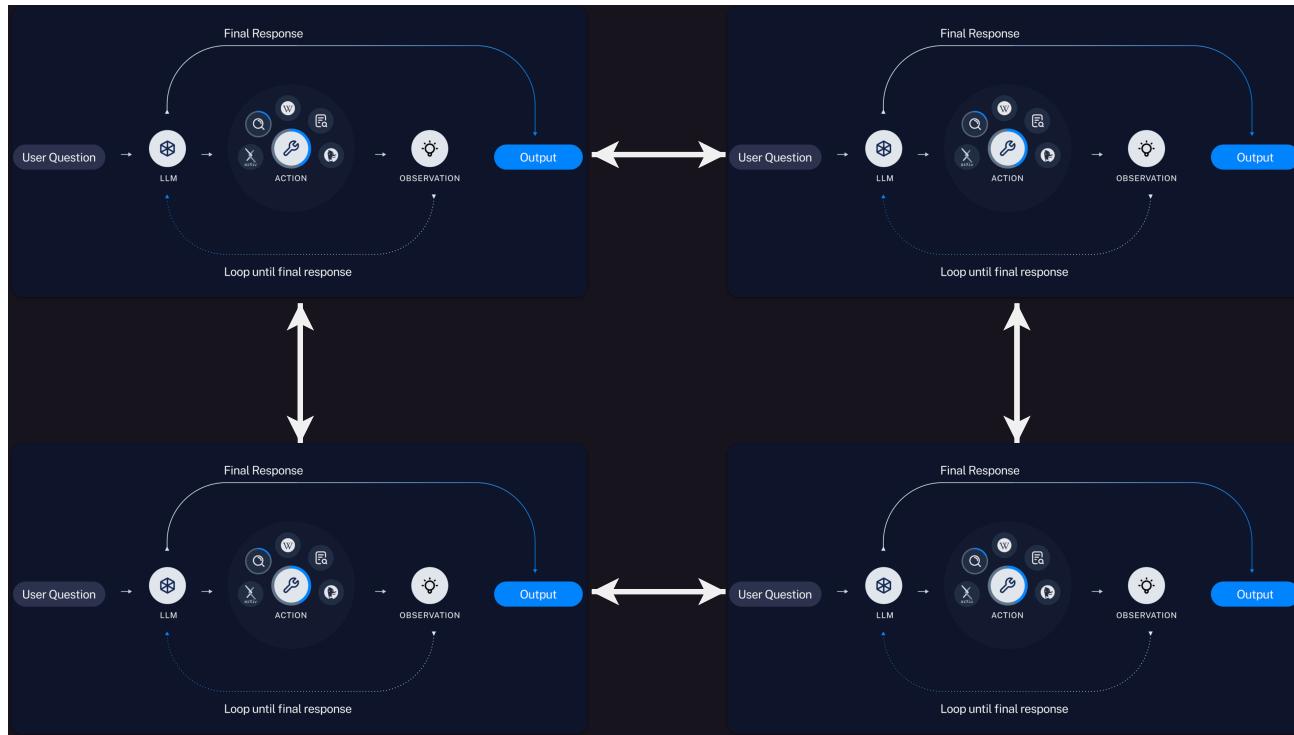


Good Agents Collaborate



- With the increasing complexity of tasks, agents need to **collaborate** to achieve their goals

Good Agents Collaborate



- With the increasing complexity of tasks, agents need to **collaborate** to achieve their goals
- AutoGen is a framework that facilitates the creation of agents that can easily collaborate through a 'conversation-centric' paradigm

What is AutoGen?

What is AutoGen?

- AutoGen is a framework for building agents that can collaborate through conversational patterns to accomplish tasks

What is AutoGen?

- AutoGen is a framework for building agents that can collaborate through conversational patterns to accomplish tasks
- Its main features

What is AutoGen?

- AutoGen is a framework for building agents that can collaborate through conversational patterns to accomplish tasks
- Its main features
 - **Conversable agents:** a generic design of agents that can leverage LLMs, human input, tools or a combination of these to facilitate creating agents with different roles

What is AutoGen?

- AutoGen is a framework for building agents that can collaborate through conversational patterns to accomplish tasks
- Its main features
 - **Conversable agents:** a generic design of agents that can leverage LLMs, human input, tools or a combination of these to facilitate creating agents with different roles
 - **Conversation Programming:** Programming paradigm centered around inter-agent conversations

Conversable Agents

Conversable Agents

- **Conversable:** Entity with a specific role that can pass messages to send and receive information to and from other conversable agents, e.g., to start or continue a conversation.

Conversable Agents

- **Conversable:** Entity with a specific role that can pass messages to send and receive information to and from other conversable agents, e.g., to start or continue a conversation.
- **Customizable:** Based on application-specific needs, each agent can be configured to have a mix of basic back-end types to display complex behavior in multi-agent conversations.

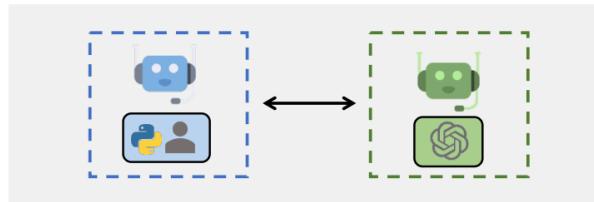
Conversable Agents

- **Conversable:** Entity with a specific role that can pass messages to send and receive information to and from other conversable agents, e.g., to start or continue a conversation.
- **Customizable:** Based on application-specific needs, each agent can be configured to have a mix of basic back-end types to display complex behavior in multi-agent conversations.
- **Example:**

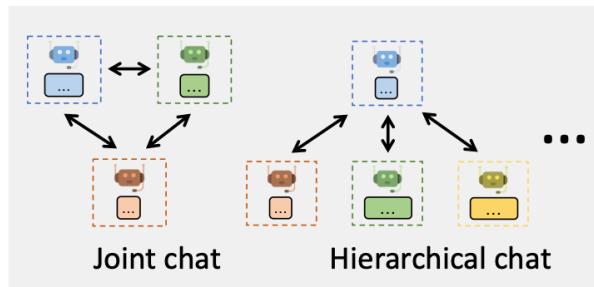
```
assistant = autogen.AssistantAgent(  
    name="assistant",  
    llm_config=llm_config,  
)
```

Conversation Programming

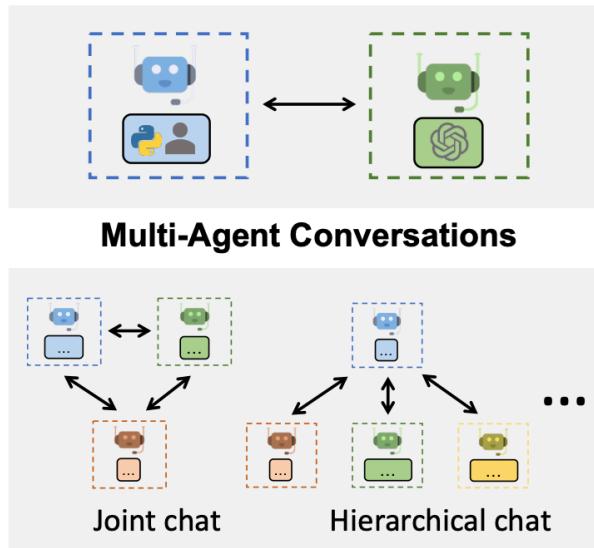
Conversation Programming



Multi-Agent Conversations

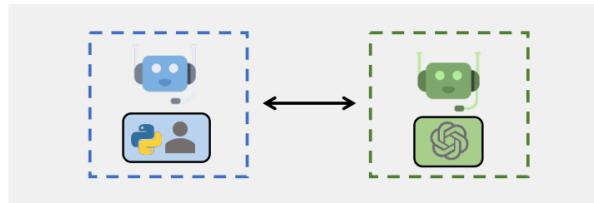


Conversation Programming

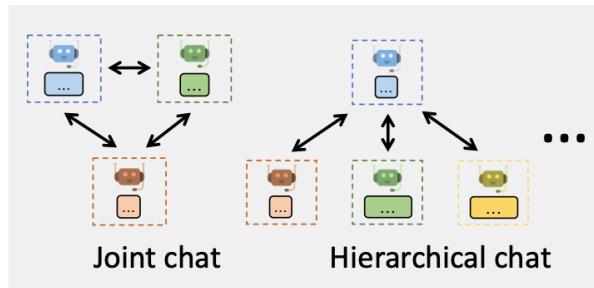


- Paradigm that blends computation and control flow within multi-agent conversations.

Conversation Programming

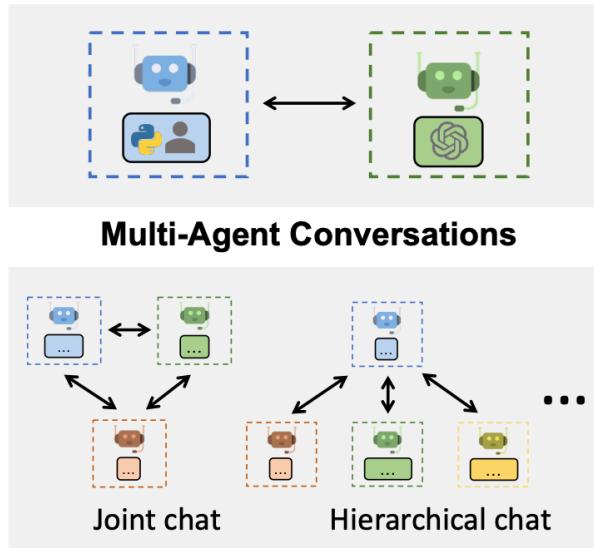


Multi-Agent Conversations



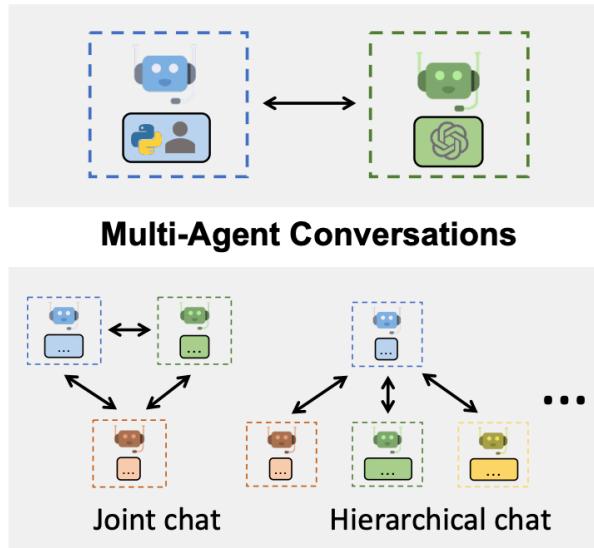
- Paradigm that blends computation and control flow within multi-agent conversations.
- Merges programming and natural language control.

Conversation Programming



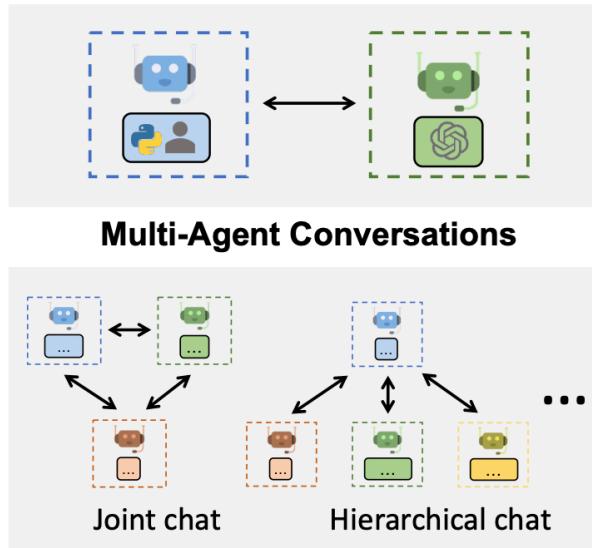
- Paradigm that blends computation and control flow within multi-agent conversations.
- Merges programming and natural language control.
- **Computation:** Role-specific, conversation-centric actions.

Conversation Programming



- Paradigm that blends computation and control flow within multi-agent conversations.
- Merges programming and natural language control.
- **Computation:** Role-specific, conversation-centric actions.
- **Control Flow:** Defined by conversation dynamics among agents.

Conversation Programming



- Paradigm that blends computation and control flow within multi-agent conversations.
- Merges programming and natural language control.
- **Computation:** Role-specific, conversation-centric actions.
- **Control Flow:** Defined by conversation dynamics among agents.
- **Efficiency:** Streamlines AI development for various skill levels.

Design Patterns

Design Patterns

- **Unified Interfaces:** Standardized interfaces for agent interactions.

Design Patterns

- **Unified Interfaces:** Standardized interfaces for agent interactions.
- **Auto-Reply:** auto-reply mechanism for continuous conversation flow.

Design Patterns

- **Unified Interfaces:** Standardized interfaces for agent interactions.
- **Auto-Reply:** auto-reply mechanism for continuous conversation flow.

Dynamic Conversations

Design Patterns

- **Unified Interfaces:** Standardized interfaces for agent interactions.
- **Auto-Reply:** auto-reply mechanism for continuous conversation flow.

Dynamic Conversations

- Supports static and dynamic flows.

Design Patterns

- **Unified Interfaces:** Standardized interfaces for agent interactions.
- **Auto-Reply:** auto-reply mechanism for continuous conversation flow.

Dynamic Conversations

- Supports static and dynamic flows.
- Customizable reply functions for adaptive conversations.

Autogen Demo - Building Our First Agent

Diverse Conversation Patterns

Diverse Conversation Patterns

- Pattern = *How you set up the interaction*

Diverse Conversation Patterns

- Pattern = *How you set up the interaction*

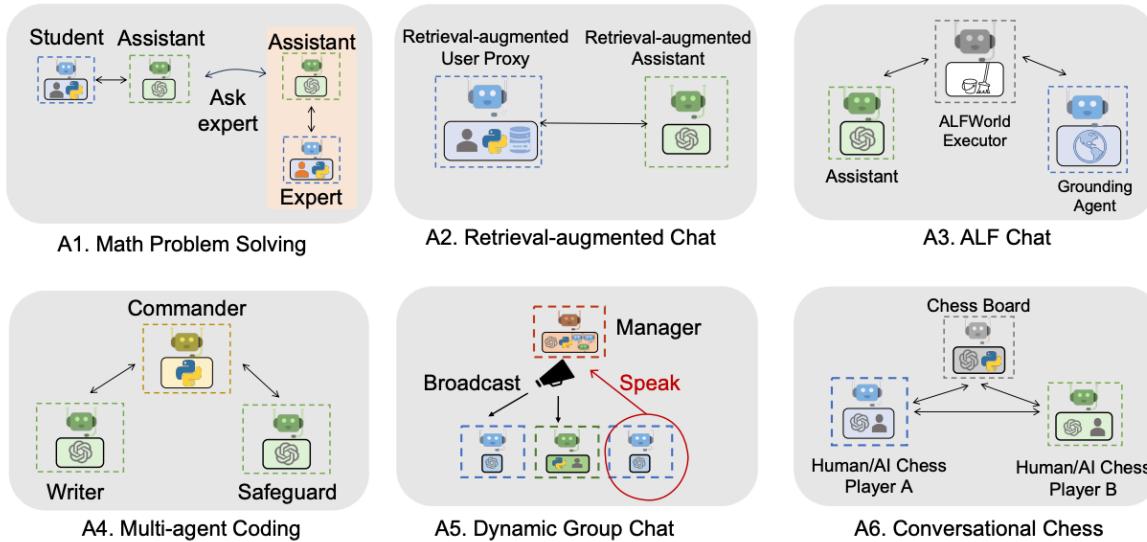


Figure 3: Six examples of diverse applications built using AutoGen. Their conversation patterns show AutoGen's flexibility and power.

- Adaptable to different agent autonomies and topologies.

Examples of Conversation Patterns

Autogen Demo (Student - Assistant - Expert)

1. Student - Assistant - Expert

1. Student - Assistant - Expert

```
assistant = autogen.AssistantAgent(  
    name="assistant",  
    system_message="You are a helpful assistant.",  
    llm_config={  
        "timeout": 600,  
        "seed": 42,  
        "config_list": config_list,  
    },  
)  
  
mathproxyagent = MathUserProxyAgent(  
    name="mathproxyagent",  
    human_input_mode="NEVER",  
    code_execution_config={"use_docker": False},  
)  
  
math_problem = (  
    "Find all  $x$  that satisfy the inequality  $(2x+10)(x+3) < (3x+9)(x+8)$ . Express  
)  
mathproxyagent.initiate_chat(assistant, problem=math_problem)
```

Autogen Demo (Retrieval Augmented Chat)

2. Retrieval Augmented Chat

2. Retrieval Augmented Chat

```
assistant = RetrieveAssistantAgent(  
    name="assistant",  
    system_message="You are a helpful assistant.",  
    llm_config={  
        "timeout": 600,  
        "cache_seed": 42,  
        "config_list": config_list,  
    },  
)  
ragproxyagent = RetrieveUserProxyAgent(  
    name="ragproxyagent",  
    human_input_mode="NEVER",  
    max_consecutive_auto_reply=3,  
    retrieve_config={  
        "task": "code",  
        "docs_path": [  
            "https://raw.githubusercontent.com/microsoft/FLAML/main/website/docs",  
            "https://raw.githubusercontent.com/microsoft/FLAML/main/website/docs",  
            os.path.join(os.path.abspath(""), "...", "website", "docs"),  
        ],  
        ...  
        ...  
        code_execution_config=False, # set to False if you don't want to execute the  
    })  
assistant.reset()  
code_problem = "How can I use FLAML to perform a classification task and use spa  
ragproxyagent.initiate_chat(assistant, problem=code_problem, search_string="spa
```

Autogen Demo (Writer - Commander - Safeguard)

3. Writer - Commander - Safeguard (multi-agent coding)

3. Writer - Commander - Safeguard (multi-agent coding)

```
assistant = autogen.AssistantAgent(  
    name="assistant",  
    llm_config={  
        "cache_seed": 42, # seed for caching and reproducibility  
        "config_list": config_list, # a list of OpenAI API configurations  
        "temperature": 0, # temperature for sampling  
    }, # configuration for autogen's enhanced inference API which is compatible  
)  
# create a UserProxyAgent instance named "user_proxy"  
user_proxy = autogen.UserProxyAgent(  
    name="user_proxy",  
    human_input_mode="NEVER",  
    max_consecutive_auto_reply=10,  
    is_termination_msg=lambda x: x.get("content", "").rstrip().endswith("TERMINA  
code_execution_config={  
    "work_dir": "coding",  
    "use_docker": False, # set to True or image name like "python:3" to use  
},  
)  
# the assistant receives a message from the user_proxy, which contains the task  
user_proxy.initiate_chat(  
    assistant,  
    message="""What date is today? Compare the year-to-date gain for META and TE  
)
```

Autogen Demo (Dynamic Group Chat)

4. Dynamic Group Chat

4. Dynamic Group Chat

```
llm_config = {"config_list": config_list_gpt4, "cache_seed": 42}
user_proxy = autogen.UserProxyAgent(
    name="User_proxy",
    system_message="A human admin.",
    code_execution_config={"last_n_messages": 2, "work_dir": "groupchat"},
    human_input_mode="TERMINATE",
)
coder = autogen.AssistantAgent(
    name="Coder",
    llm_config=llm_config,
)
pm = autogen.AssistantAgent(
    name="Product_manager",
    system_message="Creative in software product ideas.",
    llm_config=llm_config,
)
groupchat = autogen.GroupChat(agents=[user_proxy, coder, pm], messages=[], max_rou
manager = autogen.GroupChatManager(groupchat=groupchat, llm_config=llm_config)
user_proxy.initiate_chat(
    manager, message="Find a latest paper about gpt-4 on arxiv and find its potential
)
```

Break 10 minutes

Recipe for Building AutoGen Agent Workflows

Step 1: Prepare Agent Configurations

- Config path with model name & API key
- Default configuration for each agent

```
config_file_or_env = './OAI_CONFIG_LIST' # modify path
default_llm_config = {
    'temperature': 0
}
```

Step 2: Creating AgentBuilder Instance

- Create AgentBuilder instance
 - Use configuration path & default config
 - Specify builder & agent models

```
from autogen.agentchat.contrib.agent_builder import AgentBuilder  
builder = AgentBuilder(config_file_or_env=config_file_or_env, builder_model='gpt-4')
```

Step 3: Specifying the Building Task

- Define building task with description & examples
 - Helps build manager decide on agents
 - Example Task: "Create a multi-agent system for task X"

```
building_task = "Find a paper on arxiv about artificial intelligence,  
and analyze its application in some domain. For example, find a latest  
paper about gpt-4 on arxiv and find its potential applications in  
software."
```

Step 4: Building Group Chat Agents

- Use `build()` method to generate agents
- Include a user proxy for tasks involving coding
 - `agent_list, agent_configs = builder.build(building_task, default_llm_config, coding=True)`

```
agent_list, agent_configs = builder.build(building_task, default_llm_config,  
coding=True)
```

Step 5: Executing the Task

- Agents collaborate in a group chat to complete task
- Combine LLMs, human inputs, and tools
- Example: `start_task(execution_task, agent_list, llm_config)`

```
import autogen
def start_task(execution_task: str, agent_list: list, llm_config: dict):
    config_list = autogen.config_list_from_json(config_file_or_env, filter_dict={"|")
        group_chat = autogen.GroupChat(agents=agent_list, messages=[], max_round=12)
        manager = autogen.GroupChatManager(
            groupchat=group_chat, llm_config={"config_list": config_list, **llm_config})
        agent_list[0].initiate_chat(manager, message=execution_task)

start_task(
    execution_task="Find a recent paper about gpt-4 on arxiv and find its potential applications",
    agent_list=agent_list,
    llm_config=default_llm_config
)
```

source: [AutoGen Blog](#)

Step 6: Clearing Agents & Saving Configurations

- Clear agents post-task if next task differs
- Save information of built group chat agents

```
builder.clear_all_agents(recycle_endpoint=True)
saved_path = builder.save()
```

- Configurations will be saved in JSON format like such:

```
// FILENAME: save_config_TASK_MD5.json
{
  "building_task": "Find a paper on arxiv about artificial intelligence,
  and analyze its application in some domain. For example, find a latest
  paper about gpt-4 on arxiv and find its potential applications in
  software." ,
  "agent_configs": [
    {
      "name": "...",
      ....
    },
    ...
  ],
  "manager_system_message": "...",
  "code_execution_config": {...},
  "default_llm_config": {...}
}
```

Autogen Demo - Agent Builder Recipe

Autogenstudio Demo - Local LLMs

Autogen Demo - Building a Research Assistant

Autogen Demo - Personal Assistant

Conclusion and Q&A

References

- [AutoGen](#)
- [AutoGen Paper](#)
- [OpenAI](#)
- [OpenAI Function Calling](#)
- [Gen Agents](#),
- [AutoGPT](#)
- [GPT-Engineer](#)
- [BabyAGI](#)
- [Karpathy on Agents](#)
- [ReACT Paper](#)
- [HuggingGPT](#)