AutoGen Studio: A No-Code Developer Tool for Building and Debugging Multi-Agent Systems

Victor Dibia, Jingya Chen, Gagan Bansal, Suff Syed,
Adam Fourney, Erkang Zhu, Chi Wang, Saleema Amershi
Microsoft Research, Redmond, United States
{victordibia, jingyachen, gaganbansal, suffsyed, adam.fourney,
erkang.zhu, chiw, samershi}@microsoft.com

Abstract

Multi-agent systems, where multiple agents (generative AI models + tools) collaborate, are emerging as an effective pattern for solving long-running, complex tasks in numerous domains. However, specifying their parameters (such as models, tools, and orchestration mechanisms etc,.) and debugging them remains challenging for most developers. To address this challenge, we present AutoGen Studio, a no-code developer tool for rapidly prototyping, debugging, and evaluating multi-agent workflows built upon the AutoGen framework. AutoGen Studio offers a web interface and a Python API for representing LLM-enabled agents using a declarative (JSON-based) specification. It provides an intuitive drag-and-drop UI for agent workflow specification, interactive evaluation and debugging of workflows, and a gallery of reusable agent components. We highlight four design principles for no-code multi-agent developer tools and contribute an open-source implementation.[1]

1 Introduction

When combined with the ability to act (e.g., using tools), Generative AI models function as agents, enabling complex problem-solving capabilities. Importantly, recent research has shown that transitioning from prescribed (fixed) agent pipelines to a multi-agent setup with autonomous capabilities can result in desirable behaviors such as improved factuality and reasoning (Du et al., 2023), as well as divergent thinking (Liang et al., 2023). These observations have driven the development of application frameworks such as AutoGen (Wu et al., 2023), CAMEL (Li et al., 2024), and TaskWeaver (Qiao et al., 2023), which simplify the process of crafting multi-agent applications expressed as Python code. However, while multi-agent applications advance

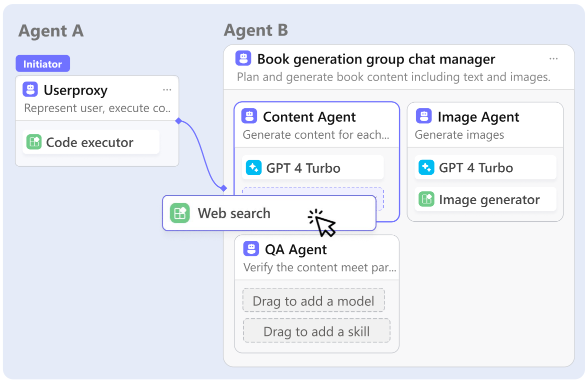

Agent A
Agent B
Initiator
Userproxy
Represent user, execute co..
Code executor
Book generation group chat manager
Plan and generate book content including text and images.
Content Agent
Generate content for each...
GPT 4 Turbo
Image Agent
Generate images
GPT 4 Turbo
Image generator
Web search
QA Agent
Verify the content meet par...
Drag to add a model
Drag to add a skill

Figure 1: AutoGen Studio provides a drag-n-drop UI where models, skills/tools, memory components can be defined, *attached* to agents and agents *attached* to workflows.

our capacity to solve complex problems, they also introduce new challenges. For example, developers must now configure a large number of parameters for these systems including defining agents (e.g., the model to use, prompts, tools or skills available to the agent, number of action steps an agent can take, task termination conditions etc.), communication and orchestration mechanisms - i.e., the order or sequence in which agents act as they collaborate on a task. Additionally, developers need to debug and make sense of complex agent interactions to extract signals for system improvement. All of these factors can create significant barriers to entry and make the multi-agent design process tedious and error-prone. To address these challenges, we have developed AutoGen Studio, a tool for rapidly prototyping, debugging, and evaluating MULTI-AGENT workflows. Our contributions are highlighted as follows:

- AutoGen Studio - a developer-focused tool (UI and backend Web and Python API) for declaratively specifying and debugging (human-in-the-loop and non-interactive) MULTI-AGENT workflows. AutoGen Studio provides a novel

[1]https://github.com/microsoft/autogen/tree/autogenstudio/samples/apps/autogen-studio

drag-and-drop experience (Figure 1) for rapidly authoring complex MULTI-AGENT agent workflows, tools for profiling/debugging agent sessions, and a gallery of reusable/shareable MULTI-AGENT components.

- We introduce profiling capabilities with visualizations of messages/actions by agents and metrics (costs, tool invocations, and tool output status) for debugging MULTI-AGENT workflows.
- Based on our experience building and supporting AUTOGEN STUDIO as an open-source tool with a significant user base (over 200K downloads within a 5-month period), we outline emerging design patterns for MULTI-AGENT developer tooling and future research directions.

To the best of our knowledge, AUTOGEN STUDIO is the first open-source project to explore a no-code interface for autonomous MULTI-AGENT application development, providing a suitable platform for research and practice in MULTI-AGENT developer tooling.

2 Related Work

2.1 Agents (LLMs + Tools)

Generative AI models face limitations, including hallucination — generating content not grounded in fact — and limited performance on reasoning tasks or novel out-of-distribution problems. To address these issues, practice has shifted towards agentic implementations where models are given access to tools to act and augment their performance (Mialon et al., 2023). Agentic implementations, such as React (Yao et al., 2022), explore a Reason and Act paradigm that uses LLMs to generate both reasoning traces and task-specific actions in an interleaved manner. As part of this process, developers have explored frameworks that build prescriptive pipelines interleaving models and tools (e.g., LIDA (Dibia, 2023), LangChain (Chase, 2022)). However, as tasks become more complex, requiring lengthy context and the ability to independently adapt to dynamic problem spaces, predefined pipelines demonstrate limited performance (Liu et al., 2024). This limitation has led to the exploration of more flexible and adaptive agent architectures.

2.2 MULTI-AGENT Frameworks

Several frameworks have been proposed to provide abstractions for creating such applications. AutoGen (Wu et al., 2023) is an open-source extensible framework that allows developers to build large MULTI-AGENT applications. CAMEL (Li et al., 2024) is designed to facilitate autonomous cooperation among communicative agents through role-playing, using inception prompting to guide chat agents toward task completion while aligning with human intentions. OS-Copilot (Wu et al., 2024) introduces a framework for building generalist agents capable of interfacing with comprehensive elements in an operating system, including the web, code terminals, files, multimedia, and various third-party applications. It explores the use of a dedicated planner module, a configurator, and an executor, as well as the concept of tools (Python functions or calls to API endpoints) or skills (tools that can be learned and reused on the fly).

Multi-Agent Core Concepts

1. **Model**: Generative AI model used to drive core agent behaviors.
2. **Skills/Tools**: Code or APIs used to address specific tasks.
3. **Memory**: Short term (e.g., lists) or long term (vector databases) used for to save and recall information.
4. **Agent**: A configuration that ties together the model, skills, memory components and behaviors.
5. **Workflow**: A configuration of a set of agents and how they interact to address tasks (e.g., order or sequence in which agents act, task planning, termination conditions etc.).

Collectively, these tools support a set of core capabilities - definition of ***agent*** parameters - such as generative AI ***models***, ***skills / tools*** or memory, and agent ***workflows*** - specifications of how these agents can collaborate. However, most of these frameworks primarily support a code-first representation of agent workflows, which presents a high barrier to entry and rapid prototyping. They also do not provide tools or metrics for agent debugging and evaluation. Additionally, they lack structured reusable templates to bootstrap or accelerate the agent workflow creation process. AUTOGEN STUDIO addresses these limitations by providing a visual interface to declaratively define and visualize agent workflows, test and evaluate these workflows, and offer templates for common MULTI-AGENT tasks to streamline development. While this work is built on the AUTOGEN open source library (Wu et al., 2023) and inherits the core abstractions for representing agents, the proposed design patterns on no-code developer tools are intended to apply to all MULTI-AGENT frameworks.

3 Design Goals

AUTOGEN STUDIO is designed to enhance the MULTI-AGENT developer experience by focusing on three core objectives:

Rapid Prototyping: Provide a *playground* where developers can quickly specify agent configurations and compose these agents into effective multi-agent workflows.

Developer Tooling: Offer *tools* designed to help developers understand and debug agent behaviors, facilitating the improvement of multi-agent systems.

Reusable Templates: Present a gallery of reusable, shareable templates to bootstrap agent workflow creation. This approach aims to establish shared standards and best practices for MULTI-AGENT system development, promoting wider adoption and implementation of MULTI-AGENT solutions.

4 System Design

AUTOGEN STUDIO is implemented across two high-level components: a frontend user interface (UI) and a backend API (web, python and command line). It can be installed via the PyPI package manager (listing 1).

```
pip install autogenstudio
autogenstudio ui --port 8081
```

listing 1: AUTOGEN STUDIO can be installed from PyPI (pip) and the UI launched from the command line.

4.1 User Interface

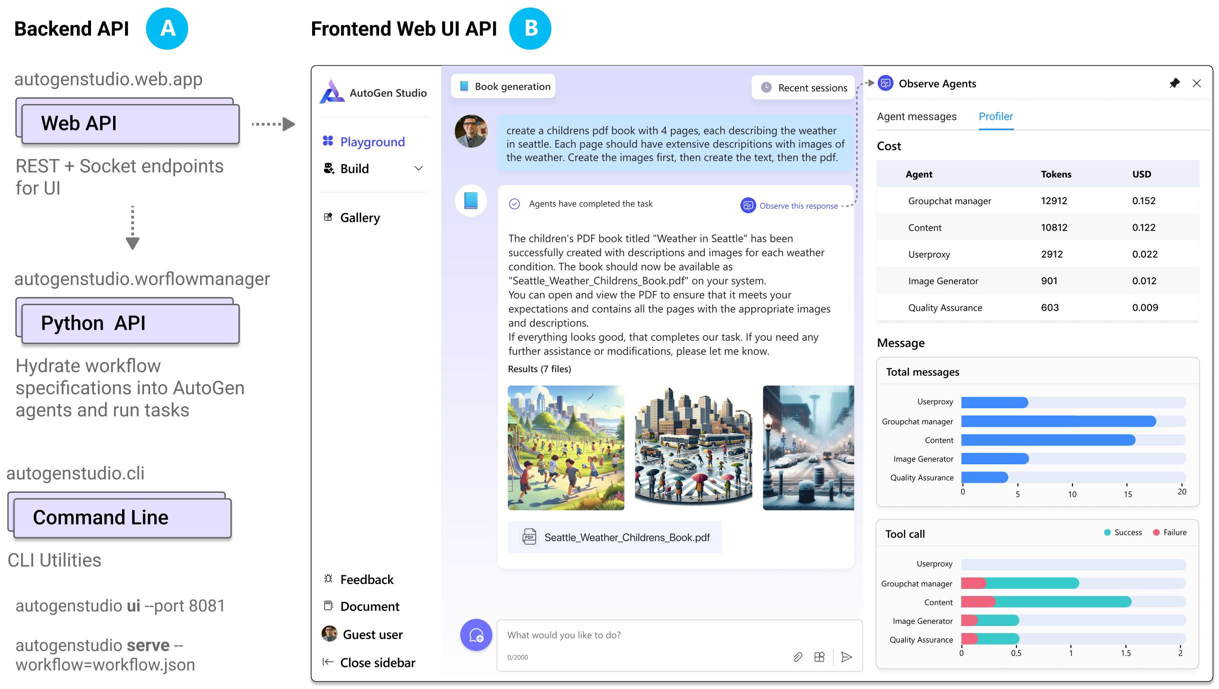
The frontend web interface in AUTOGEN STUDIO is built using React and implements three main views that support several key functionalities. The *build view* enables users to author (define-and-compose) multi-agent workflows. The *playground view* allows for interactive task execution and workflow debugging, with options to export and deploy. The *gallery view* facilitates the reuse and sharing of agent artifact templates.

4.1.1 Building Workflows

The *build* view in the UI (see Figure 1) offers a *define-and-compose* experience, allowing developers to declaratively define low-level components and iteratively compose them into a workflow. For instance, users can define configurations for models, skills/tools (represented as Python functions addressing specific tasks), or memory stores (e.g., documents organized in a vector database). Each entity is saved in a database for use across interface interactions. Subsequently, they can define an agent, attaching models, skills, and memory to it. Several agent default templates are provided following AUTOGEN abstractions - a *UserProxy* agent (has a code execution tool by default), an *AssistantAgent* (has a generative AI model default), and a *GroupChat* agent (an abstraction container for defining a list of agents, and how they interact). Finally, workflows can be defined, with existing agents attached to these workflows. The default workflow patterns supported are autonomous chat (agents exchange messages and actions across conversation turns until a termination condition is met) and sequential chat (a sequence of agents defined, each agent processes its input in order and passes on a summary of their output to the next agent). The workflow composition process is further enhanced by supporting a drag-and-drop interaction e.g., skills/models can be dragged to agents and agents into workflows.

4.1.2 Testing and Debugging Workflows

Workflows can be tested in-situ in the *build* view, or more systematically explored within the *playground* view. The playground view allows users create *sessions*, attach workflows to the session, and run tasks (single shot or multi-turn). Sessions can be shared (to illustrate workflow performance) and multiple sessions can be compared. AUTOGEN STUDIO provides two features to support debugging. First, it provides an observe view where as tasks progress, messages and actions performed by agents are streamed to the interface, and all generated artifacts are displayed (e.g., files such as images, code, documents etc). Second a post-hoc profiler view is provided where a set of metrics are visualized for each task addressed by a workflow - total number of messages exchanged, costs (generative AI model *tokens* consumed and dollar costs),



```
autogenstudio ui --port 8081

autogenstudio serve --workflow=workflow.json
```

create a childrens pdf book with 4 pages, each describing the weather in seattle. Each page should have extensive descripitions with images of the weather. Create the images first, then create the text, then the pdf.

The children's PDF book titled "Weather in Seattle" has been successfully created with descriptions and images for each weather condition. The book should now be available as "Seattle_Weather_Childrens_Book.pdf" on your system.
You can open and view the PDF to ensure that it meets your expectations and contains all the pages with the appropriate images and descriptions.
If everything looks good, that completes our task. If you need any further assistance or modifications, please let me know.

Agent	Tokens	USD
Groupchat manager	12912	0.152
Content	10812	0.122
Userproxy	2912	0.022
Image Generator	901	0.012
Quality Assurance	603	0.009

Figure 2: AUTOGEN STUDIO provides a backend api (web, python, cli) and a UI which implements a *playground* (shown), *build* and *gallery* view. In the playground view, users can run tasks in a session based on a workflow. Users can also *observe* actions taken by agents, reviewing agent messages and metrics based on a profiler module.

how often agents use tools and the *status* of tool use (success or failure), for each agent.

4.1.3 Deploying Workflows

AUTOGEN STUDIO enables users to export workflows as a JSON configuration file. An exported workflow can be seamlessly integrated into any Python application (listing 2), executed as an API endpoint using the AUTOGEN STUDIO command line interface (figure 2a), or wrapped in a Docker container for large-scale deployment on various platforms (Azure, GCP, Amazon, etc.).

```python
from autogenstudio import
    WorkflowManager
wm = WorkflowManager("workflow.
    json")
wm.run(message="What is the
    height of the Eiffel Tower")
```

listing 2: Workflows can be imported in python apps.

4.1.4 Template Gallery

The UI also features a *gallery* view - a repository of components (skills, models, agents, workflows) that users can import, extend, and reuse in their own workflows. Since each component specification is declarative (JSON), users can also easily export, version and reshare them.

4.2 Backend API - Web, Python, and Command Line

The backend API comprises three main components: a web API, a Python API, and a command-line interface. The web API consists of REST endpoints built using the FastAPI library[2], supporting HTTP GET, POST, and DELETE methods. These endpoints interact with several key classes: A $DBManager$ performs CRUD (Create, Read, Update, Delete) operations on various entities such as skills, models, agents, memory, workflows, and sessions. The $WorkflowManager$ class handles the ingestion of declarative agent workflows, converts them into AUTOGEN agent objects, and executes tasks (see listing 2). A $Profiler$ class parses agent messages to compute metrics. When a user initiates a task within a session, the system retrieves the session history, instantiates agents based on their serialized representations from the database, executes the task, streams intermediate messages to the UI via websocket, and returns the final results. AUTOGEN STUDIO also provides a command-line interface with utilities for launching the bundled UI and running exported workflows as API endpoints.

[2]FastAPI: https://fastapi.tiangolo.com/

5 Usage and Evaluation

In this project, we have adopted an in-situ, iterative evaluation approach. Since its release on GitHub (5 months), the AutoGen Studio package has been installed over 200K times and has been iteratively improved based on feedback from usage (> 135 GitHub issues). Issues highlighted several user pain points that were subsequently addressed including: (a) challenges in defining, persisting, and reusing components, resolved by implementing a database layer; (b) difficulties in authoring components, resolved by supporting automated tool generation from descriptions and integrating an IDE for editing tools; (c) frustrations caused by components failing during end-to-end tests, addressed by incorporating a test button for components (e.g.,models) and workflows in the *build* view. Figure 3 displays a plot of all AutoGen Studio issues. Each point represents an issue, based on an embedding of its text (title + body) using OpenAI's text-embedding-3-large model. The embeddings were reduced to two dimensions using UMAP, clustered with K-Means ($k = 8$), and cluster labels generated using GPT-4 (grounded on 10 samples from its centroid). Finally, in Appendix A, we demonstrate how AutoGen Studio can effectively be used to support an engineer persona in rapidly prototyping, testing, and iteratively debugging a multi-agent workflow, and deploying it as an API endpoint to address a concrete task (generating books).

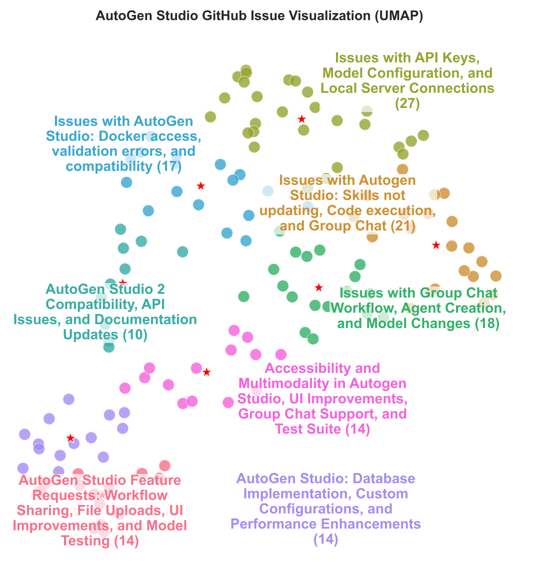


Figure 3: Plot of GitHub issues ($n = 8$ clusters) from the AutoGen Studio repo. User feedback ranged from support with workflow authoring tools (e.g., the ability configure and test models) to general installation.

6 Emerging Design Patterns and Research Directions

In the following section, we outline some of the high-level emerging patterns which we hope can help inform the design of no-code interfaces for building next-generation multi-agent applications.

6.1 Define-and-Compose Workflows

Allow users to author workflows by defining components and composing them (via drag-and-drop actions) into multi-agent workflows.

A multi-agent system can have a wide array of parameters to configure. We have found that selecting the right visual presentation of the workflow to helping users understand what parameters to configure (discovery), and how to configure them. Specifically, we have found that a define-and-compose workflow, where entities are first defined and persisted independently, and then composed ultimately into multi-agent workflows, provides a good developer experience. This includes providing tools to support authoring entities e.g., the ability define and test models, an IDE for generating/editing tools (code), and a a canvas-based visual layout of workflows with drag-and-drop interaction for associating entities in the workflow.

6.2 Debugging and Sensemaking Tools

Provide robust tools to help users debug, interpret, and rationalize the behavior and outputs of multi-agent systems.

Multi-agent workflows can be brittle and fail for multiple reasons, ranging from improperly configured models to poor instructions for agents, improper tool configuration for agents or termination conditions. A critical request has been for tools to help users debug and make sense of agent responses.

6.3 Export and Deployment

Enable seamless export and deployment of multi-agent workflows to various platforms and environments.

While a no-code tool like AutoGen Studio enables rapid iteration and demonstration of workflows, the natural progression for most use cases is that developers want to replicate the same outcomes but integrated as parts of their core applications. This stage requires seamless export and deployment of multi-agent workflows to various platforms and environments.

6.4 Collaboration and Sharing

Facilitate user collaboration on multi-agent workflow development and allow easy sharing of creations within the community.

Collaboration and sharing are key to accelerating innovation and improving multi-agent systems. By enabling users to collaborate on workflow development, share their creations, and build upon each other's work, a more dynamic and innovative development environment can be cultivated. Tools and features that support real-time collaboration, version control, and seamless sharing of workflows and components are essential to foster a community-driven approach. Additionally, offering a repository or gallery where users can publish and share their workflows, skills, and agents promotes communal learning and innovation.

7 Future Research Directions

While we have explored early implementations of the design requirements mentioned above, our efforts in building AUTOGEN STUDIO have also identified two important future research areas and associated research questions.

- **Offline Evaluation Tools**: This encompasses questions such as how can we measure the performance, reliability, and reusability of agents across tasks? How can we better understand their strengths and limitations? How can we explore alternative scenarios and outcomes? And how can we compare different agent architectures and collaboration protocols?

- **Understanding and quantifying the impact of multi-agent system design decisions**: These questions include determining the optimal number and composition of agents for a given problem, the best way to distribute responsibilities and coordinate actions among agents, and the trade-offs between centralized and decentralized control or between homogeneous and heterogeneous agents.

- **Optimizing of multi-agent systems**: Research directions here include the dynamic generation of agents based on task requirements and available resources, tuning workflow configurations to achieve the best performance, and adapting agent teams to changing environments and user preferences. Furthermore, how can we leverage human oversight and feedback to improve agent reliability, task performance and safety?

8 Conclusion

This paper introduced AUTOGEN STUDIO, a no-code developer tool for rapidly prototyping, debugging, and evaluating multi-agent workflows. Key features include a drag-and-drop interface for agent workflow composition, interactive debugging capabilities, and a gallery of reusable agent components. Through widespread adoption, we identified emerging design patterns for multi-agent developer tooling - a define and compose approach to authoring workflows, debugging tools to make sense of agent behaviors, tools to enable deployment and collaborative sharing features. AUTOGEN STUDIO lowers the barrier to entry for multi-agent application development, potentially accelerating innovation in the field. Finally we outline future research directions including developing offline evaluation tools, ablation studies to quantify the impact of MULTI-AGENT systems design decisions and methods for optimizing multi-agent systems.

9 Ethics Statement

AUTOGEN STUDIO is designed to provide a no-code environment for rapidly prototyping and testing multi-agent workflows. Our goal is to responsibly advance research and practice in solving problems with multiple agents and to develop tools that contribute to human well-being. Along with AUTOGEN, AUTOGEN STUDIO is committed to implementing features that promote safe and reliable outcomes. For example, AUTOGEN STUDIO offers profiling tools to make sense of agent actions and safeguards, such as support for Docker environments for code execution. This feature helps ensure that agents operate within controlled and secure environments, reducing the risk of unintended or harmful actions. For more information on our approach to responsible AI in AutoGen, please refer to transparency FAQS here. Finally, AUTOGEN

STUDIO is not production ready i.e., it does not focus on implementing authentication and other security measures that are required for production ready deployments.

Acknowledgements

We would like to thank members of the open-source software (OSS) community and the AI Frontiers organization at Microsoft Research for discussions and feedback along the way. Specifically, we would like to thank Piali Choudhury, Ahmed Awadallah, Robin Moeur, Jack Gerrits, Robert Barber, Grace Proebsting, Michel Pahud, Qingyun Wu, Harsha Nori and others for feedback and comments.

References

Harrison Chase. 2022. LangChain. *Github*.

Victor Dibia. 2023. Lida: A tool for automatic generation of grammar-agnostic visualizations and infographics using large language models. *arXiv preprint arXiv:2303.02927*.

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

Guohao Li, Hasan Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. 2024. Camel: Communicative agents for" mind" exploration of large language model society. *Advances in Neural Information Processing Systems*, 36.

Tian Liang, Zhiwei He, Wenxiang Jiao, Xing Wang, Yan Wang, Rui Wang, Yujiu Yang, Zhaopeng Tu, and Shuming Shi. 2023. Encouraging divergent thinking in large language models through multi-agent debate. *arXiv preprint arXiv:2305.19118*.

Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2024. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173.

Grégoire Mialon, Roberto Dessì, Maria Lomeli, Christoforos Nalmpantis, Ram Pasunuru, Roberta Raileanu, Baptiste Rozière, Timo Schick, Jane Dwivedi-Yu, Asli Celikyilmaz, et al. 2023. Augmented language models: a survey. *arXiv preprint arXiv:2302.07842*.

Bo Qiao, Liqun Li, Xu Zhang, Shilin He, Yu Kang, Chaoyun Zhang, Fangkai Yang, Hang Dong, Jue Zhang, Lu Wang, et al. 2023. Taskweaver: A code-first agent framework. *arXiv preprint arXiv:2311.17541*.

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, and Chi Wang. 2023. Autogen: Enabling next-gen llm applications via multi-agent conversation framework. *arxiv*.

Zhiyong Wu, Chengcheng Han, Zichen Ding, Zhenmin Weng, Zhoumianze Liu, Shunyu Yao, Tao Yu, and Lingpeng Kong. 2024. Os-copilot: Towards generalist computer agents with self-improvement. *arXiv preprint arXiv:2402.07456*.

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

A Jack the Software Engineer Persona Use Case

Jack is a junior software engineer who has recently joined SoftwareCon. As part of his tasks, he is required to create an application that can generate a variety of short books. The initial version should focus on generating children's books (age 5 -8 years old) based on a given query (e.g., *create a book for kids on how the sun works*) with the expectation of being generalized to support other generic tasks. Jack has heard about a MULTI-AGENT approach to building systems that can address a variety of tasks through autonomous collaboration between agents. To explore this approach, he begins by perusing the AUTOGEN STUDIO documentation, installs it, launches the UI, and performs the following steps:

A.1 Step 1: Define and Compose a Workflow

Jack starts with the *Build* view, where he reviews the default skills that come with AUTOGEN STUDIO. He sees that there are two relevant skills $generate\_pdfs$ and $generate\_images$. He verifies that he has the appropriate API keys for the $generate\_image$ skill. Next, he creates a GPT3.5 model and adds an API key.

Following best practices, Jack knows that the basic agent team with AUTOGEN consists of a *UserProxyAgent* that can execute code and an *AssistantAgent* that can solve tasks as well as write code or call available tools/skills. He creates both of these agents; for his *AssistantAgent*, he ensures that he attaches the GPT4 model he created previously and also attaches both skills. Jack moves on to the workflow tab and creates a new autonomous chat workflow where he specifies the *UserProxyAgent* as the initiator and his *AssistantAgent* as the receiver.

A.2 Step 2: Test and Iterate

Within the workflow tab, Jack tests the workflow immediately and quickly observes a few issues. Using the profiler tool and visualization of messages exchanged by the agents, he notices that there seem to be quality issues with the content of the book - namely, the *AssistantAgent* seems to generate very short messages and hence the book pages contains only 2 sentences per page whereas the requirements state that the kids are slightly older and can read much longer text.

To remedy these issues, Jack takes two actions. First, he attempts to extend the base instructions of his *AssistantAgent*, but still doesn't get pages with more than 3 sentences across interactive tests. He recalls that using more agents can help separate focus and improve task performance. He then switches to creating 4 agents: a *UserProxy*, a *ContentAssistant* with detailed instructions on generating the content for each page, a *QualityAssuranceAssistant* to verify the pages meet parameters, and an *ImageGeneratorAssistant* focused on generating images for the book. He then creates a *GroupChat* agent and adds his list of agents to it. Next, he creates a new workflow where the receiver is the *GroupChat* agent and tests the application across a few tries. Jack is satisfied with the results as full-page stories are now generated correctly. In addition, Jack is concerned about costs but can easily use the *observe message* button to explore duration, tokens used by agents, tool/skill use and LLM dollar costs for each task run.

A.3 Step 3: Export and Share

At this point, Jack has two final tasks: he wants to share his work with colleagues for feedback and then provide an API they can prototype with. AUTOGEN STUDIO makes sharing easy; First, Jack can simply export and share a link to successful sessions. Second, he can also download his workflow and share it with colleagues, saving it in a version control system like Git. Third, he can spin up an API endpoint where the agents can respond to task requests using cli commands 'autogenstudio serve –port 8000'. He can also spin up a docker container using the AUTOGEN STUDIO serve command and scale it on any platform of his choice (Azure, AWS, GCP, Hugging Face).