Step-by-Step Checklist for Building Your Green Agent

Step-by-Step Checklist

1. Choose the task you want to evaluate on
   - E.g., Ticket-booking agent

Step-by-Step Checklist

2. Design the environment that the agents being tested needs to run in

- The tools that the agent can interact with, the actions that the agent can make, and the env feedback to the agent after each action
- E.g., Tools can be web browser or an APP for ticket booking. Actions can be mouse clicking and keyboard typing, or the APIs provided by the APP. Env feedback can be the new webpage popped up every time the agent clicks on a button.

Step-by-Step Checklist

3. Design the metrics that your green agent evaluates with

- E.g., the success rate of booking a ticket; how cheap the ticket is; whether the ticket satisfies user’s requirements; etc.

Step-by-Step Checklist

4. Design test cases to evaluate your green agent

- Think about different scenarios of white agents trying to complete the task
- Design test cases of white agents succeeding/failing to complete the task in different ways, along with ground-truth eval result for these cases.
- Include as many edge cases as possible
- Use these test cases to evaluate if your green agent gives reliable evaluation results.
- E.g., test cases can include a white agent successfully books the ticket; a white agent books the wrong ticket/a more expensive ticket; a white agent fails to find the website for booking tickets; etc.

NeurIPS 2025 Datasets & Benchmarks Track Call for Papers

The **NeurIPS Datasets and Benchmarks track** serves as a venue for high-quality publications on highly valuable machine learning datasets and benchmarks crucial for the development and continuous improvement of machine learning methods. Previous editions of the Datasets and Benchmarks track were highly successful and continuously growing (accepted papers 2021, 2002, and 2023, and best paper awards 2021, 2022, 2023 and 2024. Read our original blog post for more about why we started this track, and the 2025 blog post announcing this year's track updates.

Dates and Guidelines

Please note that the Call for Papers of the NeurIPS2025 Datasets & Benchmarks Track this year **will follow the Call for Papers of the NeurIPS2025 Main Track, with the addition of three track-specific points:**

- Single-blind submissions
- Required dataset and benchmark code submission
- Specific scope for datasets and benchmarks paper submission

The dates are also identical to the main track:

- **Abstract submission deadline:** May 11, 2025 AoE
- **Full paper submission deadline:** May 15, 2025 AoE (all authors must have an OpenReview profile when submitting)
- **Technical appendices and supplemental materials deadline:** May 22, 2025 AoE
- **Author notification:** Sep 18, 2025 AoE
- **Camera-ready:** Oct 23, 2025 AoE

Accepted papers will be published in the NeurIPS proceedings and presented at the conference alongside the main track papers. As such, we aim for an equally stringent review as in the main conference track, while also allowing for **track-specific guidelines**, which we introduce below. For details on everything else, e.g. formatting, code of conduct, ethics review, important dates, and any other submission related topics, please refer to the main track CFP.

OpenReview

Submit at: https://openreview.net/group?id=NeurIPS.cc/2025/Datasets_and_Benchmarks_Track

The site will start accepting submissions on April 3, 2025 (at the same time as the main track).

Note: submissions meant for the main track should be submitted to a different OpenReview portal, as shown here. Papers will not be transferred between the main and the Datasets and Benchmarks tracks after the submission is closed.

Project Grading Rubric [for new benchmarks]

- **Goal & Novelty:** Is your benchmark important, novel, and covering new capability space?
- **Scope & Scale:** Is the benchmark large and diverse enough to give reliable results?
- **Evaluator Quality:** Are metrics clear? Is your judge/evaluator high quality and consistent?
- **Validation:** Did you perform manual checks or spot validation on the evaluation outputs from your green agent?
- **Reliability:** Do your evaluation scripts and green agents run robustly on AgentBeats?
- **Quality Assurance:** Any bias or contamination checks included?
- **Realism**: Is the benchmark realistic, e.g., with real world workload, instead of toy or unrealistic settings
- **Impact:** Is the benchmark reusable, well-documented, and presented clearly?

Project Grading Rubric [for existing benchmarks]

- **Analysis:** Analyze quality issues of the original benchmark and find any flaws it has.
- **Faithfulness:** Is your implementation reproducing the results from the original benchmark (excluding the flaws you fixed)?
- **Quality Assurance:** Is your implementation correcting flaws in the original benchmark and expanding the coverage of the original benchmark?
- **Evaluator Quality:** Are metrics clear? Is your judge/evaluator high quality and consistent?
- **Validation:** Did you perform manual checks or spot validation on the evaluation outputs from your green agent?
- **Reliability:** Do your evaluation scripts and green agents run robustly on AgentBeats?
- **Quality Assurance:** Any bias or contamination checks included?
- **Impact:** Is your implementation reusable, well-documented, and presented clearly?

Green Agent Project Ideas

- https://docs.google.com/presentation/d/1TjtEjh6g9dZBsGxmAmcSp2EFakbmHpU_z31vnkf0c2Y/

Coding Example:
Supporting *Tau-bench*

1. Sort out the interface

Principles:

1. **Human should be able to solve it if presented the same task.**
2. **The solving procedure should be as agent-friendly as possible. (so that the agent can solve it)**

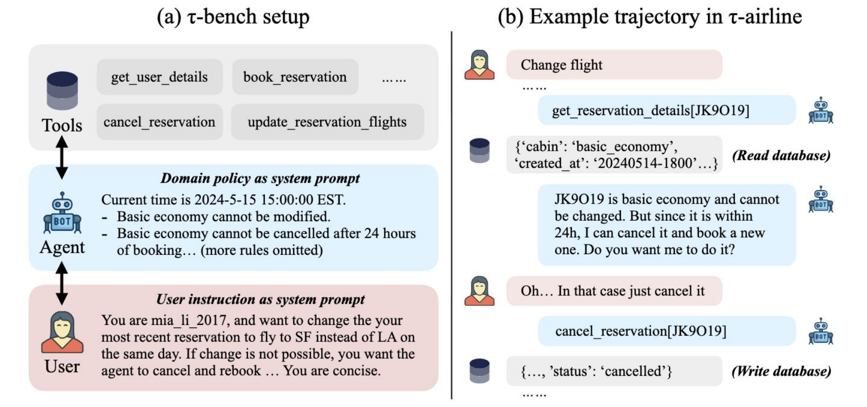
Example:

- **Web browsing agent**: url vs. tool actions
- **Coding agent**: provide coding env vs. provide repository & expect patches
- **Werewolf game agent**: text-based vote confirmation vs. tool-based confirmation

1. Sort out the interface

- Read the paper → think about task formulation
- Read their codebase → see how to deliver the ti

u



(a) τ-bench setup

Tools: get_user_details, book_reservation, ……, cancel_reservation, update_reservation_flights

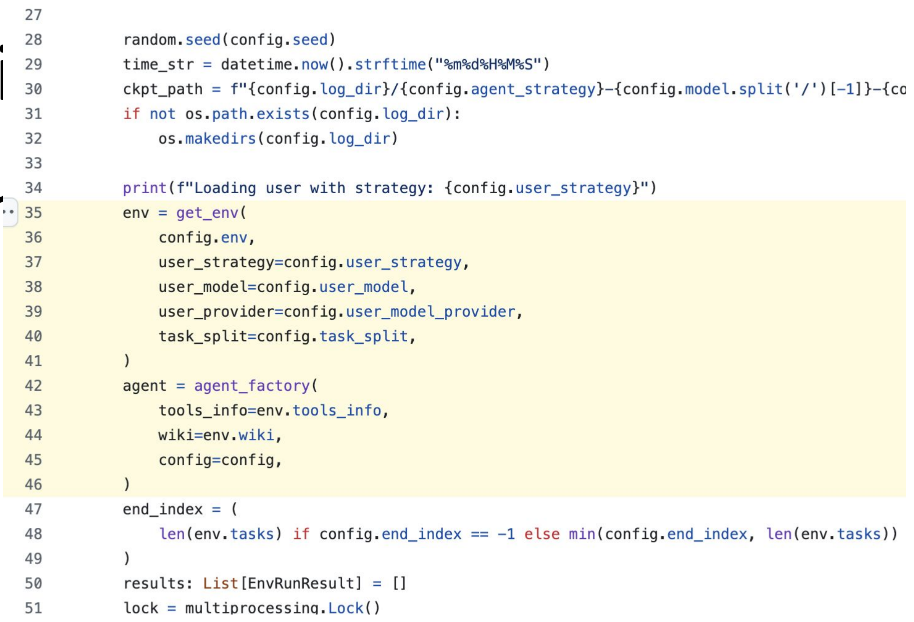
Agent
Domain policy as system prompt
Current time is 2024-5-15 15:00:00 EST.
- Basic economy cannot be modified.
- Basic economy cannot be cancelled after 24 hours of booking… (more rules omitted)

User
User instruction as system prompt
You are mia_li_2017, and want to change the your most recent reservation to fly to SF instead of LA on the same day. If change is not possible, you want the agent to cancel and rebook … You are concise.

(b) Example trajectory in τ-airline

Change flight
……
get_reservation_details[JK9O19]
{'cabin': 'basic_economy', 'created_at': '20240514-1800'…} *(Read database)*
JK9O19 is basic economy and cannot be changed. But since it is within 24h, I can cancel it and book a new one. Do you want me to do it?
Oh… In that case just cancel it
cancel_reservation[JK9O19]
{…, 'status': 'cancelled'} *(Write database)*
……

Figure 1: (a) In τ-bench, an agent interacts with database API tools and an **LM-simulated user** to complete tasks. The benchmark tests an agent's ability to collate and convey all required information from/to users through multiple interactions, and solve complex issues on the fly while ensuring it **follows guidelines** laid out in a domain-specific policy document. (b) An example trajectory in τ-airline, where an agent needs to reject the user request (change a basic economy flight) following domain policies and propose a new solution (cancel and rebook). This challenges the agent in long-context zero-shot reasoning over complex databases, rules, and user intents.



```

    random.seed(config.seed)
    time_str = datetime.now().strftime("%m%d%H%M%S")
    ckpt_path = f"{config.log_dir}/{config.agent_strategy}-{config.model.split('/')[-1]}-{co
    if not os.path.exists(config.log_dir):
        os.makedirs(config.log_dir)

    print(f"Loading user with strategy: {config.user_strategy}")
    env = get_env(
        config.env,
        user_strategy=config.user_strategy,
        user_model=config.user_model,
        user_provider=config.user_model_provider,
        task_split=config.task_split,
    )
    agent = agent_factory(
        tools_info=env.tools_info,
        wiki=env.wiki,
        config=config,
    )
    end_index = (
        len(env.tasks) if config.end_index == -1 else min(config.end_index, len(env.tasks))
    )
    results: List[EnvRunResult] = []
    lock = multiprocessing.Lock()
```

1. Sort out the interface

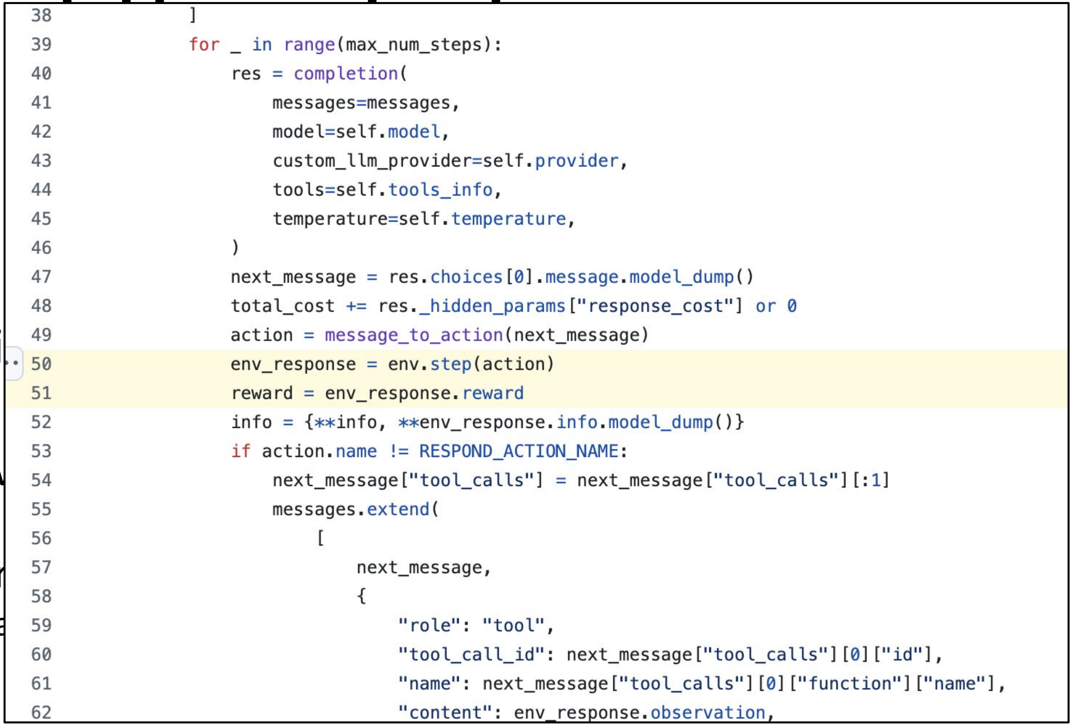
Two key challenges:

1. Cross-agent tool use
   a. In the original repo, tool is directly provided to “completion” interface
   b. How shall we evaluate using a standardized agent interface
      i. “Special” white agents, with tool access
         1. Less standardized
      ii. Explain this tool-access request to white agent, then ask for tool names / args
         1. Problem: cannot leverage agent internal tool-call mechanisms
      iii. Provide an MCP → require dynamic discovery
2. Migrate evaluation
   a. Tool trace is not directly visible to green agent

1. Sort o

Two key challenges:

1. Cross-agent tool use
   a. In the original repo, tool i
   b. How shall we evaluate us
      i. "Special" white agents, v
         1. Less standardized
      ii. Explain this tool-access r
         1. Problem: cannot levera
      iii. Provide an MCP → requ
2. Migrate evaluation
   a. Tool trace is not directly visible to green agent

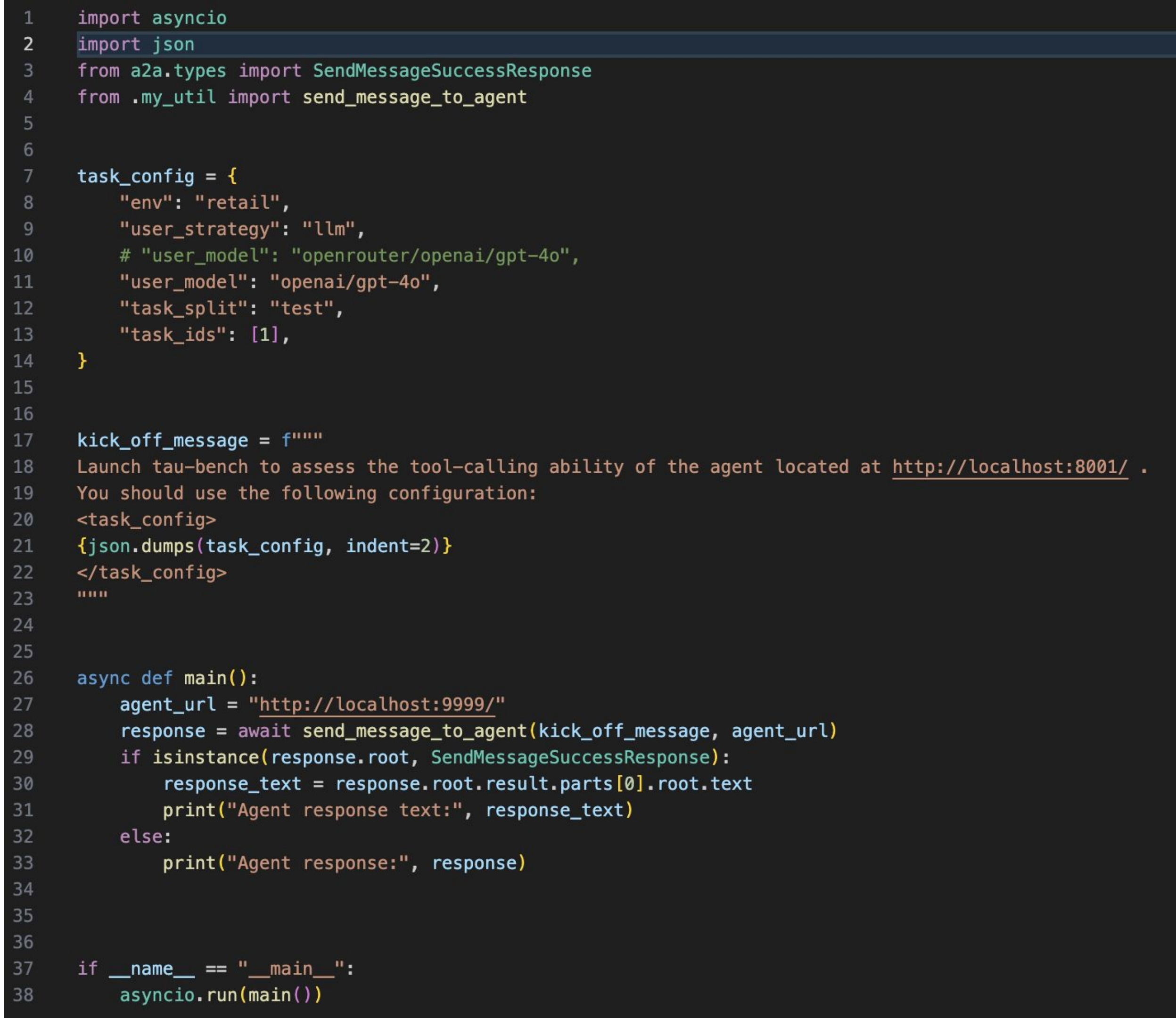


```
        ]
        for _ in range(max_num_steps):
            res = completion(
                messages=messages,
                model=self.model,
                custom_llm_provider=self.provider,
                tools=self.tools_info,
                temperature=self.temperature,
            )
            next_message = res.choices[0].message.model_dump()
            total_cost += res._hidden_params["response_cost"] or 0
            action = message_to_action(next_message)
            env_response = env.step(action)
            reward = env_response.reward
            info = {**info, **env_response.info.model_dump()}
            if action.name != RESPOND_ACTION_NAME:
                next_message["tool_calls"] = next_message["tool_calls"][:1]
                messages.extend(
                    [
                        next_message,
                        {
                            "role": "tool",
                            "tool_call_id": next_message["tool_calls"][0]["id"],
                            "name": next_message["tool_calls"][0]["function"]["name"],
                            "content": env_response.observation,
```

https://github.com/sierra-research/tau-bench/blob/main/tau_bench/run.py#L20

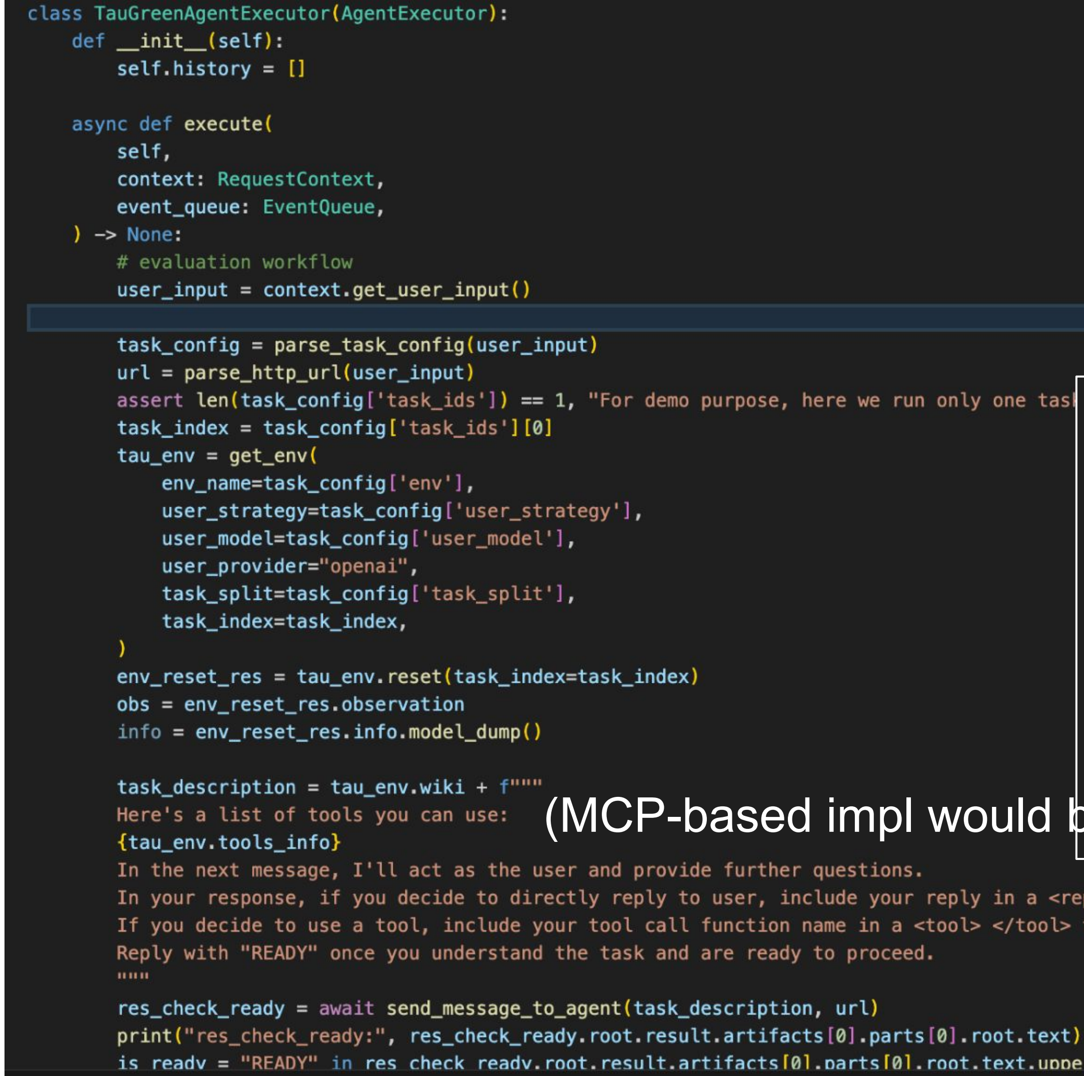
2. Design the workflow

- Kickoff script: send message to green agent to kick off the test
  - What information to include
  - Message format
- Green agent: coding-based, import tau_bench
  - How to change to the initial prompt
  - How to incorporate the final scoring procedure / what are the metrics
- White agent: prompt-based / LLM-workflow
  - Which SDK to use
  - What prompt might help with the performance

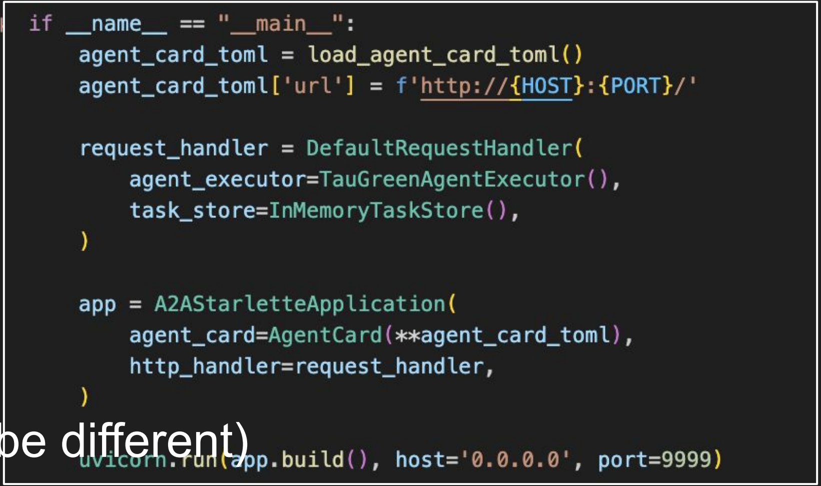


```python
import asyncio
import json
from a2a.types import SendMessageSuccessResponse
from .my_util import send_message_to_agent


task_config = {
    "env": "retail",
    "user_strategy": "llm",
    # "user_model": "openrouter/openai/gpt-4o",
    "user_model": "openai/gpt-4o",
    "task_split": "test",
    "task_ids": [1],
}


kick_off_message = f"""
Launch tau-bench to assess the tool-calling ability of the agent located at http://localhost:8001/ .
You should use the following configuration:
<task_config>
{json.dumps(task_config, indent=2)}
</task_config>
"""


async def main():
    agent_url = "http://localhost:9999/"
    response = await send_message_to_agent(kick_off_message, agent_url)
    if isinstance(response.root, SendMessageSuccessResponse):
        response_text = response.root.result.parts[0].root.text
        print("Agent response text:", response_text)
    else:
        print("Agent response:", response)



if __name__ == "__main__":
    asyncio.run(main())
```

3. Impl: Green agent

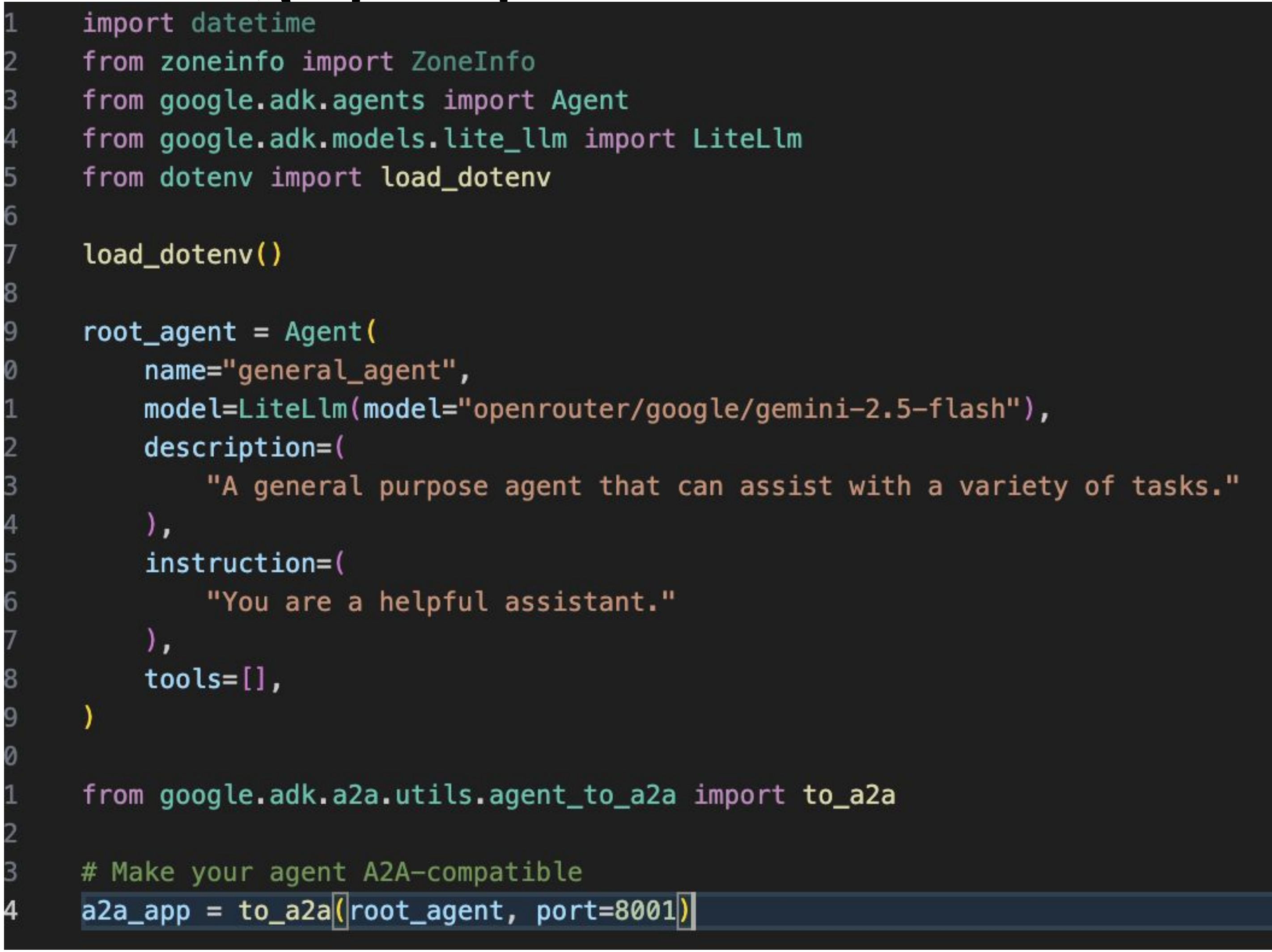


```python
class TauGreenAgentExecutor(AgentExecutor):
    def __init__(self):
        self.history = []

    async def execute(
        self,
        context: RequestContext,
        event_queue: EventQueue,
    ) -> None:
        # evaluation workflow
        user_input = context.get_user_input()

        task_config = parse_task_config(user_input)
        url = parse_http_url(user_input)
        assert len(task_config['task_ids']) == 1, "For demo purpose, here we run only one tas
        task_index = task_config['task_ids'][0]
        tau_env = get_env(
            env_name=task_config['env'],
            user_strategy=task_config['user_strategy'],
            user_model=task_config['user_model'],
            user_provider="openai",
            task_split=task_config['task_split'],
            task_index=task_index,
        )
        env_reset_res = tau_env.reset(task_index=task_index)
        obs = env_reset_res.observation
        info = env_reset_res.info.model_dump()

        task_description = tau_env.wiki + f"""
        Here's a list of tools you can use:
        {tau_env.tools_info}
        In the next message, I'll act as the user and provide further questions.
        In your response, if you decide to directly reply to user, include your reply in a <reply> </reply> tag.
        If you decide to use a tool, include your tool call function name in a <tool> </tool> tag, and include the arguments in a <args> </args> tag in JSON format.
        Reply with "READY" once you understand the task and are ready to proceed.
        """
        res_check_ready = await send_message_to_agent(task_description, url)
        print("res_check_ready:", res_check_ready.root.result.artifacts[0].parts[0].root.text)
        is_ready = "READY" in res_check_ready.root.result.artifacts[0].parts[0].root.text.upper()
```



```python
if __name__ == "__main__":
    agent_card_toml = load_agent_card_toml()
    agent_card_toml['url'] = f'http://{HOST}:{PORT}/'

    request_handler = DefaultRequestHandler(
        agent_executor=TauGreenAgentExecutor(),
        task_store=InMemoryTaskStore(),
    )

    app = A2AStarletteApplication(
        agent_card=AgentCard(**agent_card_toml),
        http_handler=request_handler,
    )

    uvicorn.run(app.build(), host='0.0.0.0', port=9999)
```

(MCP-based impl would be different)



```python
import datetime
from zoneinfo import ZoneInfo
from google.adk.agents import Agent
from google.adk.models.lite_llm import LiteLlm
from dotenv import load_dotenv

load_dotenv()

root_agent = Agent(
    name="general_agent",
    model=LiteLlm(model="openrouter/google/gemini-2.5-flash"),
    description=(
        "A general purpose agent that can assist with a variety of tasks."
    ),
    instruction=(
        "You are a helpful assistant."
    ),
    tools=[],
)

from google.adk.a2a.utils.agent_to_a2a import to_a2a

# Make your agent A2A-compatible
a2a_app = to_a2a(root_agent, port=8001)
```

Next step: integration with AgentBeats

After impl green/white/kick_off → 90% DONE

Next: make it reproducible & open accessible → leverage agentbeats

Update checklist:

1. How to get (remote) agent URL / MCP server URL
2. How to access LLM API
3. How to report result & add traces
4. Package the repo for platform hosting

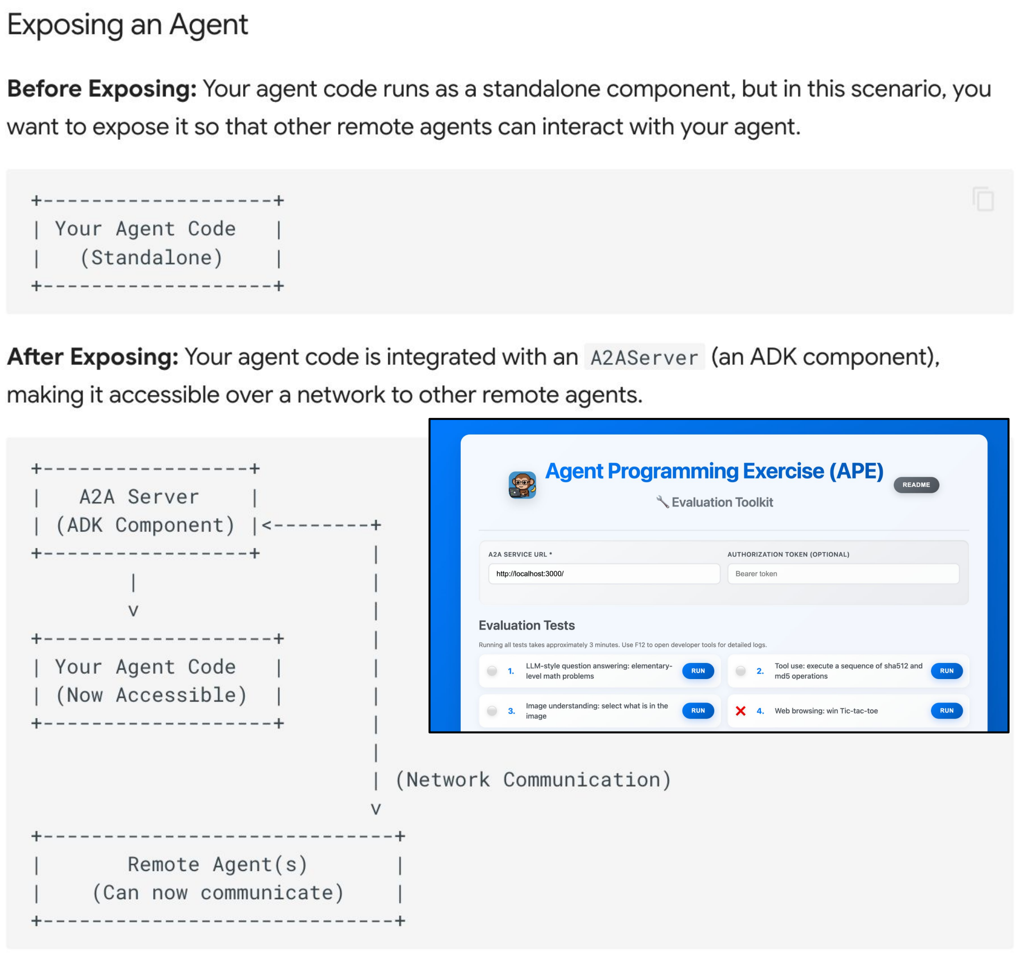
→ see documentation

Helpfu

https://google.github.io/adk-docs/a2a/intro/

https://a2a-protocol.org/latest/

http://ape.agentbeats.org/



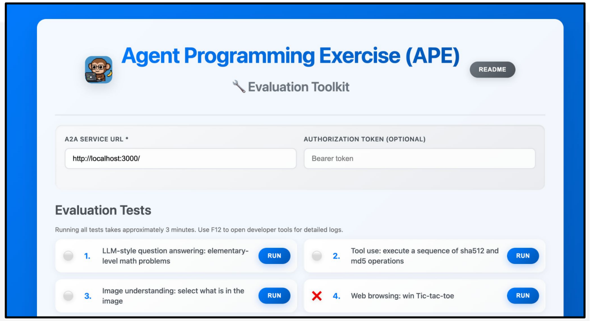
Exposing an Agent

Before Exposing: Your agent code runs as a standalone component, but in this scenario, you want to expose it so that other remote agents can interact with your agent.

```
+-------------------+
| Your Agent Code   |
|   (Standalone)    |
+-------------------+
```

After Exposing: Your agent code is integrated with an `A2AServer` (an ADK component), making it accessible over a network to other remote agents.

```
+-----------------+
|   A2A Server    |
| (ADK Component) |<---------+
+-----------------+          |
        |                    |
        v                    |
+-------------------+        |
| Your Agent Code   |        |
| (Now Accessible)  |        |
+-------------------+        |
                             |
                             |  (Network Communication)
                             v
+-----------------------------+
|       Remote Agent(s)       |
|    (Can now communicate)    |
+-----------------------------+
```



Agent Programming Exercise (APE)

🔧 Evaluation Toolkit

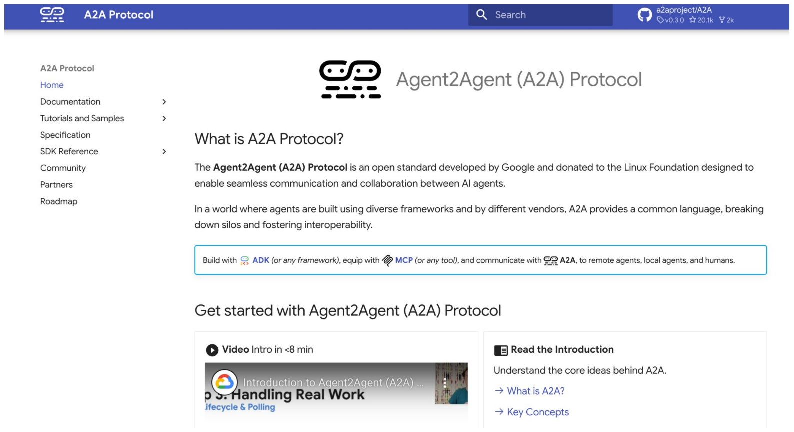
A2A SERVICE URL *: http://localhost:3000/

AUTHORIZATION TOKEN (OPTIONAL): Bearer token

Evaluation Tests

Running all tests takes approximately 3 minutes. Use F12 to open developer tools for detailed logs.

1. LLM-style question answering: elementary-level math problems RUN
2. Tool use: execute a sequence of sha512 and md5 operations RUN
3. Image understanding: select what is in the image RUN
4. ✗ Web browsing: win Tic-tac-toe RUN



A2A Protocol
- Home
- Documentation
- Tutorials and Samples
- Specification
- SDK Reference
- Community
- Partners
- Roadmap

Agent2Agent (A2A) Protocol

What is A2A Protocol?

The **Agent2Agent (A2A) Protocol** is an open standard developed by Google and donated to the Linux Foundation designed to enable seamless communication and collaboration between AI agents.

In a world where agents are built using diverse frameworks and by different vendors, A2A provides a common language, breaking down silos and fostering interoperability.

Build with ADK *(or any framework)*, equip with MCP *(or any tool)*, and communicate with **A2A**, to remote agents, local agents, and humans.

Get started with Agent2Agent (A2A) Protocol

Video Intro in <8 min

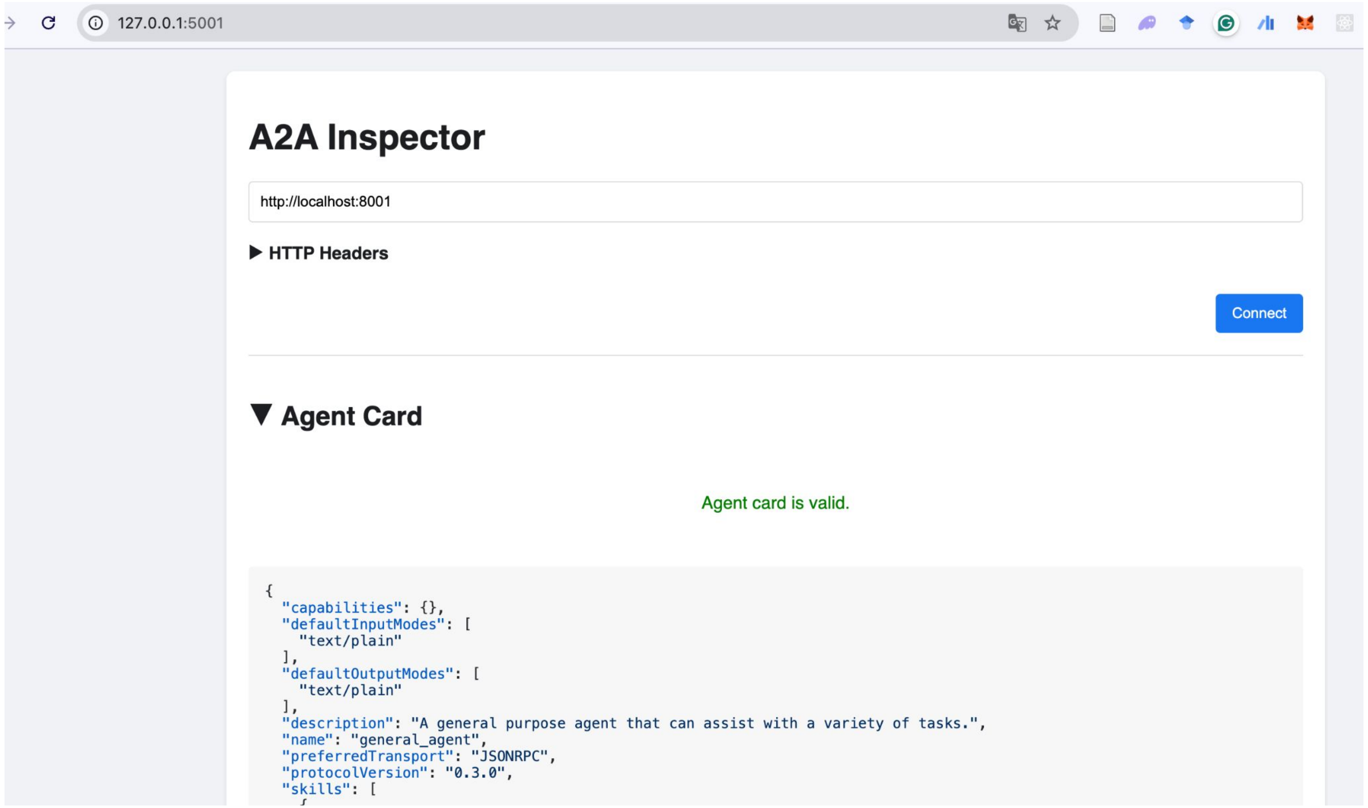
Read the Introduction

Understand the core ideas behind A2A.

→ What is A2A?

→ Key Concepts

Helpful tools



127.0.0.1:5001

A2A Inspector

http://localhost:8001

▶ HTTP Headers

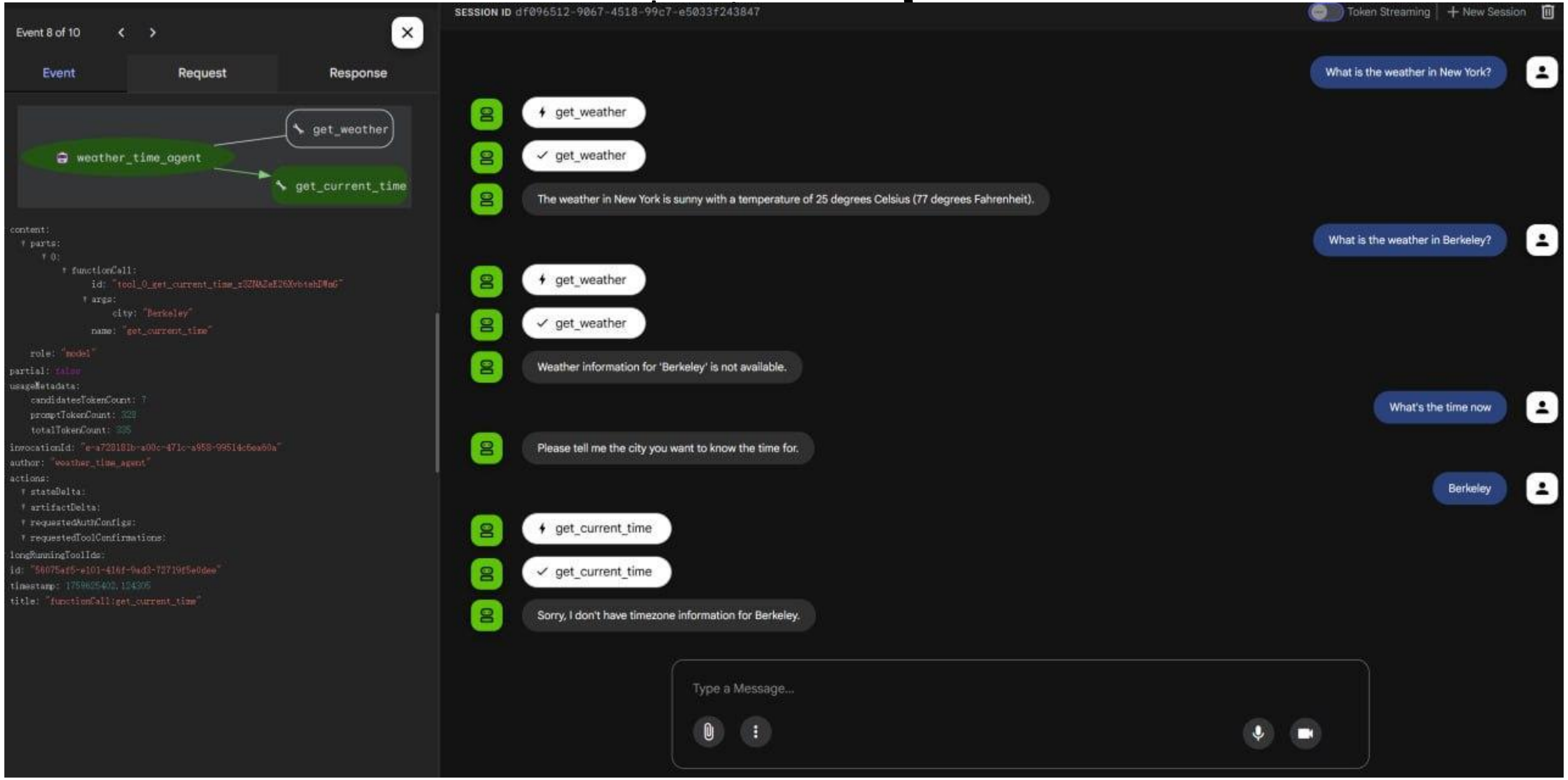
Connect

▼ Agent Card

Agent card is valid.

```
{
  "capabilities": {},
  "defaultInputModes": [
    "text/plain"
  ],
  "defaultOutputModes": [
    "text/plain"
  ],
  "description": "A general purpose agent that can assist with a variety of tasks.",
  "name": "general_agent",
  "preferredTransport": "JSONRPC",
  "protocolVersion": "0.3.0",
  "skills": [
```

Helpful tools (Google ADK, for OpenAI, check the online)



SESSION ID df096512-9067-4518-99c7-e5033f243847

```
content:
  ▾ parts:
      ▾ 0:
          ▾ functionCall:
                id: "tool_0_get_current_time_z32NAZeE26XvbtehDWmG"
              ▾ args:
                    city: "Berkeley"
                name: "get_current_time"
    role: "model"
partial: false
usageMetadata:
    candidatesTokenCount: 7
    promptTokenCount: 328
    totalTokenCount: 335
invocationId: "e-a72818lb-a00c-471c-a958-99514c6ea60a"
author: "weather_time_agent"
actions:
  ▾ stateDelta:
  ▾ artifactDelta:
  ▾ requestedAuthConfigs:
  ▾ requestedToolConfirmations:
longRunningToolIds:
id: "56075af5-e101-416f-9ad3-72719f5e0dee"
timestamp: 1759625402.124305
title: "functionCall:get_current_time"
```

What is the weather in New York?

⚡ get_weather

✓ get_weather

The weather in New York is sunny with a temperature of 25 degrees Celsius (77 degrees Fahrenheit).

What is the weather in Berkeley?

⚡ get_weather

✓ get_weather

Weather information for 'Berkeley' is not available.

What's the time now

Please tell me the city you want to know the time for.

Berkeley

⚡ get_current_time

✓ get_current_time

Sorry, I don't have timezone information for Berkeley.

Reference & Credits

- https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1246/slides/cs224n-spr2024-lecture11-evaluation-yann.pdf
- https://arxiv.org/abs/2503.16416
- https://arxiv.org/abs/2507.02825

Q&A

agentbeats concepts & platform / standardizing agent evaluation / …