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.