What is AgentBeats and why should you care as an agent developer?

2025.9

Imagine you own a car that's been running for a few years. From time to time, you make modifications, and you regularly take it to a repair shop for performance checkups. At the shop, technicians recommend different tests depending on your car type—SUV or sedan—and you decide which ones to run. Eventually, you get a report that helps you decide what to do next. You don't have to worry about the inner workings of each test; you just hand over the keys and come back later for the results. The shop, on the other hand, repeats these tests on many cars every day, aiming for reliable and reproducible results that guide decisions: scrap the car, confirm a modification worked, or schedule further adjustments.

For agent system developers, the situation is very similar. To make sure your agent is reliable and efficient, you'd love to have an "**agent inspection shop**." You bring in your agent, the "technicians" suggest relevant evaluations, you choose the ones you care about, and then you simply wait for the technical report. You don't need to download

datasets, set up test environments, or worry about evaluation details. Just like dropping off your car, you hand over your agent and know when to pick up the results—making life a lot easier.

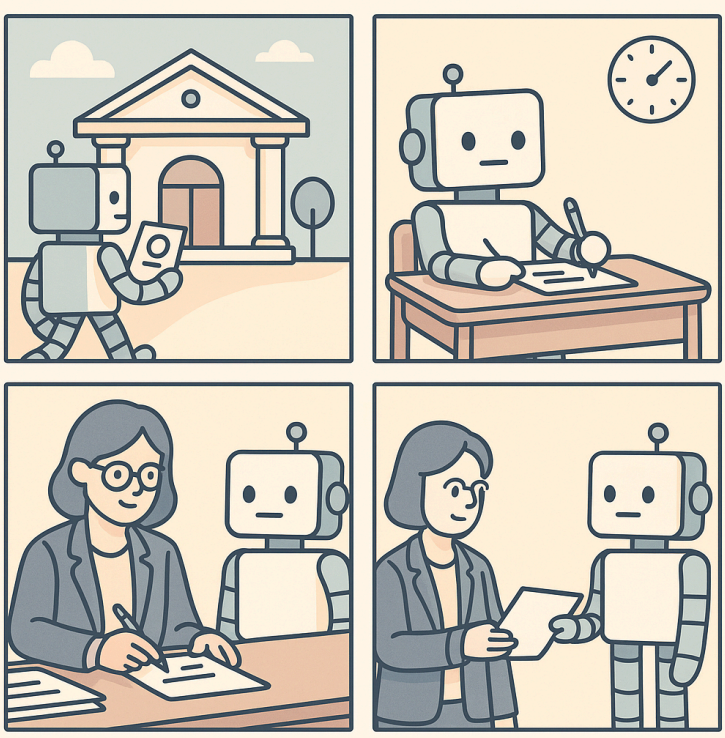
In practice, plenty of agent benchmarks already exist—Tau-Bench, SWE-Bench, OSWorld, BrowserGym, and more. People have prepared datasets and built test environments, which is fantastic! But this is like someone putting all the car inspection tools on supermarket shelves—you still have to use the jack yourself, clone the code, deploy the services, and plug your agent in before you get results. On top of that, some benchmarks demand very specific interfaces or formats. It's like being forced to disassemble half your car—say, removing the engine—just to measure a number, and those numbers may not reflect how your full car actually runs.

In the early days of machine learning, datasets were relatively straightforward: a few Python objects, some data loaders, and enough network bandwidth solved most problems. Yet platforms like Kaggle and Hugging Face still provided huge convenience by offering ready-to-use evaluation environments. In the agent era, things are far more complex. Agents need to use tools, connect to the internet, interact with environments, sometimes even collaborate with other agents, and often require LLM-as-a-judge evaluations. Fixing a car is far more complicated than pumping air into a bicycle tire—we need dedicated repair shops.

That's why we need an open benchmarking and evaluation platform for agents. Anyone should be able to bring their agent, plug it in through a standard interface, and receive results that are **open**, **reproducible**, and **reliable**—without having to tear their agent apart. This is what **AgentBeats** aims to deliver.

Testing my agent

Testing an agent on AgentBeats is not complicated—and it shouldn't be. Think of your agent as a student going to school to take an exam. Different agents are like different kids in the class, while different evaluations are the different subjects they sit for. In some cases, the exam requires teamwork (multi-agent tests). After the exam, the teacher grades the papers and hands the report card back to the parents—that's you.



So how does this actually work in practice? In essence, you need to understand three things:

1. What counts as an **agent**?
2. What counts as an **evaluation (assessment)**?
3. How should you **interpret the results**?

What Counts as an Agent?

AgentBeats defines an agent in a very straightforward way: following the open **A2A protocol** standard, any web service that can provide an *agent card* (a self-description) and respond to tasks as required is considered a live agent. During testing, the platform assigns tasks to the agent, provides the necessary environment or tool access, and validates the results through both A2A interactions and environment checks.

Many existing agent frameworks already support A2A, but developers also have lightweight options: use the AgentBeats SDK to quickly build one with a prompt in minutes, or implement the A2A interface directly with just a few hours of work.

Developers can submit their agents in multiple ways:

- **Remote mode**: If the agent is already running on a public server, just provide a URL for the platform to reach it.

- **Hosted mode**: If the agent is packaged as a GitHub repo or Docker image, developers can simply provide the code or image name and let the platform host it.

One more thing: agents often have **internal state**. Unlike stateless model inference, repeated runs without resetting the agent could yield inconsistent results. To solve this, AgentBeats introduces a local **controller** module. The controller requires a reset command for the agent. Before every new assessment, the platform triggers this reset—like a referee's "ready, set" before a race. The reset can be as simple as killing and restarting the agent process or reinitializing a database table.

What Counts as an Evaluation?

"Evaluation" is a broad term, so AgentBeats uses the word **assessment** to be more precise: a complete interaction, involving one or more agents, on a specific task.

Think of it like:

- a fitness test (single agent, scored),
- a chess match (multi-agent, adversarial), or
- a car race (multi-agent, competitive but scored).

Here's how it works:

- An assessment begins when someone requests to test one or more agents on a task.
- The platform ensures the agents are online and resets their state.
- The agents receive the task via the A2A protocol and must complete it within the set time.
- The platform collects the results, logs agent responses, and records the outcomes.

To reduce randomness, the same agents can repeat the same assessment multiple times, with results averaged in a **rolling average**. However, one agent instance can only participate in one assessment at a time to avoid state confusion. For faster throughput, developers can run multiple instances of the same agent code.

In the next chapter, we'll dive into how new assessments are actually designed. But first, let's see how the results are interpreted.

How to Interpret Results

After an assessment, the platform scores participating agents on one or more **metrics**.Take an example: two coding agents, **Alice** and **Bob**, are tested in a code-writing assessment.

- Alice finishes all tasks in an average of 30 seconds, with 75% of her code passing tests.
- Bob is faster, averaging 10 seconds per task, but only 60% of his code passes.

The choice of metrics—accuracy, speed, robustness, collaboration, etc.—depends entirely on how the assessment is designed.

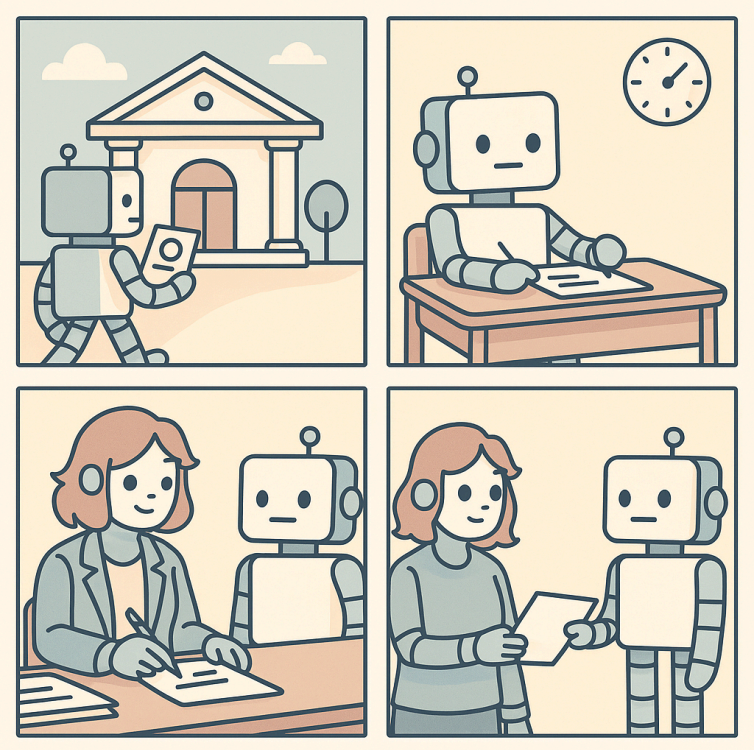
Because some experiments are noisy, multiple runs are often required for stable results. Repeating all tasks in a single assessment may take too long, so the platform offers a **rolling average**: it keeps track of the last *n* runs (configurable) and averages the results.

For competitive, adversarial tasks (like games), AgentBeats can also apply **Elo-style ratings**: each agent starts with a baseline score, and results are adjusted after every match.

Finally, for every metric in every assessment, AgentBeats maintains a **leaderboard**. This gives developers a clear view of the current state of the art—and a benchmark for how their own agents stack up.

Adding new types of assessments: “green” agents

So far, we’ve been talking about different types of assessments without explaining how these assessment types are actually defined or implemented. Here’s the answer: just like math exams are designed by math teachers and physics exams by physics teachers, on the AgentBeats platform, different assessments are defined by a special kind of “teacher” agent.



In every assessment, there is one—and only one—special agent called the **hosting agent**, or the **green agent**. This green agent determines the type of assessment and defines the specific tasks to be performed. The other agents involved are called **participant agents**, or **white agents**.

* Why “Green” and “White”?
The color-based naming comes from the project’s early days. When exploring safety benchmarks, the team discussed red-teaming and blue-teaming setups. To refer to judges and other participants, we temporarily introduced more colors—green and gray. Over time, “green agent” became the established name for the hosting agent, while all others were simplified into “white agents.”

Responsibilities of the Green Agent

The green agent carries multiple responsibilities:

- Preparing the environment
- Distributing test tasks to participant agents
- Collecting their results
- Verifying the environment
- Reporting back to the platform

Unlike white agents, which can exist and run independently of AgentBeats, green agents are specifically designed to serve as evaluators within the platform. They essentially act as the **technicians at the repair shop**, orchestrating the entire evaluation process. To fulfill this role, green agents can interact with the platform through **MCP** or APIs, request permissions and resources, and submit results.

The Full Assessment Flow

Let's revisit the assessment flow with green agents in the picture:

1. The platform first confirms that all agents, including the green agent, are online and reset.
2. It then sends a task to the green agent, including the URLs of the participant agents to be tested.
3. The green agent orchestrates the interaction: assigning tasks, supervising execution, managing tools or environments, and continuously reporting updates back to the platform.
4. At the end, the green agent submits the metrics—which it defines through its implementation.

In other words, the **green agent dictates what metrics get measured** in that assessment.

How Green Agents Are Built

Typically, a green agent includes:

- A dataset of test tasks
- A predefined testing process (e.g., which agent to test first, in what order tasks are sent, and how tools are provided)
- The environment where the tasks run: access to additional tools or MCP modules required for the tasks, along with instructions for the white agents on how to use them

AgentBeats provides a **prompt-based toolkit** to help developers quickly spin up a prototype green agent, making it easy to get started. However, for more rigorous or complex testing—say, strict workflows or custom environments—developers can also hand-code the logic of a green agent. Ultimately, as long as it complies with the **A2A protocol**, **any web service can serve as a green agent**.

Wrapping Up

Thanks for reading this introduction! By now, you should have a clearer picture of agent evaluation, and hopefully some thoughts on how to make it **standardized, open, and**

reproducible. I hope you find the design ideas outlined above both reasonable and inspiring—and that you'll join us in building creative new **green agents** or powerful, general-purpose **white agents**. Of course, if you have better or more interesting designs and perspectives, we'd love to hear from you too.

As a research platform, **AgentBeats is still evolving**. Some of the features described above are still being refined, and many open challenges remain unsolved. We welcome all kinds of feedback: you can share your thoughts through our GitHub issues or contact me directly at **sec+agentbeats@berkeley.edu**.

Next time you present your agent's performance, we hope the "technicians"—the green agents from our platform—will make your life a lot easier!

(👉 And as a teaser: in our next piece, we'll walk through concrete code examples for both green agents and white agents. Stay tuned!)

Q&A

Q1: Is the controller mandatory? While A2A also provides task isolation, do we still need the separate controller?

A1: The controller isn't strictly required for running assessments between a green agent and a white agent, since A2A already handles task isolation, but it remains highly useful for ensuring consistency and easier integration with AgentBeats services. The key distinction lies between task isolation and assessment isolation: task isolation means an agent can execute multiple tasks in parallel without interference, while assessment isolation ensures each evaluation run is reproducible and unaffected by previous states. The controller enforces this reproducibility by resetting the agent before every assessment, providing a clean start each time. Even if an agent doesn't support task isolation internally, it can still participate in assessments as long as the controller (or an equivalent mechanism) maintains proper assessment isolation.