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 Awadallah[*], **Ryen W. White**[*], **Doug Burger**[*], **Chi Wang**[*1]

[*]Microsoft Research, [†]Pennsylvania State University

[±]University of Washington,[∓]Xidian University

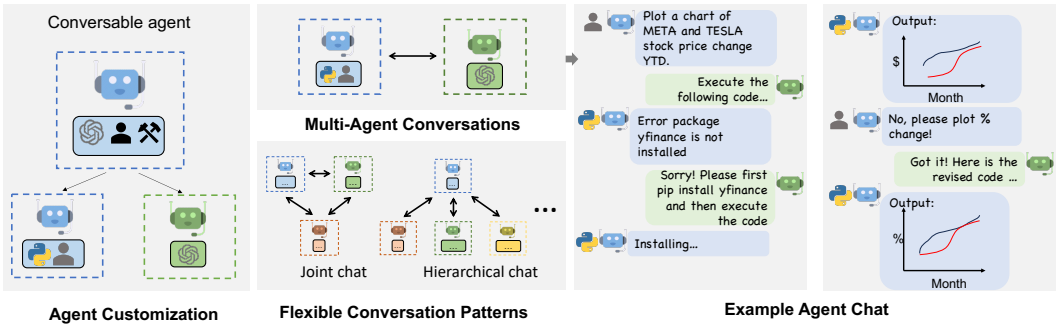


Figure 1: AutoGen enables diverse LLM-based applications using multi-agent conversations. (Left) AutoGen agents are conversable, customizable, and can be based on LLMs, tools, humans, or even a combination of them. (Top-middle) Agents can converse to solve tasks. (Right) They can form a chat, potentially with humans in the loop. (Bottom-middle) The framework supports flexible conversation patterns.

Abstract

AutoGen[2] is an open-source framework that allows developers to build LLM applications via multiple *agents* that can converse with each other to accomplish tasks. AutoGen agents are customizable, *conversable*, and can operate in various modes that employ combinations of LLMs, human inputs, and tools. Using AutoGen, developers can also flexibly define agent interaction behaviors. Both natural language and computer code can be used to program flexible conversation patterns for different applications. AutoGen serves as a generic framework for building diverse applications of various complexities and LLM capacities. Empirical studies demonstrate the effectiveness of the framework in many example applications, with domains ranging from mathematics, coding, question answering, operations research, online decision-making, entertainment, etc.

[1]Corresponding author. Email: auto-gen@outlook.com

[2]https://github.com/microsoft/autogen

1 Introduction

Large language models (LLMs) are becoming a crucial building block in developing powerful *agents* that utilize LLMs for reasoning, tool usage, and adapting to new observations (Yao et al., 2022; Xi et al., 2023; Wang et al., 2023b) in many real-world tasks. Given the expanding tasks that could benefit from LLMs and the growing task complexity, an intuitive approach to scale up the power of agents is to use multiple agents that cooperate. Prior work suggests that multiple agents can help encourage divergent thinking (Liang et al., 2023), improve factuality and reasoning (Du et al., 2023), and provide validation (Wu et al., 2023). In light of the intuition and early evidence of promise, it is intriguing to ask the following question: *how* can we facilitate the development of LLM applications that could span a broad spectrum of domains and complexities based on the multi-agent approach?

Our insight is to use *multi-agent conversations* to achieve it. There are at least three reasons confirming its general feasibility and utility thanks to recent advances in LLMs: First, because chat-optimized LLMs (e.g., GPT-4) show the ability to incorporate feedback, LLM agents can cooperate through *conversations* with each other or human(s), e.g., a dialog where agents provide and seek reasoning, observations, critiques, and validation. Second, because a single LLM can exhibit a broad range of capabilities (especially when configured with the correct prompt and inference settings), conversations between differently configured agents can help combine these broad LLM capabilities in a modular and complementary manner. Third, LLMs have demonstrated ability to solve complex tasks when the tasks are broken into simpler subtasks. Multi-agent conversations can enable this partitioning and integration in an intuitive manner. How can we leverage the above insights and support different applications with the common requirement of coordinating multiple agents, potentially backed by LLMs, humans, or tools exhibiting different capacities? We desire a multi-agent conversation framework with generic abstraction and effective implementation that has the flexibility to satisfy different application needs. Achieving this requires addressing two critical questions: (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? In practice, applications of varying complexities may need distinct sets of agents with specific capabilities, and may require different conversation patterns, such as single- or multi-turn dialogs, different human involvement modes, and static vs. dynamic conversation. Moreover, developers may prefer the flexibility to program agent interactions in natural language or code. Failing to adequately address these two questions would limit the framework's scope of applicability and generality.

While there is contemporaneous exploration of multi-agent approaches,[3] we present `AutoGen`, a generalized multi-agent conversation framework (Figure 1), based on the following new concepts.

1 **Customizable and conversable agents.** `AutoGen` uses a generic design of agents that can leverage LLMs, human inputs, tools, or a combination of them. The result is that developers can easily and quickly create agents with different roles (e.g., agents to write code, execute code, wire in human feedback, validate outputs, etc.) by selecting and configuring a subset of built-in capabilities. The agent's backend can also be readily extended to allow more custom behaviors. To make these agents suitable for multi-agent conversation, every agent is made *conversable* – they can receive, react, and respond to messages. When configured properly, an agent can hold multiple turns of conversations with other agents autonomously or solicit human inputs at certain rounds, enabling human agency and automation. The conversable agent design leverages the strong capability of the most advanced LLMs in taking feedback and making progress via chat and also allows combining capabilities of LLMs in a modular fashion. (Section 2.1)

2 **Conversation programming.** A fundamental insight of `AutoGen` is to simplify and unify complex LLM application workflows as multi-agent conversations. So `AutoGen` adopts a programming paradigm centered around these inter-agent conversations. We refer to this paradigm as *conversation programming*, which streamlines the development of intricate applications via two primary steps: (1) defining a set of conversable agents with specific capabilities and roles (as described above); (2) programming the interaction behavior between agents via conversation-centric *computation* and *control*. Both steps can be achieved via a fusion of natural and programming languages to build applications with a wide range of conversation patterns and agent behaviors. `AutoGen` provides ready-to-use implementations and also allows easy extension and experimentation for both steps. (Section 2.2)

[3] We refer to Appendix A for a detailed discussion.

AutoGen also provides a collection of multi-agent applications created using conversable agents and conversation programming. These applications demonstrate how AutoGen can easily support applications of various complexities and LLMs of various capabilities. Moreover, we perform both evaluation on benchmarks and a pilot study of new applications. The results show that AutoGen can help achieve outstanding performance on many tasks, and enable innovative ways of using LLMs, while reducing development effort. (Section 3 and Appendix D)

2 The AutoGen Framework

To reduce the effort required for developers to create complex LLM applications across various domains, a core design principle of AutoGen is to streamline and consolidate multi-agent workflows using multi-agent conversations. This approach also aims to maximize the reusability of implemented agents. This section introduces the two key concepts of AutoGen: conversable agents and conversation programming.

2.1 Conversable Agents

In AutoGen, a *conversable agent* is an 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. It maintains its internal context based on sent and received messages and can be configured to possess a set of capabilities, e.g., enabled by LLMs, tools, or human input, etc. The agents can act according to programmed behavior patterns described next.

Agent capabilities powered by LLMs, humans, and tools. Since an agent's capabilities directly influence how it processes and responds to messages, AutoGen allows flexibility to endow its agents with various capabilities. AutoGen supports many common composable capabilities for agents, including **1) LLMs.** LLM-backed agents exploit many capabilities of advanced LLMs such as role playing, implicit state inference and progress making conditioned on conversation history, providing feedback, adapting from feedback, and coding. These capabilities can be combined in different ways via novel prompting techniques[4] to increase an agent's skill and autonomy. AutoGen also offers enhanced LLM inference features such as result caching, error handling, message templating, etc., via an enhanced LLM inference layer. **2) Humans.** Human involvement is desired or even essential in many LLM applications. AutoGen lets a human participate in agent conversation via human-backed agents, which could solicit human inputs at certain rounds of a conversation depending on the agent configuration. The default *user proxy* agent allows *configurable* human involvement levels and patterns, e.g., frequency and conditions for requesting human input including the option for humans to skip providing input. **3) Tools.** Tool-backed agents have the capability to execute tools via code execution or function execution. For example, the default user proxy agent in AutoGen is able to execute code suggested by LLMs, or make LLM-suggested function calls.

Agent customization and cooperation. 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. AutoGen allows easy creation of agents with specialized capabilities and roles by reusing or extending the built-in agents. The yellow-shaded area of Figure 2 provides a sketch of the built-in agents in AutoGen. The ConversableAgent class is the highest-level agent abstraction and, by default, can use LLMs, humans, and tools. The AssistantAgent and UserProxyAgent are two pre-configured ConversableAgent subclasses, each representing a common usage mode, i.e., acting as an AI assistant (backed by LLMs) and acting as a human proxy to solicit human input or execute code/function calls (backed by humans and/or tools).

In the example on the right-hand side of Figure 1, an LLM-backed assistant agent and a tool- and human-backed user proxy agent are deployed together to tackle a task. Here, the assistant agent generates a solution with the help of LLMs and passes the solution to the user proxy agent. Then, the user proxy agent solicits human inputs or executes the assistant's code and passes the results as feedback back to the assistant.

[4] Appendix C presents an example of such novel prompting techniques which empowers the default LLM-backed assistant agent in AutoGen to converse with other agents in multi-step problem solving.

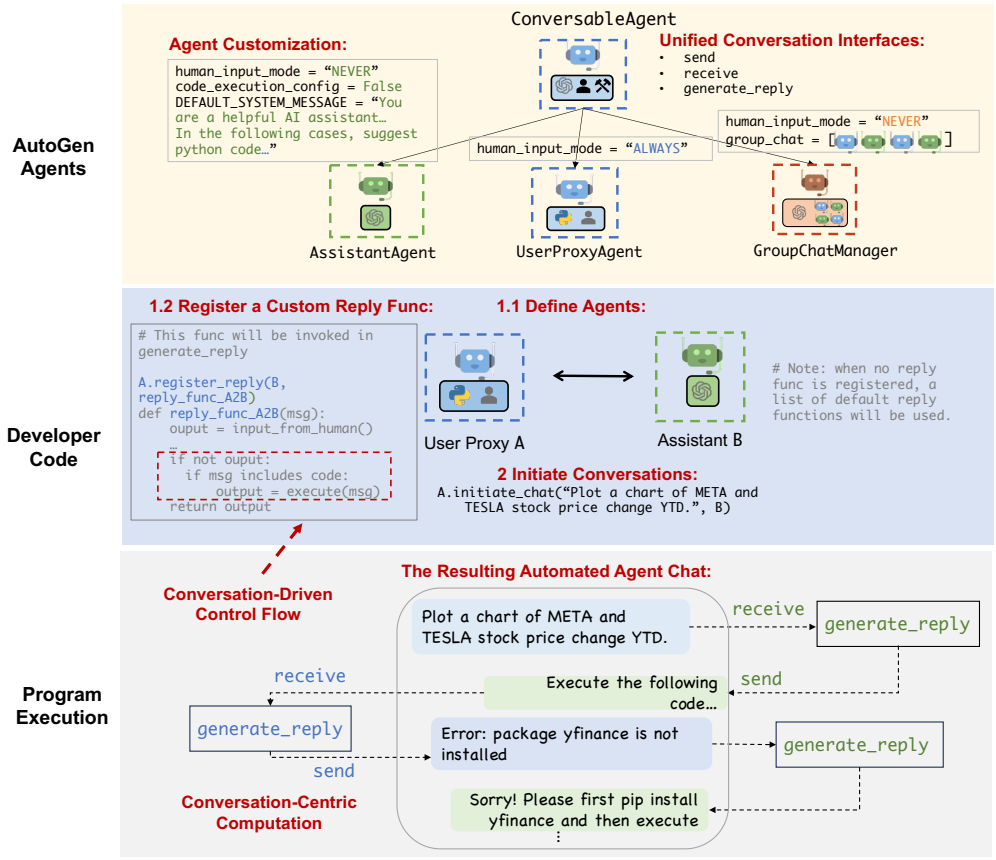


Figure 2: Illustration of how to use `AutoGen` to program a multi-agent conversation. The top sub-figure illustrates the built-in agents provided by `AutoGen`, which have unified conversation interfaces and can be customized. The middle sub-figure shows an example of using `AutoGen` to develop a two-agent system with a custom reply function. The bottom sub-figure illustrates the resulting automated agent chat from the two-agent system during program execution.

By allowing custom agents that can converse with each other, conversable agents in `AutoGen` serve as a useful building block. However, to develop applications where agents make meaningful progress on tasks, developers also need to be able to specify and mold these multi-agent conversations.

2.2 Conversation Programming

As a solution to the above problem, `AutoGen` utilizes *conversation programming*, a paradigm that considers two concepts: the first is *computation* – the actions agents take to compute their response in a multi-agent conversation. And the second is *control flow* – the sequence (or conditions) under which these computations happen. As we will show in the applications section, the ability to program these helps implement many flexible multi-agent conversation patterns. In `AutoGen`, these computations are conversation-centric. An agent takes actions relevant to the conversations it is involved in and its actions result in message passing for consequent conversations (unless a termination condition is satisfied). Similarly, control flow is conversation-driven – the participating agents' decisions on which agents to send messages to and the procedure of computation are functions of the inter-agent conversation. This paradigm helps one to reason intuitively about a complex workflow as agent action taking and conversation message-passing between agents.

Figure 2 provides a simple illustration. The bottom sub-figure shows how individual agents perform their role-specific, conversation-centric computations to generate responses (e.g., via LLM inference calls and code execution). The task progresses through conversations displayed in the dialog box. The middle sub-figure demonstrates a conversation-based control flow. When the assistant receives a message, the user proxy agent typically sends the human input as a reply. If there is no input, it executes any code in the assistant's message instead.

AutoGen features the following design patterns to facilitate conversation programming:

1. **Unified interfaces and auto-reply mechanisms for automated agent chat.** Agents in AutoGen have unified conversation interfaces for performing the corresponding conversation-centric computation, including a `send/receive` function for sending/receiving messages and a `generate_reply` function for taking actions and generating a response based on the received message. AutoGen also introduces and by default adopts an **agent auto-reply** mechanism to realize conversation-driven control: Once an agent receives a message from another agent, it automatically invokes `generate_reply` and sends the reply back to the sender unless a termination condition is satisfied. AutoGen provides built-in reply functions based on LLM inference, code or function execution, or human input. One can also register custom reply functions to customize the behavior pattern of an agent, e.g., chatting with another agent before replying to the sender agent. Under this mechanism, once the reply functions are registered, and the conversation is initialized, the conversation flow is naturally induced, and thus the agent conversation proceeds naturally without any extra control plane, i.e., a special module that controls the conversation flow. For example, with the developer code in the blue-shaded area (marked "Developer Code") of Figure 2, one can readily trigger the conversation among the agents, and the conversation would proceed automatically, as shown in the dialog box in the grey shaded area (marked "Program Execution") of Figure 2. The auto-reply mechanism provides a decentralized, modular, and unified way to define the workflow.
2. **Control by fusion of programming and natural language.** AutoGen allows the usage of programming and natural language in various control flow management patterns: 1) **Natural-language control via LLMs.** In AutoGen, one can control the conversation flow by prompting the LLM-backed agents with natural language. For instance, the default system message of the built-in `AssistantAgent` in AutoGen uses natural language to instruct the agent to fix errors and generate code again if the previous result indicates there are errors. It also guides the agent to confine the LLM output to certain structures, making it easier for other tool-backed agents to consume. For example, instructing the agent to reply with "TERMINATE" when all tasks are completed to terminate the program. More concrete examples of natural language controls can be found in Appendix C. 2) **Programming-language control.** In AutoGen, Python code can be used to specify the termination condition, human input mode, and tool execution logic, e.g., the max number of auto replies. One can also register programmed auto-reply functions to control the conversation flow with Python code, as shown in the code block identified as "Conversation-Driven Control Flow" in Figure 2. 3) **Control transition between natural and programming language.** AutoGen also supports flexible control transition between natural and programming language. One can achieve transition from code to natural-language control by invoking an LLM inference containing certain control logic in a customized reply function; or transition from natural language to code control via LLM-proposed function calls (Eleti et al., 2023).

In the conversation programming paradigm, one can realize multi-agent conversations of diverse patterns. In addition to static conversation with predefined flow, AutoGen also supports dynamic conversation flows with multiple agents. AutoGen provides two general ways to achieve this: 1) Customized `generate_reply` function: within the customized `generate_reply` function, one agent can hold the current conversation while invoking conversations with other agents depending on the content of the current message and context. 2) Function calls: In this approach, LLM decides whether or not to call a particular function depending on the conversation status. By messaging additional agents in the called functions, the LLM can drive dynamic multi-agent conversation. In addition, AutoGen supports more complex dynamic group chat via built-in `GroupChatManager`, which can dynamically select the next speaker and then broadcast its response to other agents. We elaborate on this feature and its application in Section 3. We provide implemented working systems to showcase all these different patterns, with some of them visualized in Figure 3.

3 Applications of AutoGen

We demonstrate six applications using AutoGen (see Figure 3) to illustrate its potential in simplifying the development of high-performance multi-agent applications. These applications are selected based on their real-world relevance (A1, A2, A4, A5, A6), problem difficulty and solving capabilities enabled by AutoGen (A1, A2, A3, A4), and innovative potential (A5, A6). Together, these criteria showcase AutoGen's role in advancing the LLM-application landscape.

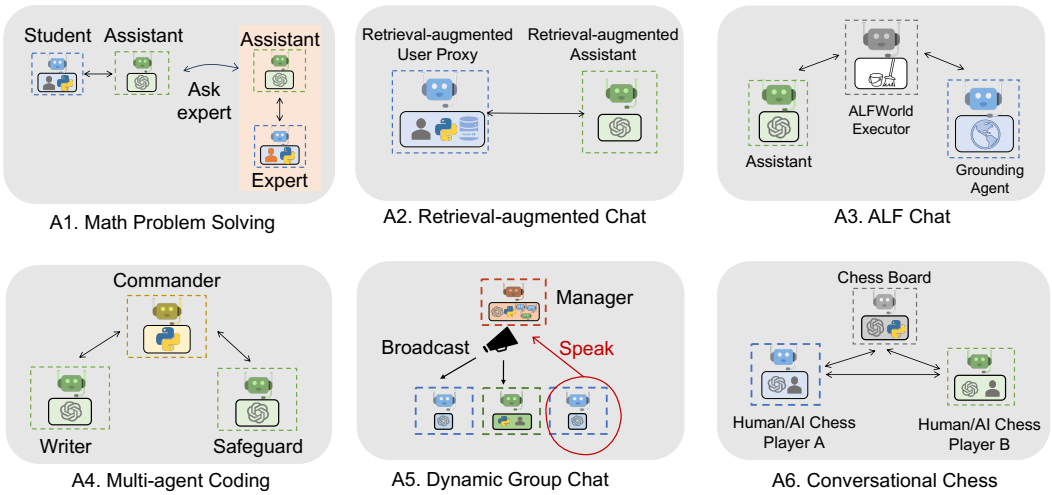


Figure 3: Six examples of diverse applications built using `AutoGen`. Their conversation patterns show `AutoGen`'s flexibility and power.

A1: Math Problem Solving

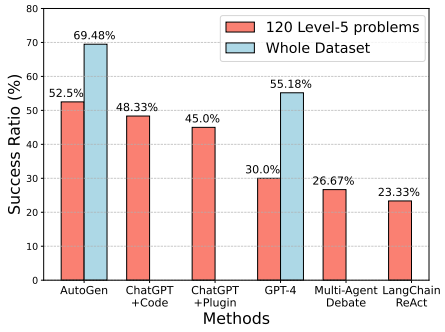
Mathematics is a foundational discipline and the promise of leveraging LLMs to assist with math problem solving opens up a new plethora of applications and avenues for exploration, including personalized AI tutoring, AI research assistance, etc. This section demonstrates how `AutoGen` can help develop LLM applications for math problem solving, showcasing strong performance and flexibility in supporting various problem-solving paradigms.

(**Scenario 1**) We are able to build a system for autonomous math problem solving by directly reusing two built-in agents from `AutoGen`. We evaluate our system and several alternative approaches, including open-source methods such as Multi-Agent Debate (Liang et al., 2023), LangChain ReAct (LangChain, 2023), vanilla GPT-4, and commercial products ChatGPT + Code Interpreter, and ChatGPT + Plugin (Wolfram Alpha), on the MATH (Hendrycks et al., 2021) dataset and summarize the results in Figure 4a. We perform evaluations over 120 randomly selected level-5 problems and on the entire[5] test dataset from MATH. The results show that the built-in agents from `AutoGen` already yield better performance out of the box compared to the alternative approaches, even including the commercial ones. (**Scenario 2**) We also showcase a human-in-the-loop problem-solving process with the help of `AutoGen`. To incorporate human feedback with `AutoGen`, one only needs to set `human_input_mode='ALWAYS'` in the `UserProxyAgent` of the system in scenario 1. We demonstrate that this system can effectively incorporate human inputs to solve challenging problems that cannot be solved without humans. (**Scenario 3**) We further demonstrate a novel scenario where *multiple* human users can participate in the conversations during the problem-solving process. Our experiments and case studies for these scenarios show that AutoGen enables better performance or new experience compared to other solutions we experimented with. Due to the page limit, details of the evaluation, including case studies in three scenarios are in Appendix D.

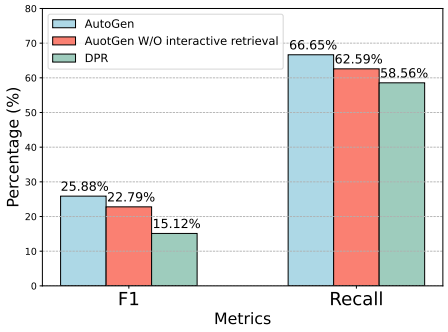
A2: Retrieval-Augmented Code Generation and Question Answering

Retrieval augmentation has emerged as a practical and effective approach for mitigating the intrinsic limitations of LLMs by incorporating external documents. In this section, we employ `AutoGen` to build a Retrieval-Augmented Generation (RAG) system (Lewis et al., 2020; Parvez et al., 2021) named Retrieval-augmented Chat. The system consists of two agents: a Retrieval-augmented User Proxy agent and a Retrieval-augmented Assistant agent, both of which are extended from built-in agents from `AutoGen`. The Retrieval-augmented User Proxy includes a vector database (Chroma,

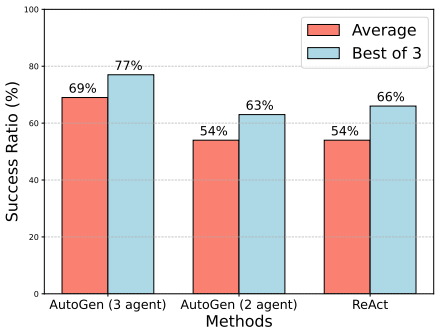
[5]We did not evaluate ChatGPT on the whole dataset since it requires substantial manual effort and is restricted by its hourly message-number limitation. Multi-agent debate and LangChain ReAct were also not evaluated since they underperformed vanilla GPT-4 on the smaller test set.



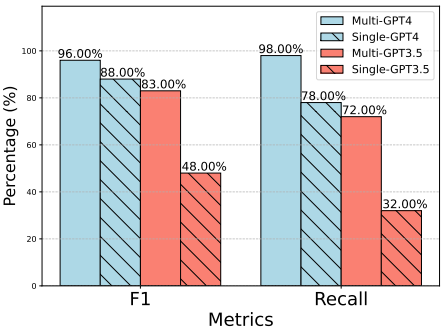
(a) A1: Performance on MATH (w/ GPT-4).



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



(c) A3: Performance on ALFWorld.



(d) A4: Performance on OptiGuide.

Figure 4: Performance on four applications A1-A4. (a) shows that `AutoGen` agents can be used out of the box to achieve the most competitive performance on math problem solving tasks; (b) shows that `AutoGen` can be used to realize effective retrieval augmentation and realize a novel interactive retrieval feature to boost performance on Q&A tasks; (c) shows that `AutoGen` can be used to introduce a three-agent system with a grounding agent to improve performance on ALFWorld; (d) shows that a multi-agent design is helpful in boosting performance in coding tasks that need safeguards.

2023) with SentenceTransformers (Reimers & Gurevych, 2019) as the context retriever. A detailed workflow description of the Retrieval-augmented Chat is provided in Appendix D.

We evaluate Retrieval-augmented Chat in both question-answering and code-generation scenarios. (**Scenario 1**) We first perform an evaluation regarding natural question answering on the Natural Questions dataset (Kwiatkowski et al., 2019) and report results in Figure 4b. In this evaluation, we compare our system with DPR (Dense Passage Retrieval) following an existing evaluation[6] practice (Adlakha et al., 2023). Leveraging the conversational design and natural-language control, `AutoGen` introduces a novel *interactive retrieval* feature in this application: whenever the retrieved context does not contain the information, instead of terminating, the LLM-based assistant would reply "*Sorry, I cannot find any information about... UPDATE CONTEXT.*" which will invoke more retrieval attempts. We conduct an ablation study in which we prompt the assistant agent to say *"I don't know"* instead of *"UPDATE CONTEXT."* in cases where relevant information is not found, and report results in Figure 4b. The results show that the interactive retrieval mechanism indeed plays a non-trivial role in the process. We give a concrete example and results using this appealing feature in Appendix D. (**Scenario 2**) We further demonstrate how Retrieval-augmented Chat aids in generating code based on a given codebase that contains code not included in GPT-4's training data. Evaluation and demonstration details for both scenarios are included in Appendix D.

[6]The results of DPR with GPT-3.5 shown in Figure 4b are from (Adlakha et al., 2023). We use GPT-3.5 as a shorthand for GPT-3.5-turbo.

A3: Decision Making in Text World Environments

In this subsection, we demonstrate how `AutoGen` can be used to develop effective applications that involve interactive or online decision making. We perform the study using the ALFWorld (Shridhar et al., 2021) benchmark, which includes a diverse collection of synthetic language-based interactive decision-making tasks in household environments.

With `AutoGen`, we implemented a two-agent system to solve tasks from ALFWorld. It consists of an LLM-backed assistant agent responsible for suggesting plans to conduct a task and an executor agent responsible for executing actions in the ALFWorld environments. This system integrates ReAct prompting (Yao et al., 2022), and is able to achieve similar performance. A common challenge encountered in both ReAct and the `AutoGen`-based two-agent system is their occasional inability to leverage basic commonsense knowledge about the physical world. This deficiency can lead to the system getting stuck in a loop due to repetitive errors. Fortunately, the modular design of `AutoGen` allows us to address this issue effectively: With `AutoGen`, we are able to introduce a grounding agent, which supplies crucial commonsense knowledge–such as *"You must find and take the object before you can examine it. You must go to where the target object is before you can use it."*–whenever the system exhibits early signs of recurring errors. It significantly enhances the system's ability to avoid getting entangled in error loops. We compare the task-solving performance of the two variants of our system with GPT-3.5-turbo and ReAct[7] on the 134 unseen tasks from ALFWorld and report results in Figure 4c. The results show that introducing a grounding agent could bring in a 15% performance gain on average. Upon examining the systems' outputs, we observe that the grounding agent, by delivering background commonsense knowledge at the right junctures, significantly mitigated the tendency of the system to persist with a flawed plan, thereby avoiding the creation of error loops. For an example trajectory comparing the systems see Appendix D, Figure 10.

A4: Multi-Agent Coding

In this subsection, we use `AutoGen` to build a multi-agent coding system based on OptiGuide (Li et al., 2023a), a system that excels at writing code to interpret optimization solutions and answer user questions, such as exploring the implications of changing a supply-chain decision or understanding why the optimizer made a particular choice. The second sub-figure of Figure 3 shows the `AutoGen`-based implementation. The workflow is as follows: the end user sends questions, such as "*What if we prohibit shipping from supplier 1 to roastery 2?*" to the Commander agent. The Commander coordinates with two assistant agents, including the Writer and the Safeguard, to answer the question. The Writer will craft code and send the code to the Commander. After receiving the code, the Commander checks the code safety with the Safeguard; if cleared, the Commander will use external tools (e.g., Python) to execute the code, and request the Writer to interpret the execution results. For instance, the writer may say "*if we prohibit shipping from supplier 1 to roastery 2, the total cost would increase by 10.5%.*" The Commander then provides this concluding answer to the end user. If, at a particular step, there is an exception, e.g., security red flag raised by Safeguard, the Commander redirects the issue back to the Writer with debugging information. The process might be repeated multiple times until the user's question is answered or timed-out.

With `AutoGen` the core workflow code for OptiGuide was reduced from over 430 lines to 100 lines, leading to significant productivity improvement. We provide a detailed comparison of user experience with ChatGPT+Code Interpreter and `AutoGen`-based OptiGuide in Appendix D, where we show that `AutoGen`-based OptiGuide could save around 3x of user's time and reduce user interactions by 3 - 5 times on average. We also conduct an ablation showing that multi-agent abstraction is necessary. Specifically, we construct a single-agent approach where a single agent conducts both the code-writing and safeguard processes. We tested the single- and multi-agent approaches on a dataset of 100 coding tasks, which is crafted to include equal numbers of safe and unsafe tasks. Evaluation results as reported in Figure 4d show that the multi-agent design boosts the F-1 score in identifying unsafe code by 8% (with GPT-4) and 35% (with GPT-3.5-turbo).

[7] Results of ReAct are obtained by directly running its official code with default settings. The code uses `text-davinci-003` as backend LM and does not support GPT-3.5-turbo or GPT-4.

A5: Dynamic Group Chat

AutoGen provides native support for a *dynamic group chat* communication pattern, in which participating agents share the same context and converse with the others in a dynamic manner instead of following a pre-defined order. Dynamic group chat relies on ongoing conversations to guide the flow of interaction among agents. These make dynamic group chat ideal for situations where collaboration without strict communication order is beneficial. In AutoGen, the GroupChatManager class serves as the conductor of conversation among agents and repeats the following three steps: dynamically selecting a speaker, collecting responses from the selected speaker, and broadcasting the message (Figure 3-A5). For the dynamic speaker-selection component, we use a role-play style prompt. Through a pilot study on 12 manually crafted complex tasks, we observed that compared to a prompt that is purely based on the task, utilizing a role-play prompt often leads to more effective consideration of both conversation context and role alignment during the problem-solving and speaker-selection process. Consequently, this leads to a higher success rate and fewer LLM calls. We include detailed results in Appendix D.

A6: Conversational Chess

Using AutoGen, we developed Conversational Chess, a natural language interface game shown in the last sub-figure of Figure 3. It features built-in agents for players, which can be human or LLM, and a third-party board agent to provide information and validate moves based on standard rules.

With AutoGen, we enabled two essential features: (1) Natural, flexible, and engaging game dynamics, enabled by the customizable agent design in AutoGen. Conversational Chess supports a range of game-play patterns, including AI-AI, AI-human, and human-human, with seamless switching between these modes during a single game. An illustrative example of these entertaining game dynamics can be found in Figure 15, Appendix D. (2) Grounding, which is a crucial aspect to maintain game integrity. During gameplay, the board agent checks each proposed move for legality; if a move is invalid, the agent responds with an error, prompting the player agent to re-propose a legal move before continuing. This process ensures that only valid moves are played and helps maintain a consistent gaming experience. As an ablation study, we removed the board agent and instead only relied on a relevant prompt *"you should make sure both you and the opponent are making legal moves"* to ground their move. The results highlighted that without the board agent, illegitimate moves caused game disruptions. The modular design offered flexibility, allowing swift adjustments to the board agent in response to evolving game rules or varying chess rule variants. A comprehensive demonstration of this ablation study is in Appendix D.

4 Discussion

We introduced an open-source library, AutoGen, that incorporates the paradigms of conversable agents and conversation programming. This library utilizes capable agents that are well-suited for multi-agent cooperation. It features a unified conversation interface among the agents, along with an auto-reply mechanisms, which help establish an agent-interaction interface that capitalizes on the strengths of chat-optimized LLMs with broad capabilities while accommodating a wide range of applications. AutoGen serves as a general framework for creating and experimenting with multi-agent systems that can easily fulfill various practical requirements, such as reusing, customizing, and extending existing agents, as well as programming conversations between them.

Our experiments, as detailed in Section 3, demonstrate that this approach offers numerous benefits. The adoption of AutoGen has resulted in improved performance (over state-of-the-art approaches), reduced development code, and decreased manual burden for existing applications. It offers flexibility to developers, as demonstrated in A1 (scenario 3), A5, and A6, where AutoGen enables multi-agent chats to follow a dynamic pattern rather than fixed back-and-forth interactions. It allows humans to engage in activities alongside multiple AI agents in a conversational manner. Despite the complexity of these applications (most involving more than two agents or dynamic multi-turn agent cooperation), the implementation based on AutoGen remains straightforward. Dividing tasks among separate agents promotes modularity. Furthermore, since each agent can be developed, tested, and maintained separately, this approach simplifies overall development and code management.

Although this work is still in its early experimental stages, it paves the way for numerous future directions and research opportunities. For instance, we can explore effective integration of existing agent implementations into our multi-agent framework and investigate the optimal balance between automation and human control in multi-agent workflows. As we further develop and refine `AutoGen`, we aim to investigate which strategies, such as agent topology and conversation patterns, lead to the most effective multi-agent conversations while optimizing the overall efficiency, among other factors. While increasing the number of agents and other degrees of freedom presents opportunities for tackling more complex problems, it may also introduce new safety challenges that require additional studies and careful consideration.

We provide more discussion in Appendix B, including guidelines for using `AutoGen` and direction of future work. We hope `AutoGen` will help improve many LLM applications in terms of speed of development, ease of experimentation, and overall effectiveness and safety. We actively welcome contributions from the broader community.

Ethics statement

There are several potential ethical considerations that could arise from the development and use of the `AutoGen` framework.

- Privacy and Data Protection: The framework allows for human participation in conversations between agents. It is important to ensure that user data and conversations are protected, and that developers use appropriate measures to safeguard privacy.
- Bias and Fairness: LLMs have been shown to exhibit biases present in their training data (Navigli et al., 2023). When using LLMs in the `AutoGen` framework, it is crucial to address and mitigate any biases that may arise in the conversations between agents. Developers should be aware of potential biases and take steps to ensure fairness and inclusivity.
- Accountability and Transparency: As discussed in the future work section, as the framework involves multiple agents conversing and cooperating, it is important to establish clear accountability and transparency mechanisms. Users should be able to understand and trace the decision-making process of the agents involved in order to ensure accountability and address any potential issues or biases.
- Trust and Reliance: `AutoGen` leverages human understanding and intelligence while providing automation through conversations between agents. It is important to consider the impact of this interaction on user experience, trust, and reliance on AI systems. Clear communication and user education about the capabilities and limitations of the system will be essential (Cai et al., 2019).
- Unintended Consequences: As discussed before, the use of multi-agent conversations and automation in complex tasks may have unintended consequences. In particular, allowing LLM agents to make changes in external environments through code execution or function calls, such as installing packages, could be risky. Developers should carefully consider the potential risks and ensure that appropriate safeguards are in place to prevent harm or negative outcomes.

Acknowledgements

The work presented in this report was made possible through discussions and feedback from Peter Lee, Johannes Gehrke, Eric Horvitz, Steven Lucco, Umesh Madan, Robin Moeur, Piali Choudhury, Saleema Amershi, Adam Fourney, Victor Dibia, Guoqing Zheng, Corby Rosset, Ricky Loynd, Ece Kamar, Rafah Hosn, John Langford, Ida Momennejad, Brian Krabach, Taylor Webb, Shanka Subhra Mondal, Wei-ge Chen, Robert Gruen, Yinan Li, Yue Wang, Suman Nath, Tanakorn Leesatapornwongsa, Xin Wang, Shishir Patil, Tianjun Zhang, Saehan Jo, Ishai Menache, Kontantina Mellou, Runlong Zhou, Feiran Jia, Hamed Khanpour, Hamid Palangi, Srinagesh Sharma, Julio Albinati Cortez, Amin Saied, Yuzhe Ma, Dujian Ding, Linyong Nan, Prateek Yadav, Shannon Shen, Ankur Mallick, Mark Encarnación, Lars Liden, Tianwei Yue, Julia Kiseleva, Anastasia Razdaibiedina, and Luciano Del Corro. Qingyun Wu would like to acknowledge the funding and research support from the College of Information Science and Technology at Penn State University.

References

Vaibhav Adlakha, Parishad BehnamGhader, Xing Han Lu, Nicholas Meade, and Siva Reddy. Evaluating correctness and faithfulness of instruction-following models for question answering. *arXiv preprint arXiv:2307.16877*, 2023.

Saleema Amershi, Dan Weld, Mihaela Vorvoreanu, Adam Fourney, Besmira Nushi, Penny Collisson, Jina Suh, Shamsi Iqbal, Paul N Bennett, Kori Inkpen, et al. Guidelines for human-ai interaction. In *Proceedings of the 2019 chi conference on human factors in computing systems*, 2019.

Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. Concrete problems in ai safety, 2016.

AutoGPT. Documentation — auto-gpt. `https://docs.agpt.co/`, 2023.

BabyAGI. Github — babyagi. `https://github.com/yoheinakajima/babyagi`, 2023.

Carrie J. Cai, Samantha Winter, David F. Steiner, Lauren Wilcox, and Michael Terry. "hello ai": Uncovering the onboarding needs of medical practitioners for human-ai collaborative decision-making. *Proceedings of the ACM on Human-Computer Interaction*, 2019.

Tianle Cai, Xuezhi Wang, Tengyu Ma, Xinyun Chen, and Denny Zhou. Large language models as tool makers. *arXiv preprint arXiv:2305.17126*, 2023.

Chroma. Chromadb. `https://github.com/chroma-core/chroma`, 2023.

Victor Dibia. LIDA: A tool for automatic generation of grammar-agnostic visualizations and infographics using large language models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, Toronto, Canada, July 2023. Association for Computational Linguistics.

Yihong Dong, Xue Jiang, Zhi Jin, and Ge Li. Self-collaboration code generation via chatgpt. *arXiv preprint arXiv:2304.07590*, 2023.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate. *arXiv preprint arXiv:2305.14325*, 2023.

Atty Eleti, Jeff Harris, and Logan Kilpatrick. Function calling and other api updates. `https://openai.com/blog/function-calling-and-other-api-updates`, 2023.

Guidance. Guidance. `https://github.com/guidance-ai/guidance`, 2023.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021.

Sirui Hong, Xiawu Zheng, Jonathan Chen, Yuheng Cheng, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, et al. Metagpt: Meta programming for multi-agent collaborative framework. *arXiv preprint arXiv:2308.00352*, 2023.

Eric Horvitz. Principles of mixed-initiative user interfaces. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, 1999.

HuggingFace. Transformers agent. `https://huggingface.co/docs/transformers/transformers_agents`, 2023.

Geunwoo Kim, Pierre Baldi, and Stephen McAleer. Language models can solve computer tasks. *arXiv preprint arXiv:2303.17491*, 2023.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, et al. Natural questions: a benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 2019.

LangChain. Introduction — langchain. https://python.langchain.com/en/latest/index.html, 2023.

Mike Lewis, Denis Yarats, Yann N Dauphin, Devi Parikh, and Dhruv Batra. Deal or no deal? end-to-end learning for negotiation dialogues. *arXiv preprint arXiv:1706.05125*, 2017.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 2020.

Beibin Li, Konstantina Mellou, Bo Zhang, Jeevan Pathuri, and Ishai Menache. Large language models for supply chain optimization. *arXiv preprint arXiv:2307.03875*, 2023a.

Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. Camel: Communicative agents for "mind" exploration of large scale language model society, 2023b.

Tian Liang, Zhiwei He, Wenxiang Jiao, Xing Wang, Yan Wang, Rui Wang, Yujiu Yang, Zhaopeng Tu, and Shuming Shi. Encouraging divergent thinking in large language models through multi-agent debate, 2023.

Evan Zheran Liu, Kelvin Guu, Panupong Pasupat, Tianlin Shi, and Percy Liang. Reinforcement learning on web interfaces using workflow-guided exploration. *arXiv preprint arXiv:1802.08802*, 2018.

Jerry Liu. LlamaIndex, November 2022. URL https://github.com/jerryjliu/llama_index.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Roberto Navigli, Simone Conia, and Björn Ross. Biases in large language models: Origins, inventory and discussion. *ACM Journal of Data and Information Quality*, 2023.

OpenAI. ChatGPT plugins. https://openai.com/blog/chatgpt-plugins, 2023.

Joon Sung Park, Joseph C O'Brien, Carrie J Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. Generative agents: Interactive simulacra of human behavior. *arXiv preprint arXiv:2304.03442*, 2023.

Md Rizwan Parvez, Wasi Uddin Ahmad, Saikat Chakraborty, Baishakhi Ray, and Kai-Wei Chang. Retrieval augmented code generation and summarization. *arXiv preprint arXiv:2108.11601*, 2021.

Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. Gorilla: Large language model connected with massive apis. *arXiv preprint arXiv:2305.15334*, 2023.

Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 11 2019. URL https://arxiv.org/abs/1908.10084.

Semantic-Kernel. Semantic kernel. https://github.com/microsoft/semantic-kernel, 2023.

Bokui Shen, Fei Xia, Chengshu Li, Roberto Martín-Martín, Linxi Fan, Guanzhi Wang, Claudia Pérez-D'Arpino, Shyamal Buch, Sanjana Srivastava, Lyne Tchapmi, et al. igibson 1.0: A simulation environment for interactive tasks in large realistic scenes. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2021.

Tianlin Shi, Andrej Karpathy, Linxi Fan, Jonathan Hernandez, and Percy Liang. World of bits: An open-domain platform for web-based agents. In *International Conference on Machine Learning*. PMLR, 2017.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. ALFWorld: Aligning Text and Embodied Environments for Interactive Learning. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021. URL https://arxiv.org/abs/2010.03768.

Oriol Vinyals, Timo Ewalds, Sergey Bartunov, Petko Georgiev, Alexander Sasha Vezhnevets, Michelle Yeo, Alireza Makhzani, Heinrich Küttler, John Agapiou, Julian Schrittwieser, et al. Starcraft ii: A new challenge for reinforcement learning. *arXiv preprint arXiv:1708.04782*, 2017.

Chi Wang, Qingyun Wu, Markus Weimer, and Erkang Zhu. Flaml: A fast and lightweight automl library. *Proceedings of Machine Learning and Systems*, 2021.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023a.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, et al. A survey on large language model based autonomous agents. *arXiv preprint arXiv:2308.11432*, 2023b.

Daniel S. Weld and Oren Etzioni. The first law of robotics (a call to arms). In *AAAI Conference on Artificial Intelligence*, 1994.

Max Woolf. Langchain problem. https://minimaxir.com/2023/07/langchain-problem/, 2023.

Yiran Wu, Feiran Jia, Shaokun Zhang, Qingyun Wu, Hangyu Li, Erkang Zhu, Yue Wang, Yin Tat Lee, Richard Peng, and Chi Wang. An empirical study on challenging math problem solving with gpt-4. *arXiv preprint arXiv:2306.01337*, 2023.

Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, et al. The rise and potential of large language model based agents: A survey. *arXiv preprint arXiv:2309.07864*, 2023.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

A Related Work

We examine existing LLM-based agent systems or frameworks that can be used to build LLM applications. We categorize the related work into single-agent and multi-agent systems and specifically provide a summary of differentiators comparing `AutoGen` with existing multi-agent systems in Table 1. Note that many of these systems are evolving open-source projects, so the remarks and statements about them may only be accurate as of the time of writing. We refer interested readers to detailed LLM-based agent surveys (Xi et al., 2023; Wang et al., 2023b)

Single-Agent Systems:

- **AutoGPT**: AutoGPT is an open-source implementation of an AI agent that attempts to autonomously achieve a given goal (AutoGPT, 2023). It follows a single-agent paradigm in which it augments the AI model with many useful tools, and does not support multi-agent collaboration.
- **ChatGPT+ (with code interpreter or plugin)**: ChatGPT, a conversational AI service or agent, can now be used alongside a code interpreter or plugin (currently available only under the premium subscription plan ChatGPT Plus) (OpenAI, 2023). The code interpreter enables ChatGPT to execute code, while the plugin enhances ChatGPT with a wide range of curated tools.
- **LangChain Agents**: LangChain is a general framework for developing LLM-based applications (LangChain, 2023). LangChain Agents is a subpackage for using an LLM to choose a sequence of actions. There are various types of agents in LangChain Agents, with the ReAct agent being a notable example that combines reasoning and acting when using LLMs (mainly designed for LLMs prior to ChatGPT) (Yao et al., 2022). All agents provided in LangChain Agents follow a single-agent paradigm and are not inherently designed for communicative and collaborative modes. A significant summary of its limitations can be found in (Woolf, 2023). Due to these limitations, even the multi-agent systems in LangChain (e.g., re-implementation of CAMEL) are not based on LangChain Agents but are implemented from scratch. Their connection to LangChain lies in the use of basic orchestration modules provided by LangChain, such as AI models wrapped by LangChain and the corresponding interface.
- **Transformers Agent**: Transformers Agent (HuggingFace, 2023) is an experimental natural-language API built on the transformers repository. It includes a set of curated tools and an agent to interpret natural language and use these tools. Similar to AutoGPT, it follows a single-agent paradigm and does not support agent collaboration.

`AutoGen` differs from the single-agent systems above by supporting multi-agent LLM applications.

Multi-Agent Systems:

- **BabyAGI**: BabyAGI (BabyAGI, 2023) is an example implementation of an AI-powered task management system in a Python script. In this implemented system, multiple LLM-based agents are used. For example, there is an agent for creating new tasks based on the objective and the result of the previous task, an agent for prioritizing the task list, and an agent for completing tasks/sub-tasks. As a multi-agent system, BabyAGI adopts a static agent conversation pattern, i.e., a predefined order of agent communication, while `AutoGen` supports both static and dynamic conversation patterns and additionally supports tool usage and human involvement.
- **CAMEL**: CAMEL (Li et al., 2023b) is a communicative agent framework. It demonstrates how role playing can be used to let chat agents communicate with each other for task completion. It also records agent conversations for behavior analysis and capability understanding. An Inception-prompting technique is used to achieve autonomous cooperation between agents. Unlike `AutoGen`, CAMEL does not natively support tool usage, such as code execution. Although it is proposed as an infrastructure for multi-agent conversation, it only supports static conversation patterns, while `AutoGen` additionally supports dynamic conversation patterns.
- **Multi-Agent Debate:** Two recent works investigate and show that multi-agent debate is an effective way to encourage divergent thinking in LLMs (Liang et al., 2023) and to improve the factuality and reasoning of LLMs (Du et al., 2023). In both works, multiple LLM inference instances are constructed as multiple agents to solve problems with agent debate. Each agent is simply an LLM inference instance, while no tool or human is involved, and the inter-agent conversation needs to follow a pre-defined order. These works attempt to build LLM applications with multi-agent conversation, while `AutoGen`, designed as a generic infrastructure, can be used to facilitate this development and enable more applications with dynamic conversation patterns.

- **MetaGPT**: MetaGPT (Hong et al., 2023) is a specialized LLM application based on a multi-agent conversation framework for automatic software development. They assign different roles to GPTs to collaboratively develop software. They differ from `AutoGen` by being specialized solutions to a certain scenario, while `AutoGen` is a generic infrastructure to facilitate building applications for various scenarios.

There are a few other specialized single-agent or multi-agent systems, such as Voyager (Wang et al., 2023a) and Generative Agents (Park et al., 2023), which we skip due to lower relevance. In Table 1, we summarize differences between `AutoGen` and the most relevant multi-agent systems.

Table 1: Summary of differences between `AutoGen` and other related multi-agent systems. **infrastructure**: whether the system is designed as a generic infrastructure for building LLM applications. **conversation pattern**: the types of patterns supported by the implemented systems. Under the 'static' pattern, agent topology remains unchanged regardless of different inputs. `AutoGen` allows flexible conversation patterns, including both static and dynamic patterns that can be customized based on different application needs. **execution-capable**: whether the system can execute LLM-generated code; **human involvement**: whether (and how) the system allows human participation during the execution process of the system. `AutoGen` allows flexible human involvement in multi-agent conversation with the option for humans to skip providing inputs.

Aspect	`AutoGen`	Multi-agent Debate	CAMEL	BabyAGI	MetaGPT
Infrastructure	✓	✗	✓	✗	✗
Conversation pattern	flexible	static	static	static	static
Execution-capable	✓	✗	✗	✗	✓
Human involvement	chat/skip	✗	✗	✗	✗