

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 (2023)

Microsoft Research · Penn State University · University of Washington · Xidian University

<https://arxiv.org/abs/2308.08155>

Jong-Hwan Ko (고종환)
Department of AI Convergence
Pukyong National University

One sentence

This paper solves the problem of building many different LLM applications on one reusable design by letting customizable agents (LLM, tool, human) work through programmed conversations: AutoGen.

Conversable agents

Every agent can receive, react, and reply to messages. Its brain can be an LLM, a tool, a human, or a mix.

Conversation programming

The workflow is the conversation. Auto-reply plus natural-language prompts and Python rules decide who talks next and when to stop.

Evidence

Six applications: math, retrieval Q&A, ALFWorld, supply-chain coding, group chat, and chess.

A single LLM call cannot cover complex tasks, and earlier agent systems were too narrow

Why now

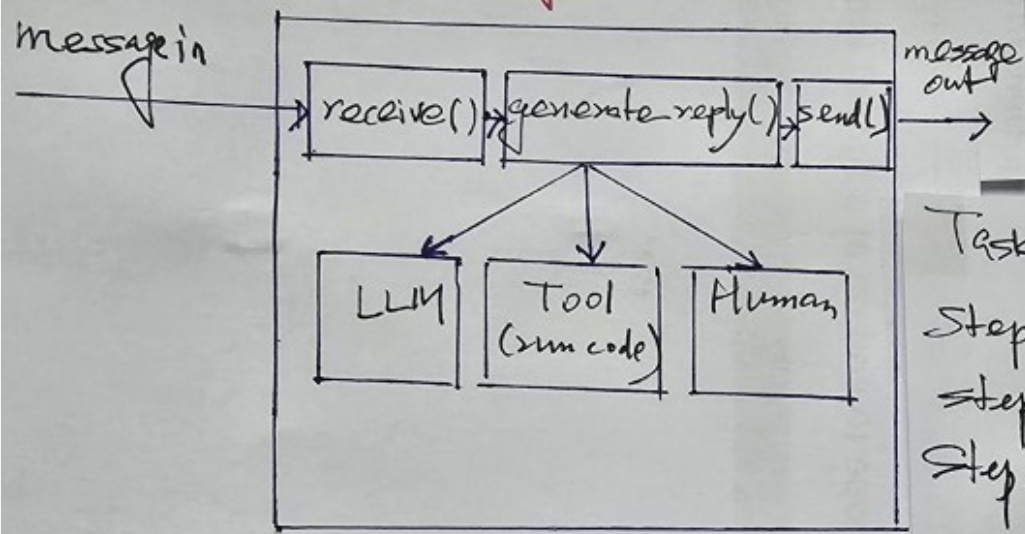
- Chat-optimized LLMs such as GPT-4 can take feedback, so agents can critique and correct each other.
- One LLM can play many roles depending on its prompt, so agents can be combined like modules.
- Hard tasks become easier when split into subtasks, and conversation is a natural way to split and merge.

Earlier system	What it could not do
AutoGPT, LangChain Agents, Transformers Agent	Single-agent only: no agent-to-agent collaboration
BabyAGI	Static, predefined order of agent messages
CAMEL	Role-play chat without code or tool execution

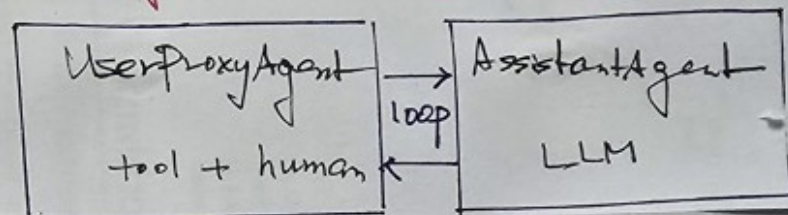
Two design questions the paper must answer: (1) How to build agents that are capable, reusable, and customizable? (2) How to give one unified interface for many conversation patterns?

SLIDE 4: Core Mechanism: Agents reply to each other automatically.

A. Inside one agent



B. Two-agent auto-reply loop



Task: "Plot META vs TESLA stock price changes."

Step 1: Task goes from UserProxyAgent to AssistantAgent.

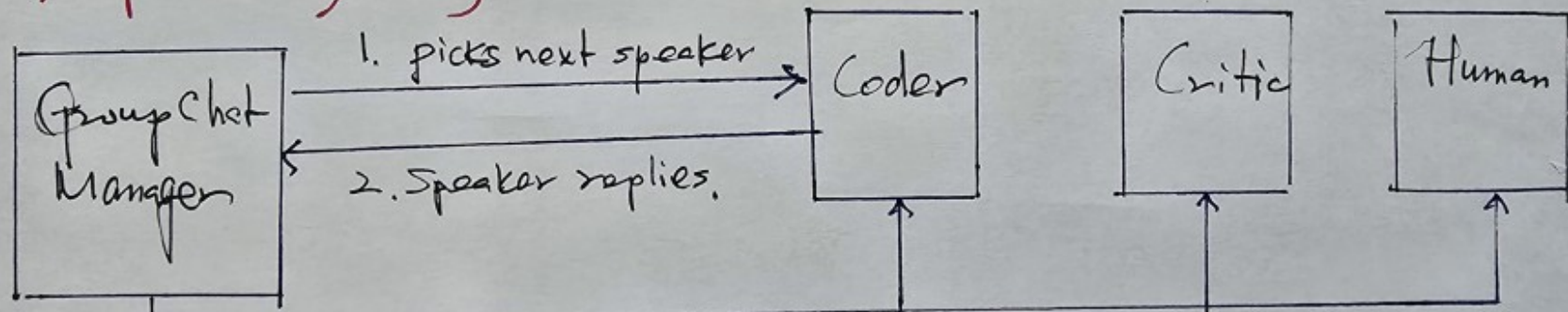
Step 2: AssistantAgent writes code and sends it to UPA.

Step 3: UserProxyAgent runs it and replies with an error or a result.

Step 4: AssistantAgent fixes code and resends it to UPA.

Loop repeats until the reply is "TERMINATE" or max. number of auto-replies is reached.

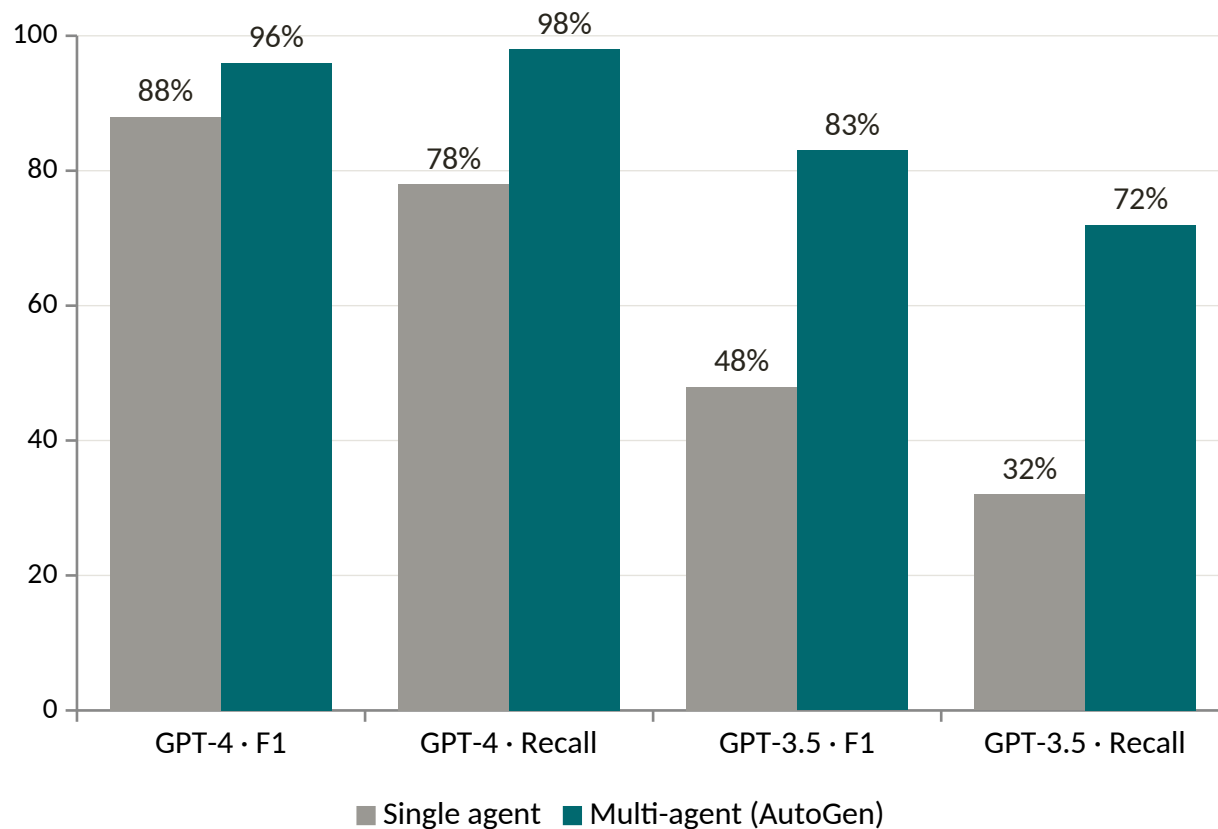
C. Group Chat (dynamic)



3. Manager broadcasts the reply to everyone,

Key result: splitting writer and safeguard into two agents makes unsafe-code detection far more reliable

Performance on OptiGuide: detecting unsafe code (%)



What it proves

- Role separation itself helps, not only a stronger model: F1 +8 points with GPT-4.
- A weaker model benefits most: F1 +35 points with GPT-3.5 (48% to 83%).
- This is the cleanest test of the paper's central claim because only the agent split changes.

What it does not prove

- 100 tasks built by the authors, half safe and half unsafe.
- Extra agents mean extra LLM calls; cost and latency are not compared here.

Weakest assumption: an LLM can be trusted to steer the conversation

The assumption

Control flow can be handed to the LLMs themselves: replying "TERMINATE" to stop, and choosing the next speaker in group chat with a role-play prompt.

Evidence gap

Speaker selection was checked on only 12 hand-crafted tasks. Most benchmarks were run by the authors, and the math comparison mixes different systems instead of isolating one factor.

Where it breaks

A Safeguard that misses one unsafe script still lets the executor run it (96% F1 is not 100%). Two agents that never say "TERMINATE" loop until the reply limit.

Fatal in practice? Not for supervised prototypes (human input on, reply limit set). Risky for fully autonomous runs, which the authors also warn about.

Connection: from research prototype to production frameworks

Follow-up work

AutoGen v0.4 (Jan 2025): a full redesign with an asynchronous, event-driven core, layered Core / AgentChat / Extensions, and built-in tracing.

Microsoft Agent Framework (Oct 2025, public preview): merges AutoGen with Semantic Kernel.

Same idea, other names

The paper's own related work: multi-agent debate for factuality and reasoning, and CAMEL for role play. AutoGen adds code execution and human turns.

My own use in research

Writer, reviewer, and executor agents that give feedback on my code, with the teacher as a human-in-the-loop agent (human input always on). The paper itself lists AI tutoring as a target use.

Take-away: an agent team plus a dedicated checker is a simple, powerful way to make LLM applications more reliable.