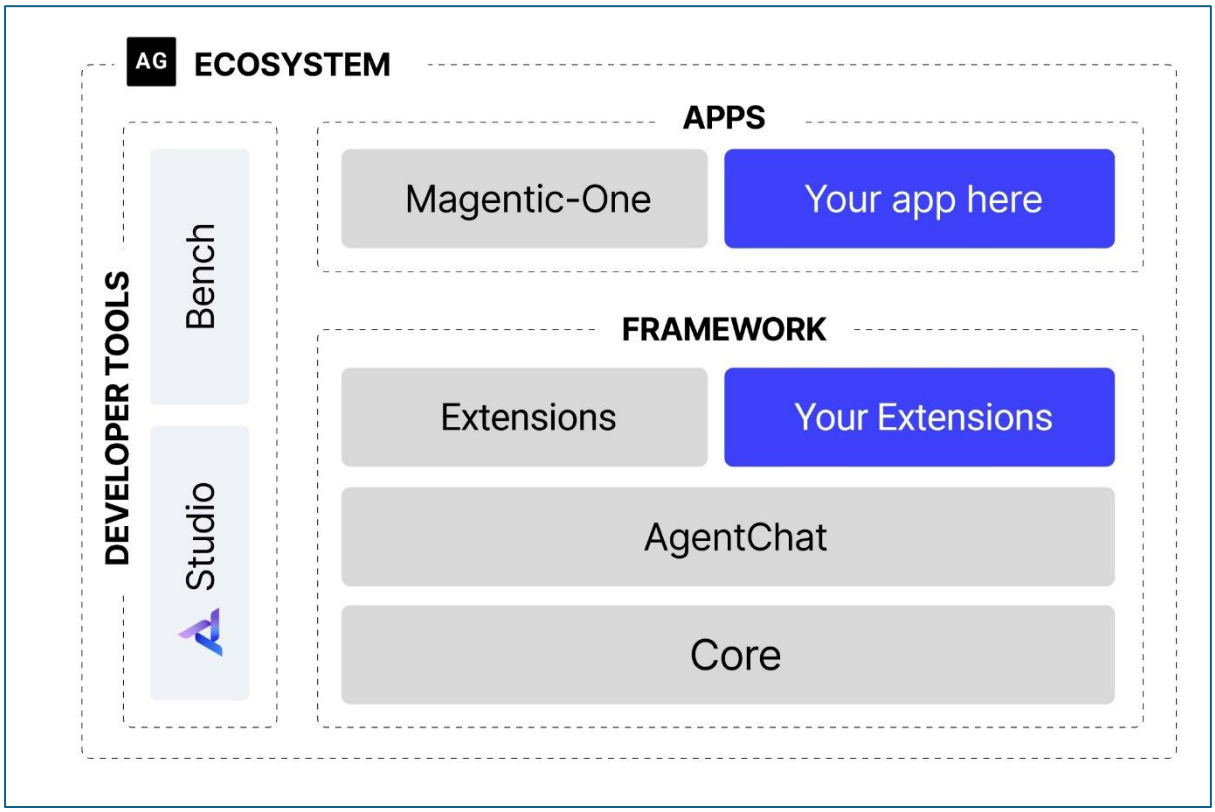
AutoGen Framework

AutoGen is a **programming framework for agentic AI**, meaning it helps developers create systems where multiple AI agents interact, reason, and collaborate to complete tasks. Think of it as a toolkit for orchestrating intelligent conversations between agents, humans, and tools.



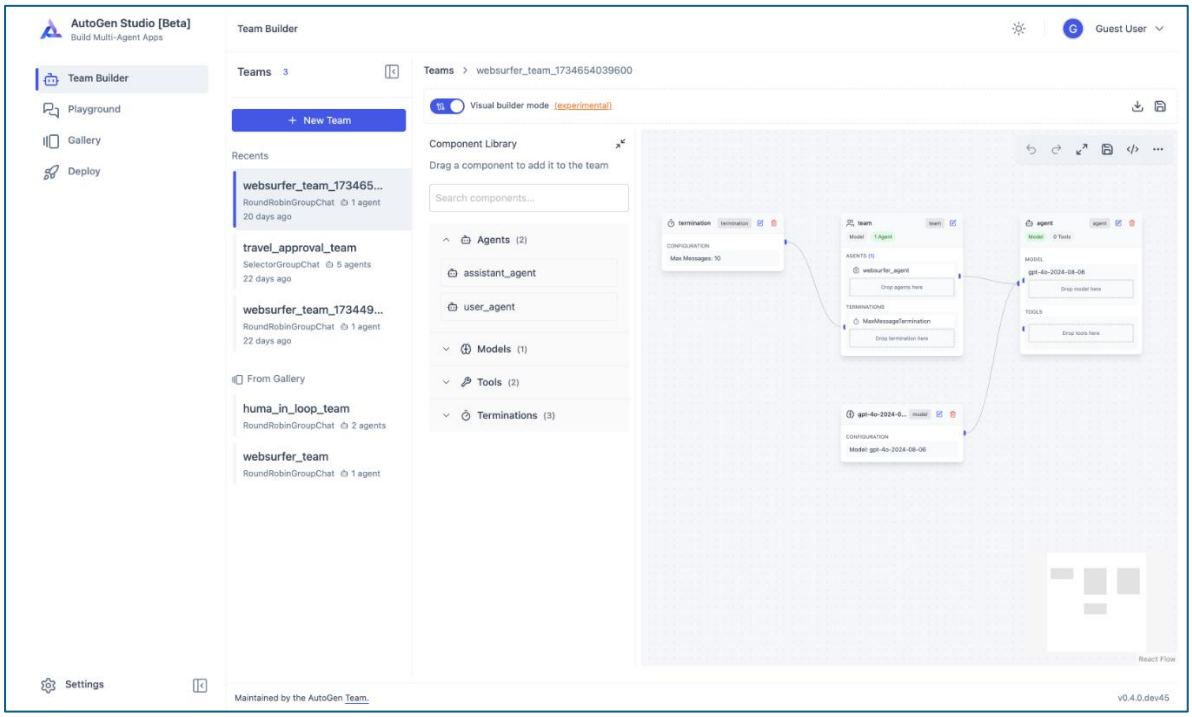
The AutoGen ecosystem provides everything you need to create AI agents, especially multi-agent workflows -- framework, developer tools, and applications.

The *framework* uses a layered and extensible design. Layers have clearly divided responsibilities and build on top of layers below. This design enables you to use the framework at different levels of abstraction, from high-level APIs to low-level components.

- Core API implements message passing, event-driven agents, and local and distributed runtime for flexibility and power. It also support cross-language support for .NET and Python.
- AgentChat API implements a simpler but opinionated API for rapid prototyping. This API is built on top of the Core API and is closest to what users of v0.2 are familiar with and supports common multi-agent patterns such as two-agent chat or group chats.

- Extensions API enables first- and third-party extensions continuously expanding framework capabilities. It support specific implementation of LLM clients (e.g., OpenAI, AzureOpenAI), and capabilities such as code execution.

The ecosystem also supports two essential *developer tools*:



- AutoGen Studio provides a no-code GUI for building multi-agent applications.
- AutoGen Bench provides a benchmarking suite for evaluating agent performance.

You can use the AutoGen framework and developer tools to create applications for your domain. For example, Magentic-One is a state-of-the-art multi-agent team built using AgentChat API and Extensions API that can handle a variety of tasks that require web browsing, code execution, and file handling.

Key Features:

- Multi-agent orchestration: Easily build systems where agents talk to each other to solve problems.
- Conversable agents: Agents can engage in natural language conversations with humans or other agents.
- Tool integration: Agents can use external tools (APIs, databases, etc.) to enhance capabilities.
- Autonomous and human-in-the-loop workflows: Supports both fully automated and semi-automated systems.

- Flexible conversation patterns: Enables complex workflows like brainstorming, planning, and decision-making.
- LLM optimization: Helps overcome limitations of LLMs by distributing tasks across specialized agents.

Use Case: Multi-Agent Code Debugging Assistant

Objective

To build an AI-powered debugging assistant where multiple agents.

Agents Involved

Agent Name	***Role***
UserProxyAgent	Represents the human user, initiates the debugging task
CodeReviewerAgent	Analyzes the code for bugs and suggests improvements
CodeExecutorAgent	Runs the code and returns output or errors
FixerAgent	Applies fixes based on feedback from the reviewer and executor

Workflow

1. **UserProxyAgent** receives a buggy Python script from the user.
2. It forwards the script to **CodeReviewerAgent**, which inspects the logic and identifies potential bugs.
3. The script is then passed to **CodeExecutorAgent**, which runs it and captures runtime errors or unexpected outputs.
4. Based on the review and execution results, **FixerAgent** proposes code changes.
5. The updated script is re-evaluated by the reviewer and executor until no issues remain.
6. The final, corrected script is returned to the user.

Tools Used

- **Python interpreter** (via subprocess or sandboxed execution)
- **LLMs** (e.g., GPT-4) for reasoning and code generation
- **AutoGen's GroupChat** to manage the conversation loop between agents

Benefits

- Automates the debugging process
- Reduces developer effort and time
- Encourages modular, explainable reasoning
- Supports human-in-the-loop for final validation