AUTOGEN: ENABLING NEXT-GEN LLM APPLICATIONS VIA MULTI-AGENT CONVERSATION

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu,
Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang,
Shaokun Zhang, Jiale Liu, Ahmed Hassan
Awadallah, Ryen W White, Doug Burger, Chi Wang

Accepted by ICLR2024

Paper link

Outline

1. INTRODUCTION
2. THE AUTOGEN FRAMEWORK
3. APPLICATIONS OF AUTOGEN
4. DISCUSSION

Introduction

As the breadth and complexity of tasks applied to LLMs continue to increase, we need to enhance LLM capabilities through a **multi-agent approach**. The advantages of using multiple agents are:

- LLM agents can collaborate through conversations with each other.
- Differently configured agents work together to combine LLM capabilities in complementary ways.
- LLMs can solve complex tasks by breaking them into subtasks, with multi-agent dialogues helping to manage this process.

Problems encountered :

1. How can we design individual agents that are capable, reusable, customizable, and effective in multi-agent collaboration ?
2. How can we develop a straightforward, unified interface that can accommodate a wide range of agent conversation patterns?

Thus, their proposed AutoGen encompasses these two concepts :

- Customizable and conversable agents
- Conversation programming

THE AUTOGEN FRAMEWORK

Core Design Principle of AutoGen

- Aims to **reduce the effort** needed to build complex LLM applications
- Transforms complex tasks into manageable steps via **multi-agent discussions**
- Also focuses on **maximizing the reusability** of agents
- Introduces two key concepts:
  - Conversable Agents
  - Conversation Programming

Conversable Agents

- Has a specific role
- Can send/receive messages to/from other agents
- Maintains conversation context based on the message history
- Configurable with LLMs, tools, or human input
- Behaves based on predefined logic

Conversable Agents

- AutoGen agents can be powered by:
  - **LLMs**: Enable reasoning, role-playing, code generation, and history-aware responses.These capabilities can be **combined and enhanced** using **prompting techniques.**
  - **Humans**: AutoGen supports human participation via human-backed agents
    - The default UserProxyAgent allows:
      - Configurable involvement frequency and conditions
      - Option to skip human input when needed

Conversable Agents

- AutoGen agents can be powered by:
  - **Tools:** have the capability to execute tools via code execution or function execution.
    - For example, the default UserProxyAgent in AutoGen is able to execute code suggested by LLMs, or make LLM-suggested function calls.

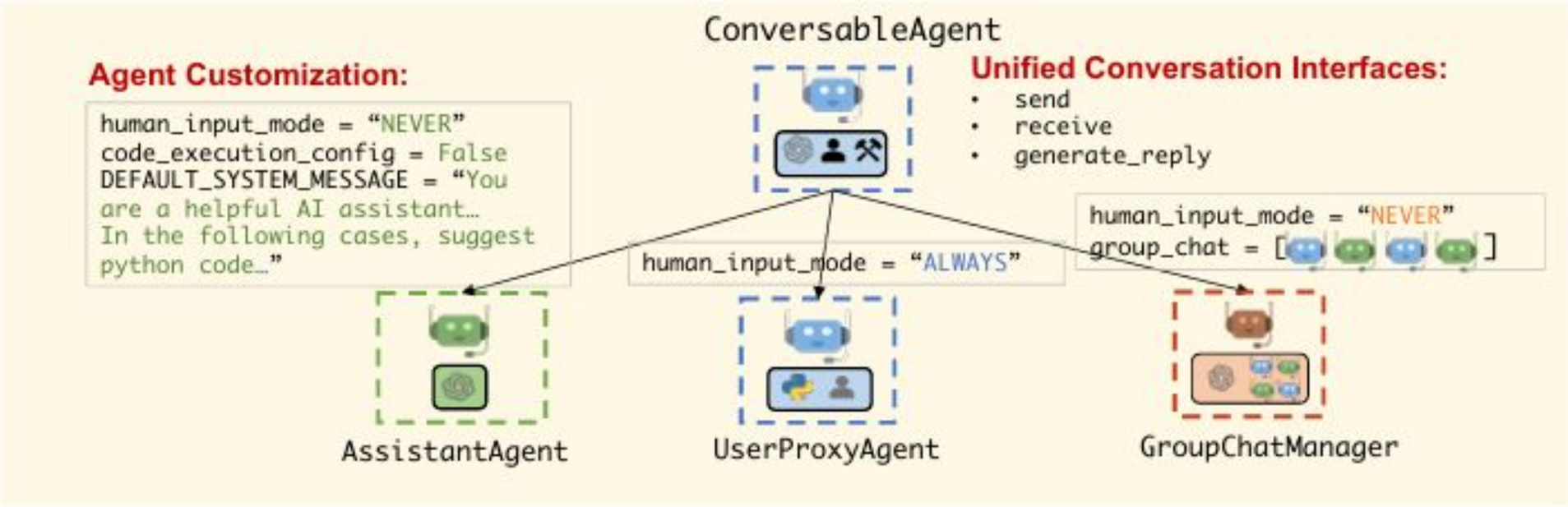
Conversable Agents

- Agent Customization
  - Agents can be **customized** with different back-end types (LLMs, humans, tools)
  - AutoGen supports easy agent creation by reusing/extending built-in classes

Conversable Agents

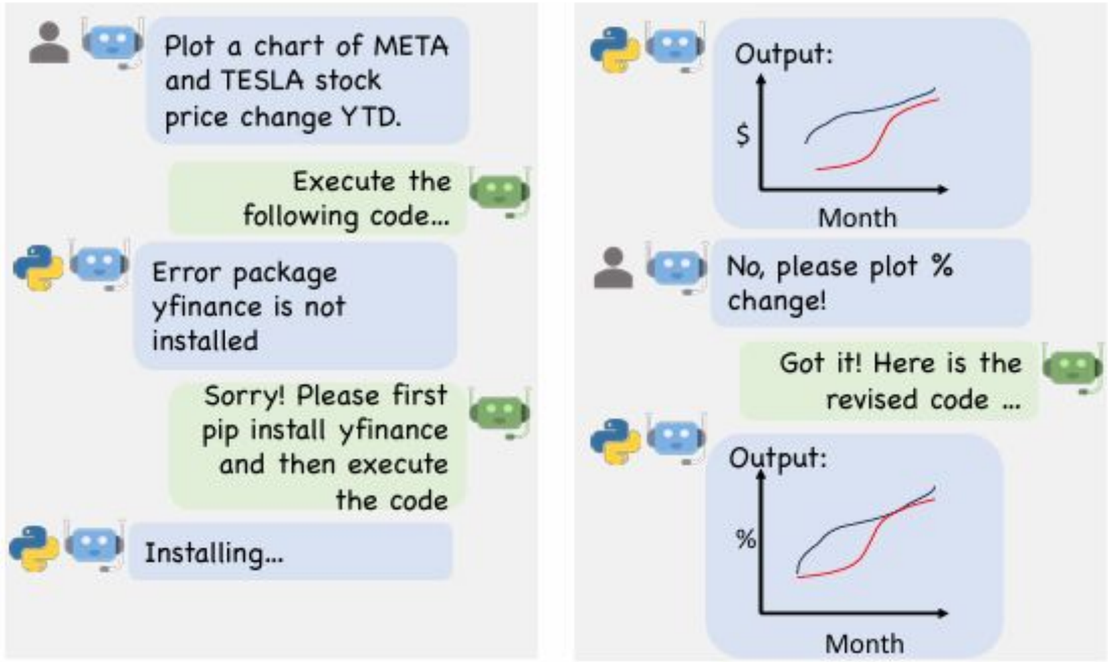
- ConversableAgent: Basic agent, supports all back-ends
- AssistantAgent : LLM-backed AI assistant
- UserProxyAgent: Human/tool-backed agent to receive input or execute code

AutoGen Agents



Conversable Agents

- AssistantAgent generates solution using LLM
- UserProxyAgent executes the code or asks for human input
- Feedback loop enables task refinement



Example Agent Chat

Conversation Programming

- Developers must **control and coordinate** multi-agent conversations to make meaningful task progress
- AutoGen introduces a paradigm called **Conversation Programming**, which includes:
  - Computation - The actions agents take to generate responses based on conversation
  - Control Flow - The **order & condition** for executing computations
- Both computation and control flow are **conversation-driven**

Conversation Programming

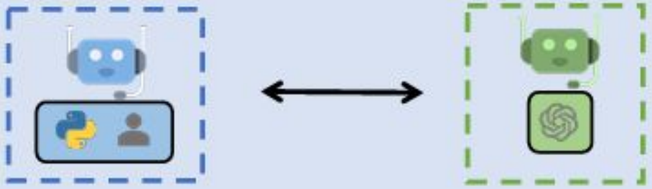
Developer Code

1.2 Register a Custom Reply Func:

```
# 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
```

1.1 Define Agents:




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

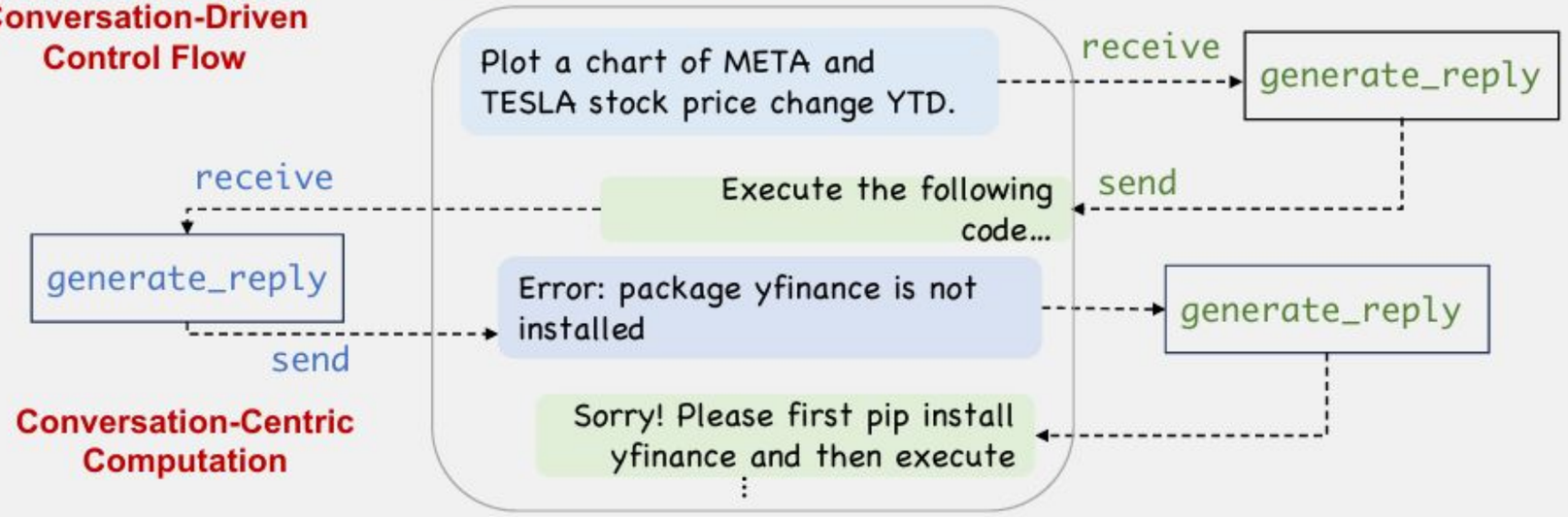
2 Initiate Conversations:

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

Program Execution

Conversation-Driven Control Flow

The Resulting Automated Agent Chat:



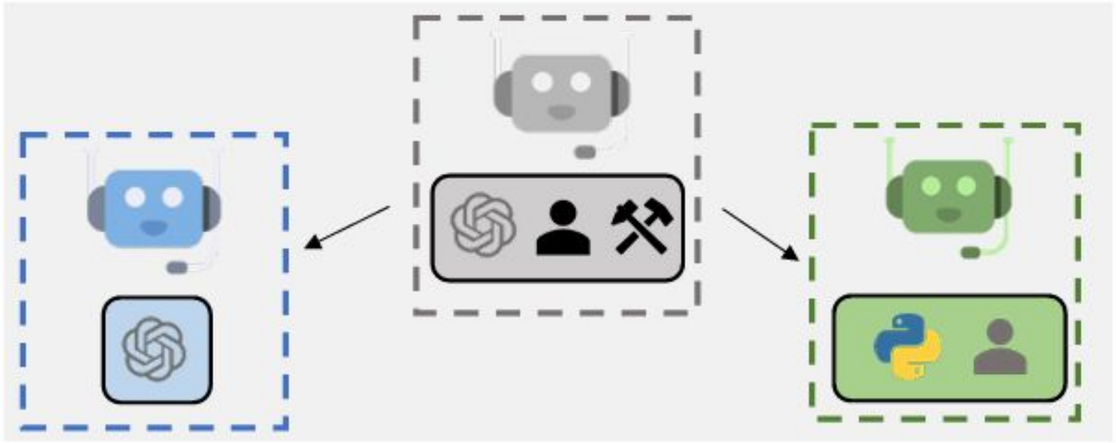
Conversation-Centric Computation

Conversation Programming

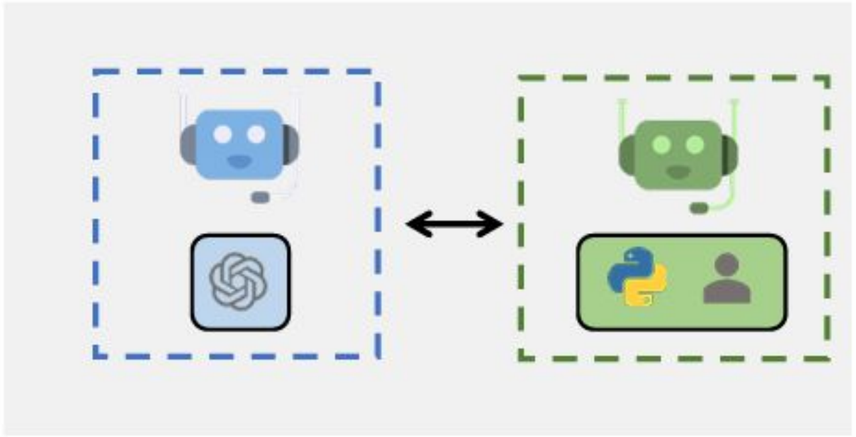
- Unified interfaces and auto-reply mechanisms
  - Unified interfaces for all agents:
    1. send() / receive() → message passing
    2. generate_reply() → respond to message
    3. register_reply() → define custom behaviors
- Auto-reply mechanism:
  - Automatically triggers reply when message is received
  - Supports built-in or custom reply (LLM, code, human input)
- No central controller needed — conversation flows **autonomously**

Conversation Programming

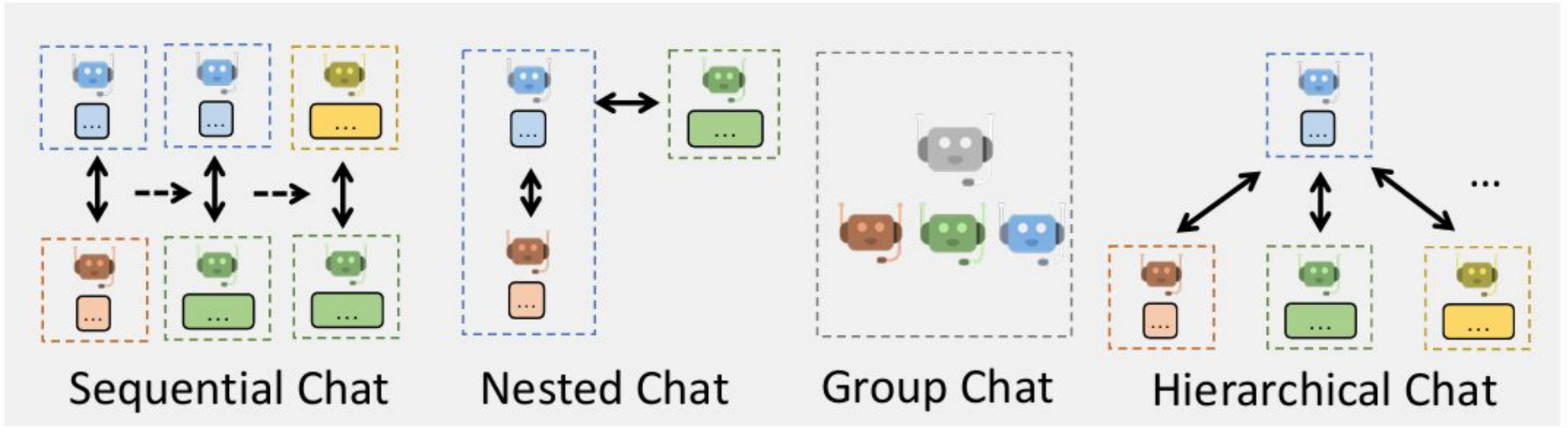
- Flexible Control Flow
  - Natural-language control: Use prompts to guide LLM behavior
  - Programming-language control: Python can define logic. For example, termination, input conditions, auto-reply limits
  - Mixed control transition:
    - From code → LLM: custom reply function can invoke LLM
    - From LLM → code: LLM proposes function calls to trigger execution



Agent Customization



Multi-Agent Conversations



Flexible Conversation Patterns

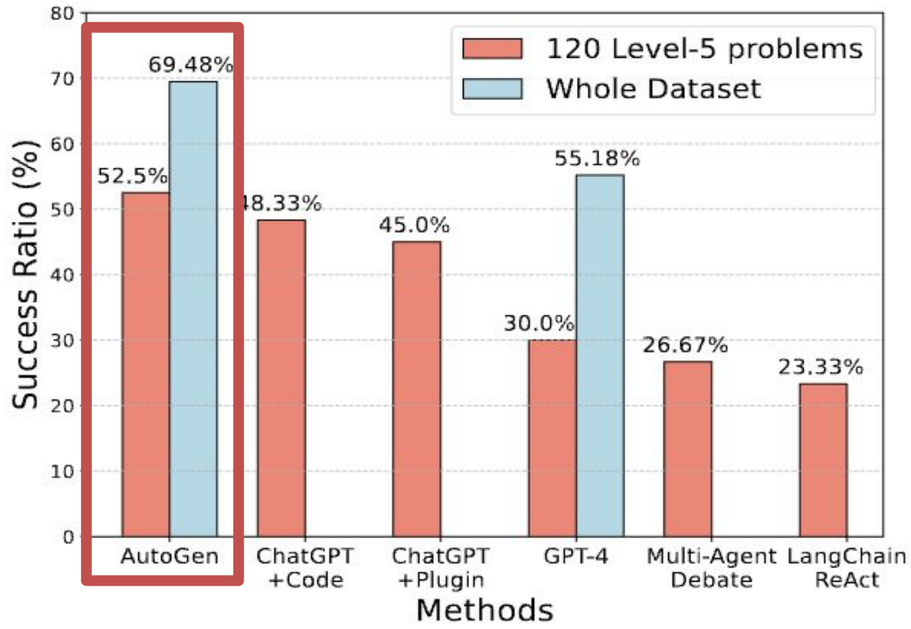
Conversation Programming

- Composable Conversation Patterns
  - AutoGen supports composing agents in various conversation patterns: sequential chat, nested chat, group chat, Hierarchical Chat
  - Patterns can be:
    - Static (predefined structure)
    - Dynamic (depends on message/context)

APPLICATIONS OF AUTOGEN

Case A1 – Math problem solving

- Scenario 1 (Operate completely autonomously)
  - Dataset
    - MATH (Hendrycks et al., 2021)
  - Evaluation
    - 120 level 5 problems
    - Whole dataset (5000 problems)



(a) A1: Performance on MATH (w/ GPT-4).

APPLICATIONS OF AUTOGEN

Case A1 – Math problem solving

- Scenario 1 (Operate completely autonomously)
  - Problem solving rate
    - Q1 a square root fraction
    - Q2 second number theory
    - AutoGen better performance

	Correctness	Failure Reason
AutoGen	3/3	N/A.
AutoGPT	0/3	The LLM gives code without the print function so the result is not printed.
ChatGPT+Plugin	1/3	The return from Wolfram Alpha contains 2 simplified results, including the correct answer, but GPT-4 always chooses the wrong answer.
ChatGPT+Code Interpreter	2/3	Returns a wrong decimal result.
LangChain ReAct	0/3	LangChain gives 3 different wrong answers.
Multi-Agent Debate	0/3	It gives 3 different wrong answers due to calculation errors.

(a) Evaluation on the first problem that asks to simplify a square root fraction.

	Correctness	Failure Reason
AutoGen	2/3	The final answer from code execution is wrong.
AutoGPT	0/3	The LLM gives code without the print function so the result is not printed.
ChatGPT+Plugin	1/3	For one trial, GPT-4 got stuck because it keeps giving wrong queries and has to be stopped. Another trial simply gives a wrong answer.
ChatGPT+Code Interpreter	0/3	It gives 3 different wrong answers.
LangChain ReAct	0/3	LangChain gives 3 different wrong answers.
Multi-Agent Debate	0/3	It gives 3 different wrong answers.

(b) Evaluation on the second number theory problem.

APPLICATIONS OF AUTOGEN

Case A1 – Math problem solving

- Scenario 1 (Operate completely autonomously)
  - Quality evaluation
    - Verbosity (Detail to Brief)
      AutoGPT > AutoGen/LangChain/
      ChatGPT+Code Interpreter > ChatGPT + Plugin
    - System stability
      Smoothly :AutoGen & ChatGPT+Code Interpreter
      AutoGPT : without the print' statement and cannot revise
      ChatGPT + Wolfram: stuck in a loop
      Langchain: handle parse error

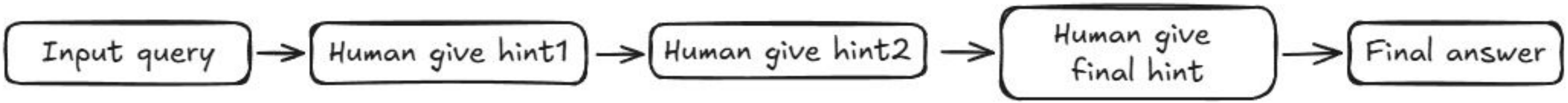
APPLICATIONS OF AUTOGEN

Case A1 – Math problem solving

- Scenario 2 (Human in loop)

  Problems cannot be solved without humans

  Every question test 3 times



APPLICATIONS OF AUTOGEN

Case A1 – Math problem solving

- Scenario 2 (Human in loop)
  - Question

    Find the equation of the plane which bisects the angle between the planes $3x - 6y + 2z + 5 = 0$ and $4x - 12y + 3z - 3 = 0$, and which contains the point $(-5, -1, -5)$.
  - Result(success/loss)

    AutoGen 3/0

    ChatGPT + Code Interpreter 2/1 -> fail to use hint

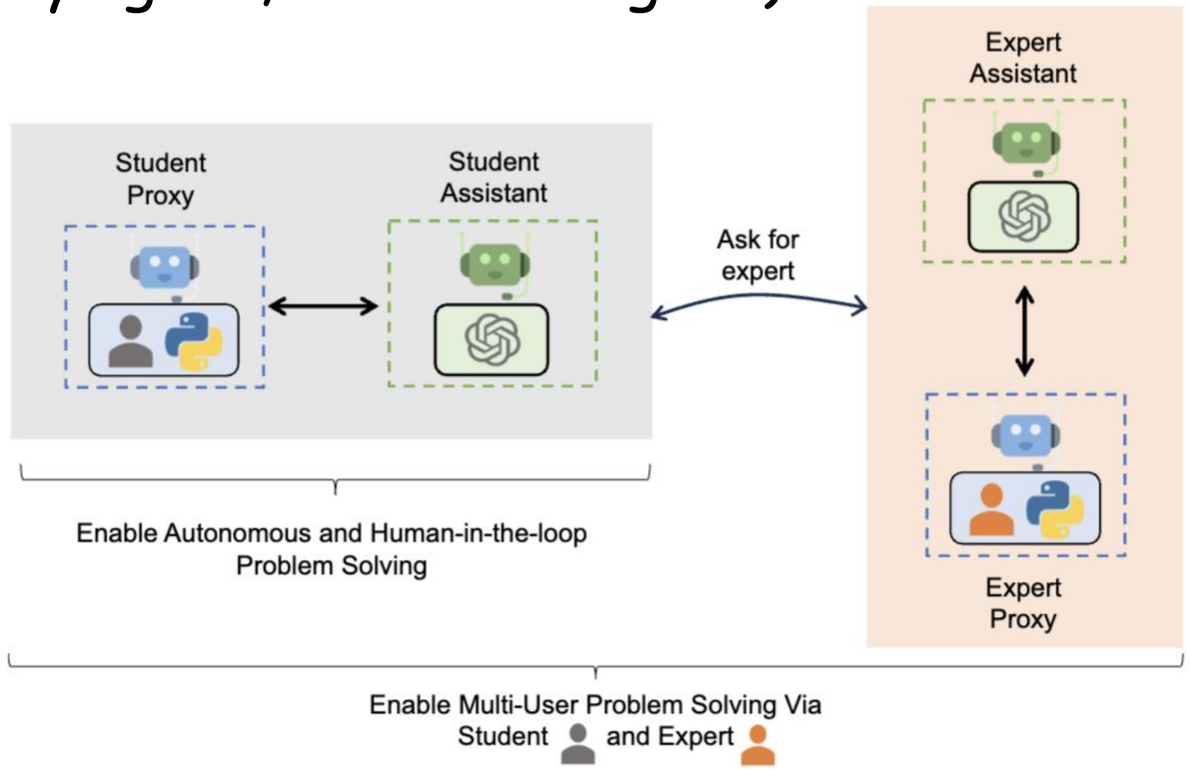
    ChatGPT + Plugin 2/1 -> error symbol in answer

    AutoGPT 0/3 -> derived an incorrect distance equation(1)
    & code execution errors(2)

APPLICATIONS OF AUTOGEN

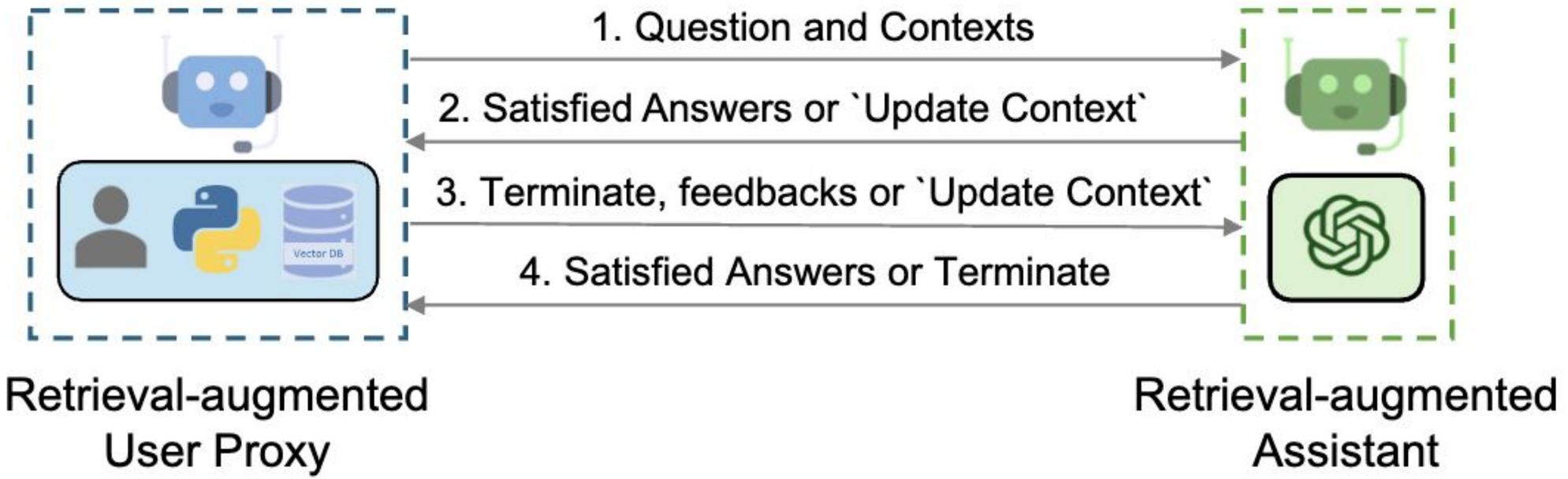
Case A1 – Math problem solving

- Scenario 3 (Multiple user participant)
  Easily to realize (UserProxyAgent, AssistantAgent)
  High expansibility



APPLICATIONS OF AUTOGEN

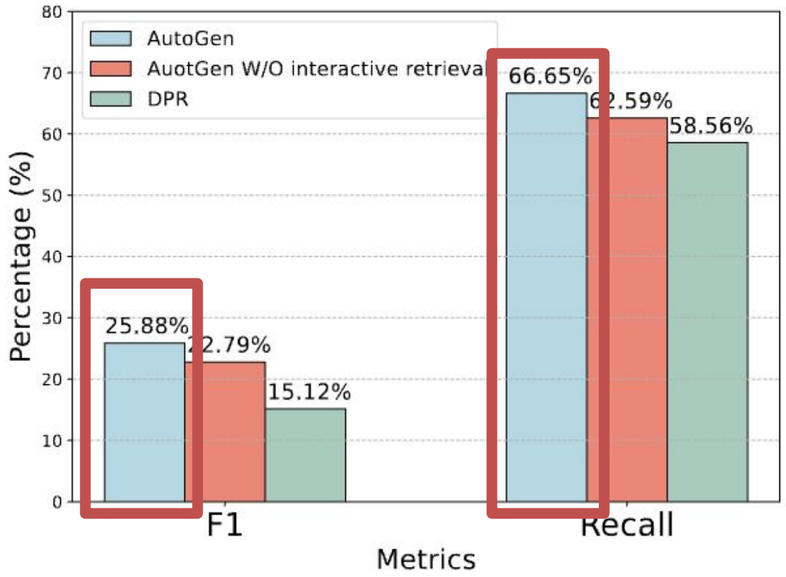
Case A2 – RAG & QA



APPLICATIONS OF AUTOGEN

Case A2 – RAG & QA

- Scenario 1
  an evaluation regarding natural question answering
  Dataset: Natural Questions



(b) A2: Q&A tasks (w/ GPT-3.5).

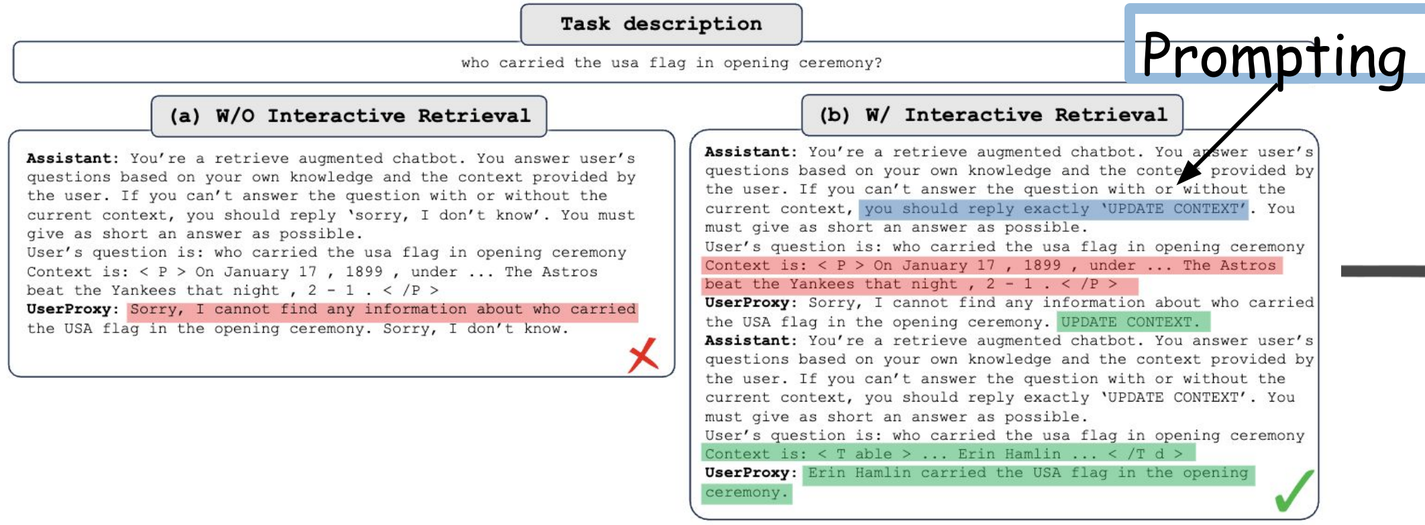


Figure 8: Retrieval-augmented Chat without (W/O) and with (W/) *interactive retrieval*.

APPLICATIONS OF AUTOGEN

Case A2 – RAG & QA

- Scenario 2

  Generate code base on codebase

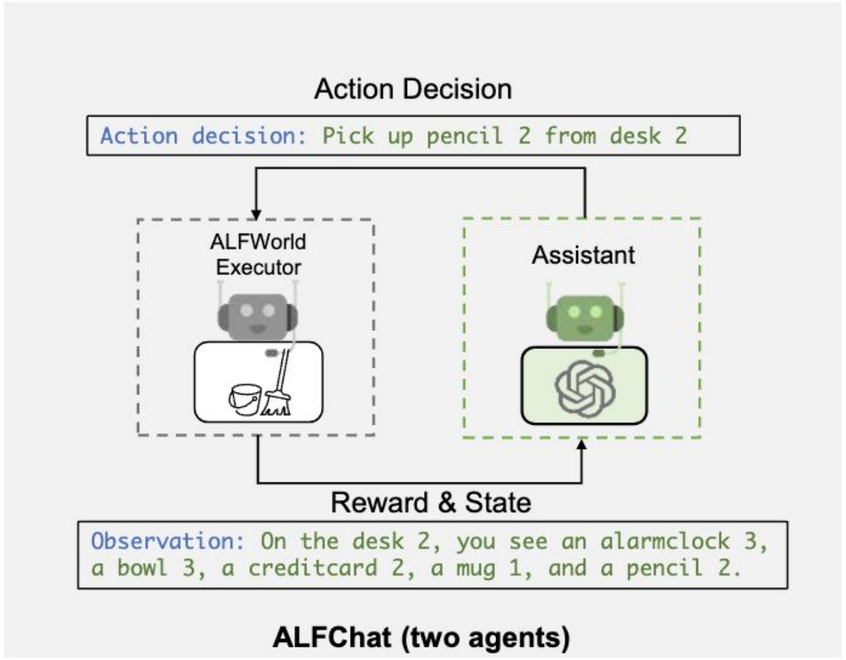
  Set use_spark -> visit newest document

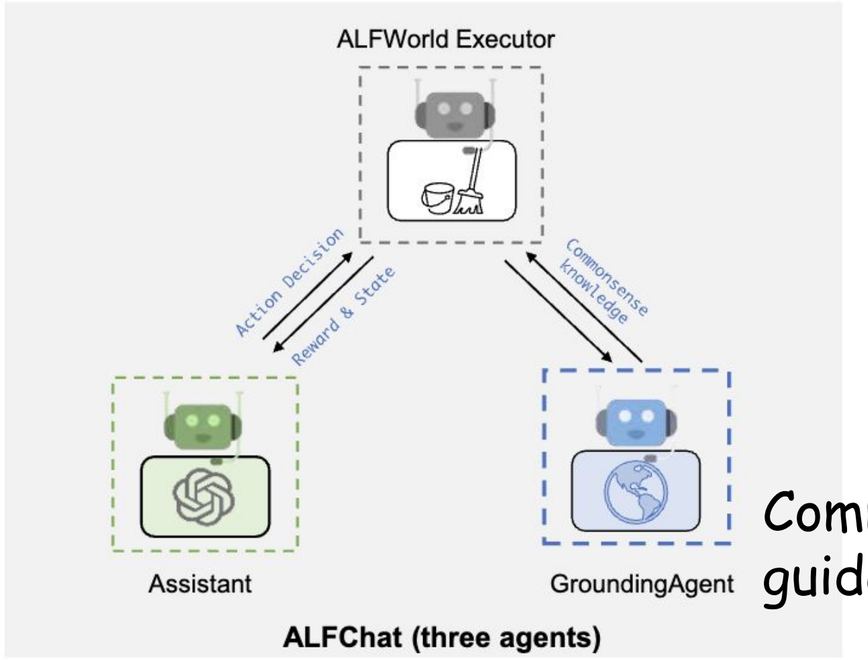
  Enhance efficiency on retrieve -> optimize code quality

APPLICATIONS OF AUTOGEN

Case A3 – Decision making

Benchmark: ALFWorld (simulate real-world household scenes)





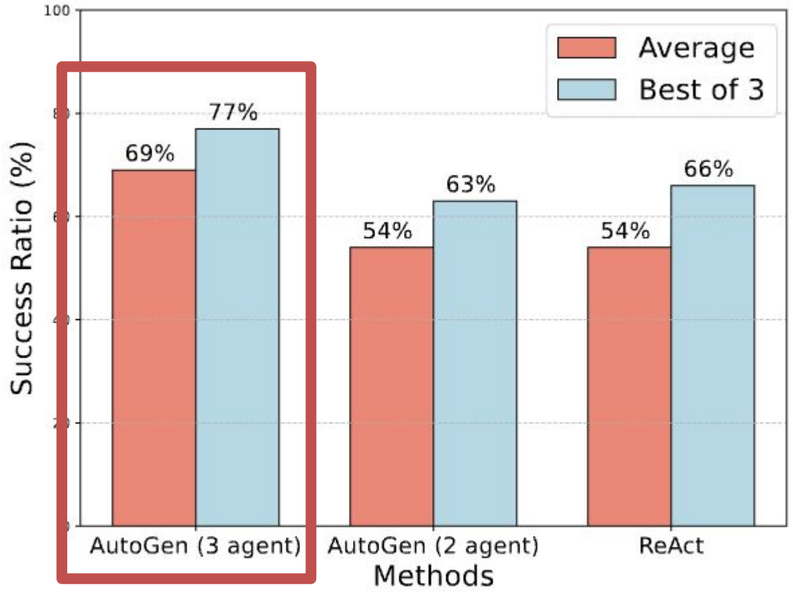
Common sense guide

APPLICATIONS OF AUTOGEN

Case A3 – Decision making

Method	Pick	Clean	Heat	Cool	Look	Pick 2	All
ReAct (avg)	63	52	48	71	61	24	54
ALFChat (2 agents)(avg)	61	58	57	67	50	19	54
ALFChat (3 agents)(avg)	79	64	70	76	78	41	69
ReAct (best of 3)	75	62	61	81	78	35	66
ALFChat (2 agents)(best of 3)	71	61	65	76	67	35	63
AFLChat (3 agents)(best of 3)	92	74	78	86	83	41	77

Table 3: Comparisons between ReAct and the two variants of ALFChat on the ALFWorld benchmark. For each task, we report the success rate out of 3 attempts. Success rate denotes the number of tasks successfully completed by the agent divided by the total number of tasks. The results show that adding a grounding agent significantly improves the task success rate in ALFChat.

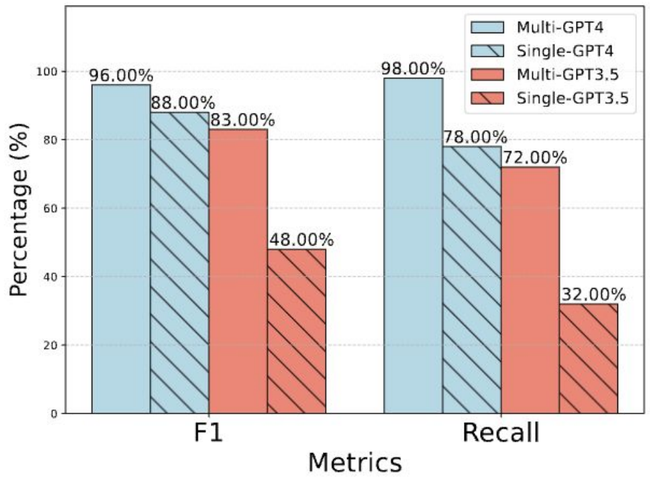


(c) A3: Performance on ALFWorld.

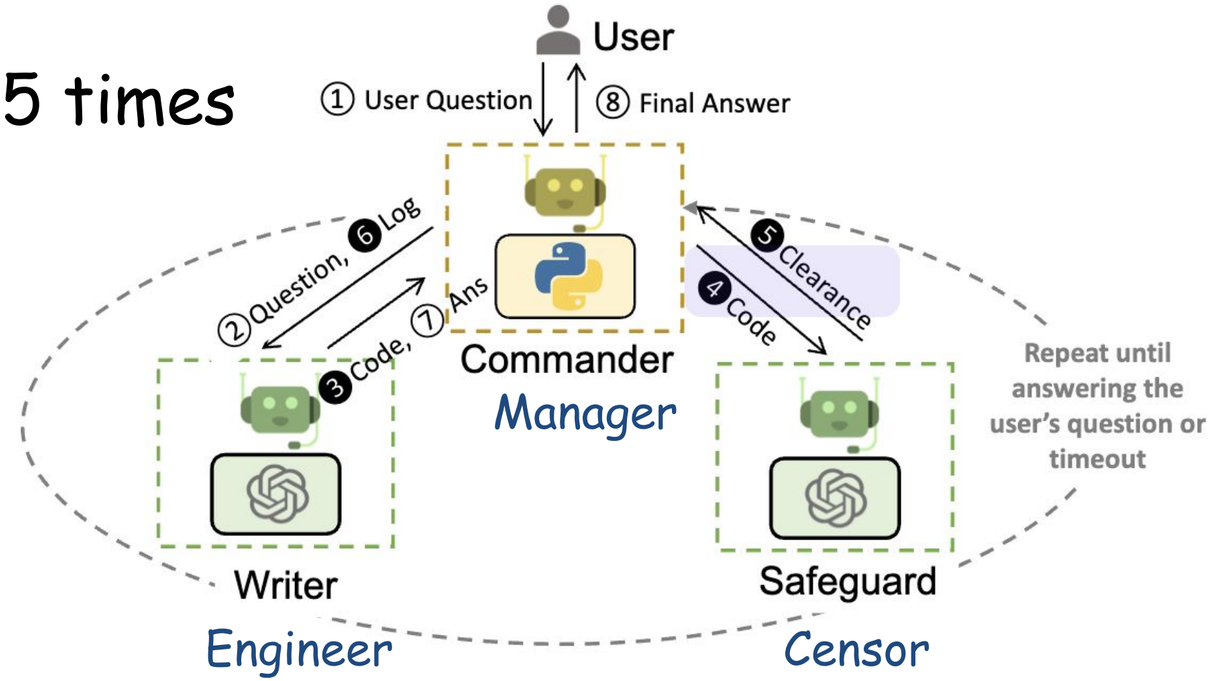
APPLICATIONS OF AUTOGEN

Case A4 – Multiagent coding

- Base on OptiGuide
- Reduce code 430 -> 100 lines
- Solve 3x of user's time
- Reduce user interactions 3-5 times



(d) A4: Performance on OptiGuide.

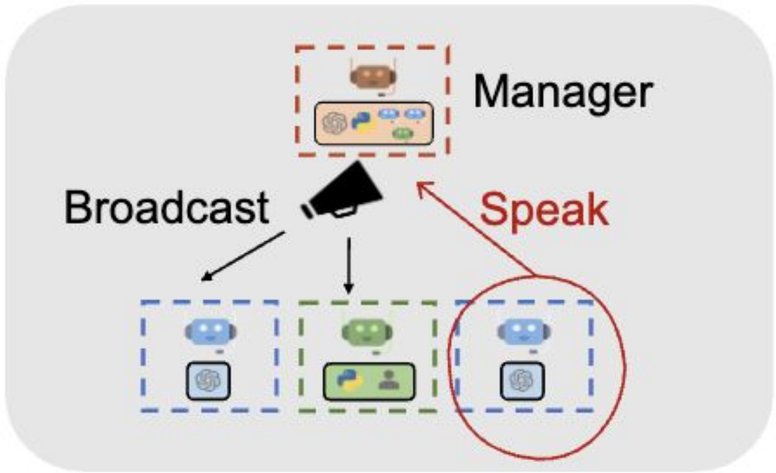


APPLICATIONS OF AUTOGEN

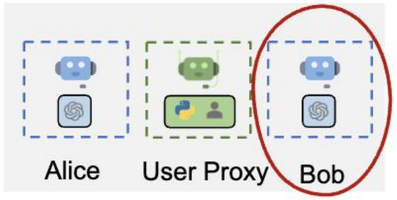
Case A5 - Dynamic Group Chat

Chat base on ongoing conversion

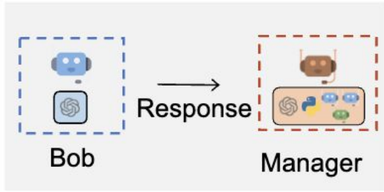
GropChatManager(Manager): conductor of conversion



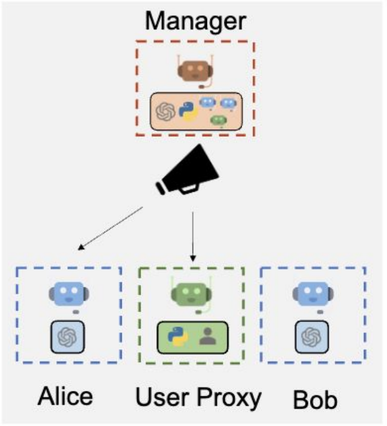
A5. Dynamic Group Chat



1. Select a Speaker



2. Ask the Speaker to Respond



3. Broadcast

APPLICATIONS OF AUTOGEN

Case A5 - Dynamic Group Chat

- Success on 12 tasks

Table 5: Number of successes on the 12 tasks (higher the better).

Model	Two Agent	Group Chat	Group Chat with a task-based speaker selection policy
GPT-3.5-turbo	8	**9**	7
GPT-4	9	**11**	8

- Average termination failure on 12 tasks

Table 6: Average # LLM calls and number of termination failures on the 12 tasks (lower the better)

Model	Two Agent	Group Chat	Group Chat with a task-based speaker selection policy
GPT-3.5-turbo	9.9, 9	5.3, 0	4, 0
GPT-4	6.8, 3	4.5, 0	4, 0

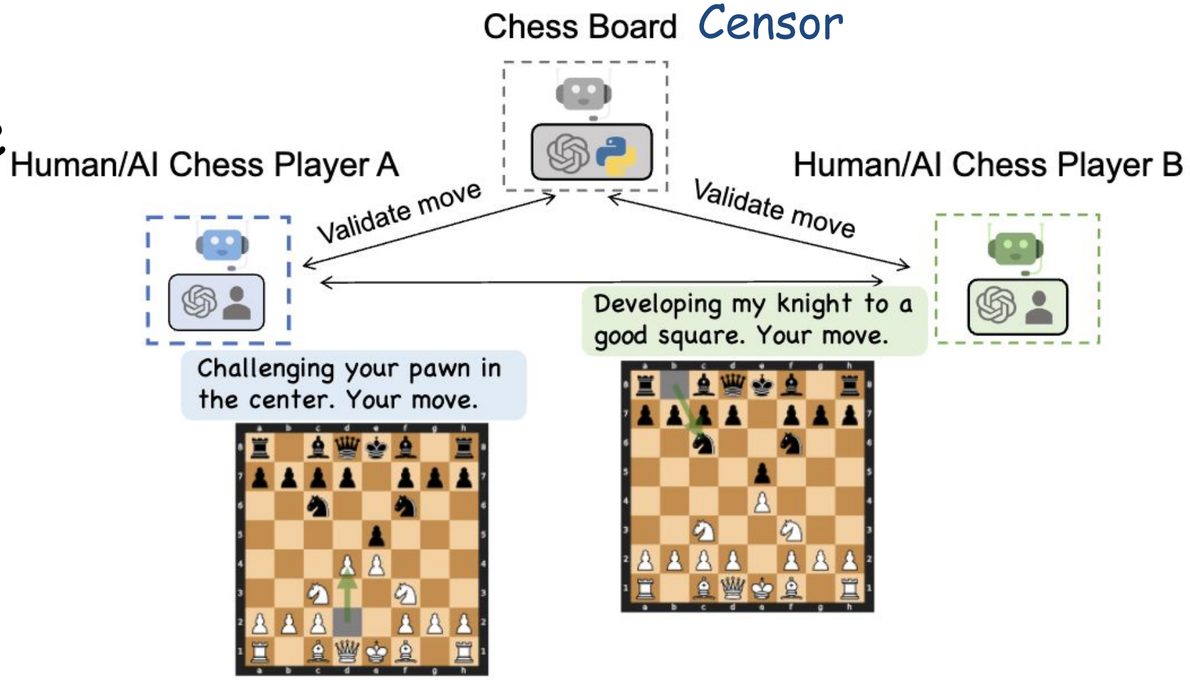
APPLICATIONS OF AUTOGEN

Case A6 Conversational Chess

Board Agent: Censor to ensure move is legal & give info

Player agent: AI/ Human

- Natural dialogue flow
- Accurate rule execution
- Rich interactive experience
- Effective error handling



APPLICATIONS OF AUTOGEN

Case A6 Conversational Chess

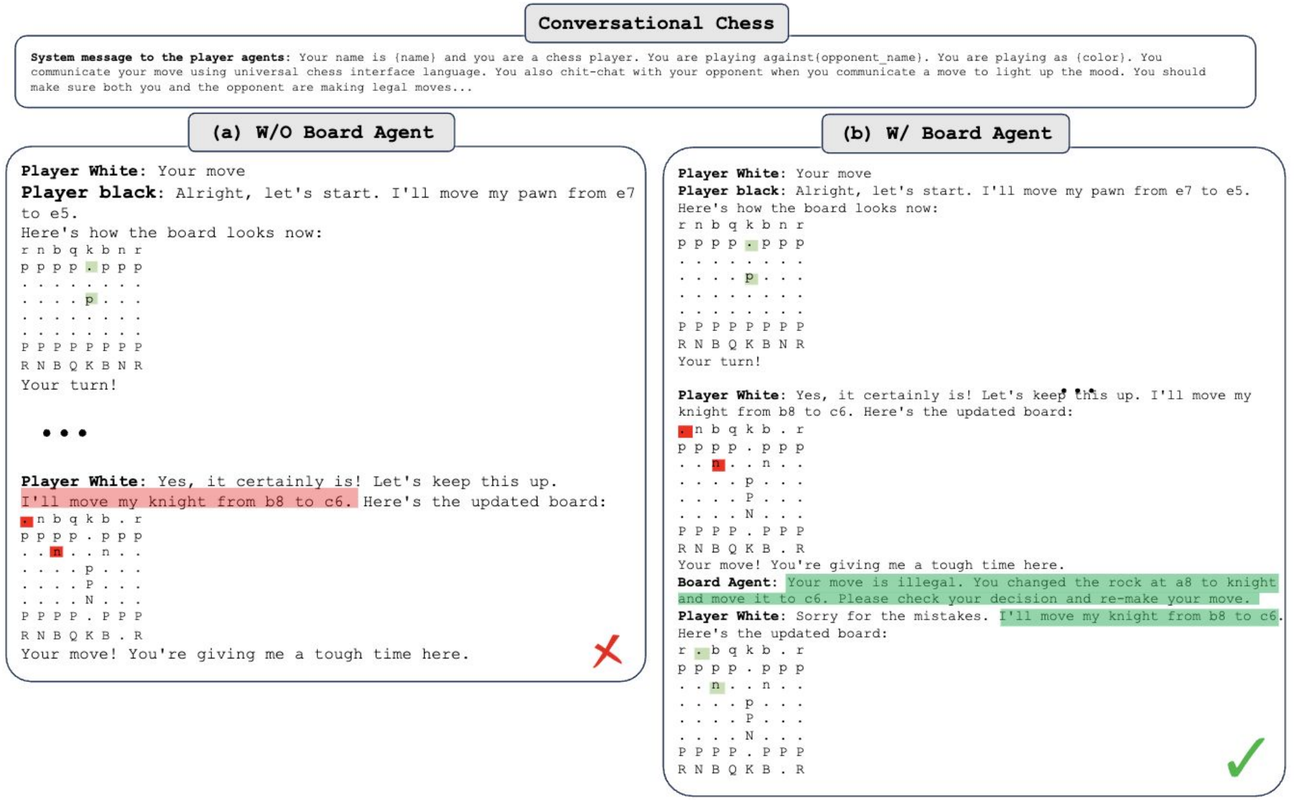


Figure 16: Comparison of two designs–(a) without a board agent, and (b) with a board agent–in Conversational Chess.

Discussion

AutoGen as a general multi-agent conversation framework, has possesses powerful capabilities to empower the development of various complex and diverse LLM applications.

Future improvement of AutoGen :

- Integration of existing agent implementations into multi-agent framework
- Balance between automation and human control
- Agent topology structure and conversation patterns
- Security challenges

Thanks for listening