Agent Evaluations

Shreya Havaldar

Announcements

- **Make-up 50% of lost points for HW 3**
  - If you are unhappy with your grade, you can add another component to your agent and demo at my OH on 4/10
  - The added component can be anything — be creative!
- *Note: This cannot be applied towards the 10 point deduction for missing your demo*

Today’s Lecture

1. **Evals overview**
2. Benchmarking
3. Vibe coding!

Why evaluate agents?

Many agent systems look impressive in isolated examples.
But… isolated examples are not enough to establish quality.

An agent may work once, fail the next time, or succeed only on easy tasks.

Evaluation gives us a disciplined way to measure:

1. What an agent can actually do
2. How much confidence we should place in it

Agent eval vs. LLM eval

LLM eval:

- LLMs are judged from one input and one output.

Agent eval:

- Agents act over time, maintain context, call tools, make decisions, and sometimes change the external environment.
- We must evaluate
  - (1) what the agent says
  - (2) what the agent does

Agents can fail in hidden ways

An agent could return a reasonable-looking final answer while failing internally. It may:

- Call the wrong tool
- Misuse a retrieved document
- Ignore an instruction
- Repeat unnecessary steps
- Leave the environment in the wrong state.

→ **The final output alone cannot confirm the agent behaved correctly**

Defining evaluation

Agent evaluation is the systematic measurement of whether an agent completes tasks correctly under defined conditions.

Evals should be systematic:
→ follow a repeatable and explicit procedure

Evals should have defined conditions:
→setting of the evaluation (task, tools, environment, constraints, etc.) is specified in advance.

The core evaluation + iteration loop

1. Define tasks and success criteria
2. Run the agent on those tasks
3. Record traces and outcomes
4. Apply one or more graders
5. Aggregate the results
6. Use results to revise the system

What do we actually analyze?

For ordinary LLMs, the unit of analysis is a text response.

For agents, there are many units of analysis:

- The text response
- The sequence of tool calls
- The intermediate reasoning state
- The resulting environment state
- … etc.

What counts as success

Success depends on the task and the setting. Evaluation must be tailored to the actual use case.

> *A shopping assistant: “success” is finding the right item under the right budget and review constraints*

> *A scheduling agent: “success” is creating the correct meeting without violating calendar or policy constraints.*

How to measure success

Once you know what to analyze, how do you evaluate it?

A **“grader”** is any mechanism that decides whether a component of an agentic system performed well.

- You need different graders to analyze the output text, reasoning trace, tool calls, etc.
- The graders are some of the most important parts of an evaluation!

Types of graders

We usually see the following graders:

1. Rule-based
2. LLM-as-judge
3. Human evaluation

Rule-based grading

Works well when tasks have explicit answers or clear programmatic checks.

Weakness: it becomes brittle when tasks are open-ended or when multiple solutions exist.

```python
import json

def check_validity(tool_call_output: str):
    #ensure the tool call output is a valid JSON object
    try:
        json.loads(tool_call_output)
    except:
        return False
    #ensure the total price is less than $100
    total_price = json.loads(tool_call_output)["price"]
    if total_price > 100:
        return False
    return True
```

Example: ensuring a shopping agent's tool call returns a valid JSON for an item under $100

LLM-as-judge grading

Useful when quality is hard to capture with simple rules.

Weakness: require proper calibration because they can be inconsistent, overly generous, or sensitive to prompt phrasing.

Example: ensuring the summary of a long document is relevant or coherent

Human evaluation

Useful when the task is nuanced, high-stakes, or difficult to score automatically.

Weakness: human evaluation is expensive, slower, and harder to scale.

Example: evaluating whether a customer service agent was helpful and efficient

Hybrid grading

In practice, the best grading strategies are often hybrid.

A full evaluation system may use:

- Rule-based checks for tool inputs/outputs
- LLM graders for open-ended reasoning traces
- Human review for overall agent usefulness

Metrics

Graders evaluate each component of an agentic system. How do we aggregate the results of graders across many runs?

Performance metrics usually fall into four major categories:

1. Task-level metrics
2. Trajectory-level metrics
3. Tool-level metrics
4. System-level metrics.

Task-level metrics

Task-level metrics ask whether the agent completed the specific task successfully.

> *Examples: accuracy, precision, recall, pass rate, resolution rate, final-state correctness, or user-goal completion.*

Task-level metrics focus on the output of a task, but they do not explain how the agent achieved the output.

Trajectory-level metrics

Trajectory-level metrics focus on the process the agent followed.

> *Examples: number of steps taken, unnecessary detours, repeated loops, recovery after tool failure, and plan sensibility/efficiency.*

Trajectory metrics are important to evaluate because bad processes can introduce risks/delays, even when the final answer is acceptable.

Tool-level metrics

Tool-level metrics evaluate how the agent interacts with external tools.

> *Examples: whether the agent selected the appropriate tool, passed correct arguments, and avoided redundant calls.*

Tool-level metrics are important because in many real systems, poor tool use is one of the main causes of agent failure.

System-level metrics

System-level metrics capture the more “engineering” side of performance.

> *Examples: latency, throughput, token usage, computational cost, and reliability under load.*

System-level metrics are important because a highly-capable agent may be impractical if it is too slow or too expensive.

Activity (15 min)

Get into groups of 2-3, and outline an evaluation framework for an agent!
https://tinyurl.com/4hhwwp73

You should include:

1. What tasks to evaluate and what counts as “success” for each task
2. What grader to make for each task (rule-based, LLM-as-a-judge, human eval)
3. How to run the evaluation system in practice – what metrics to log, what data is needed, etc.

Today’s Lecture

1. Evals overview
2. **Benchmarking**
3. Vibe coding!

Benchmarking

A benchmark is a standardized test used to compare how well different systems perform on the same set of tasks.

In AI, a benchmark usually includes:

- a fixed dataset or task suite
- a scoring method
- a leaderboard or shared baseline

Benchmarking

Benchmarking gives us a way to compare agents and models under shared conditions.

This is useful because **it creates a common reference point** across agents that share a goal.

- Example: comparing ChatGPT and Gemini
- Example: comparing Cursor and Claude Code

Benchmarking: Pros

Benchmarks are useful for:

- Broad capability comparisons
- Model selection
- Tracking progress over time.

They are especially helpful when **we want a fast external check of where a system stands relative to other systems.** They also provide a common language for discussing performance across the field.

Benchmarking: Cons

Benchmarks are NOT useful for:

- Matching your exact application

They may omit your tools, your users, your workflows, your safety constraints, or your domain-specific definitions of success.

→ **A strong benchmark score does not necessarily imply strong real-world performance.**

Benchmark leakage and overfitting

As benchmarks become popular, systems may be optimized directly or indirectly for those benchmarks.

This can lead to **overfitting**, where improvements in benchmark scores outpace improvements in real-world usefulness.

That is one reason custom evals remain necessary even when good benchmarks exist.

Examples of agent benchmarks

Popular agent benchmarks usually emerge because they isolate a capability that researchers and builders care about in practice.

- **GAIA:** whether an assistant can solve realistic, multi-step tasks.
- **WebArena:** whether an agent can act effectively in web environments.
- **SWE-bench:** whether an agent can do software engineering work.
- **Terminal-Bench:** whether an agent can complete command-line tasks
- **τ-bench:** whether a conversational agent can complete interactions successfully.

GAIA

Level 1

Question: What was the actual enrollment count of the clinical trial on H. pylori in acne vulgaris patients from Jan-May 2018 as listed on the NIH website?
Ground truth: 90

Level 2

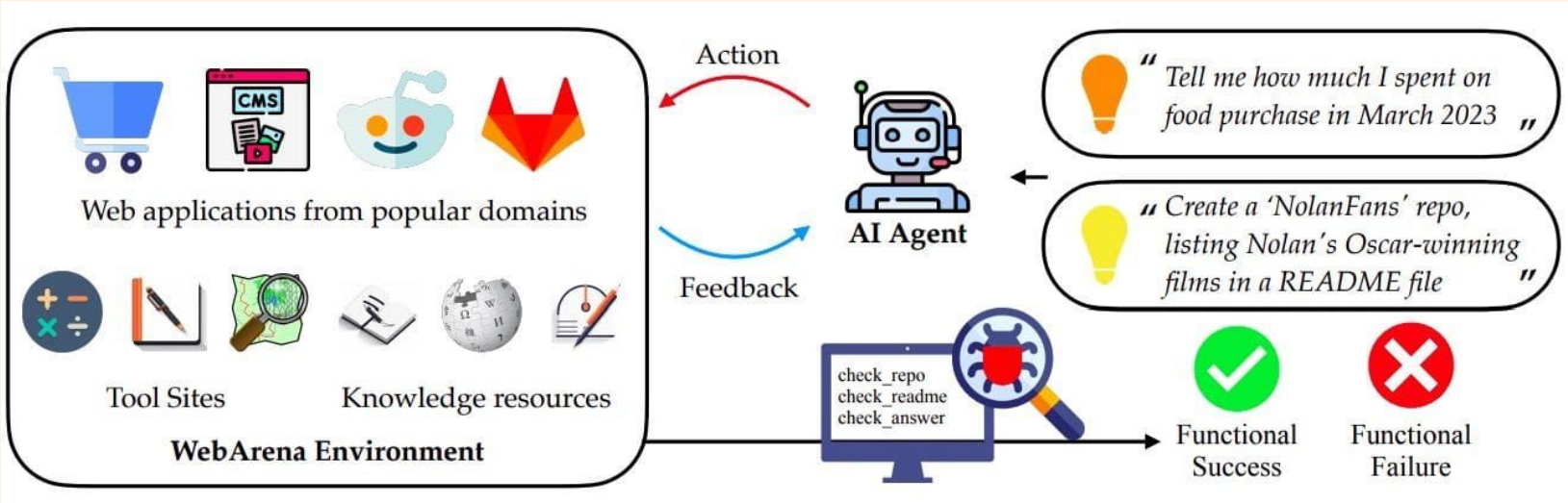
Question: If this whole pint is made up of ice cream, how many percent above or below the US federal standards for butterfat content is it when using the standards as reported by Wikipedia in 2020? Answer as + or - a number rounded to one decimal place.
Ground truth: +4.6

Level 3

Question: In NASA's Astronomy Picture of the Day on 2006 January 21, two astronauts are visible, with one appearing much smaller than the other. As of August 2023, out of the astronauts in the NASA Astronaut Group that the smaller astronaut was a member of, which one spent the least time in space, and how many minutes did he spend in space, rounded to the nearest minute? Exclude any astronauts who did not spend any time in space. Give the last name of the astronaut, separated from the number of minutes by a semicolon. Use commas as thousands separators in the number of minutes.
Ground truth: White; 5876

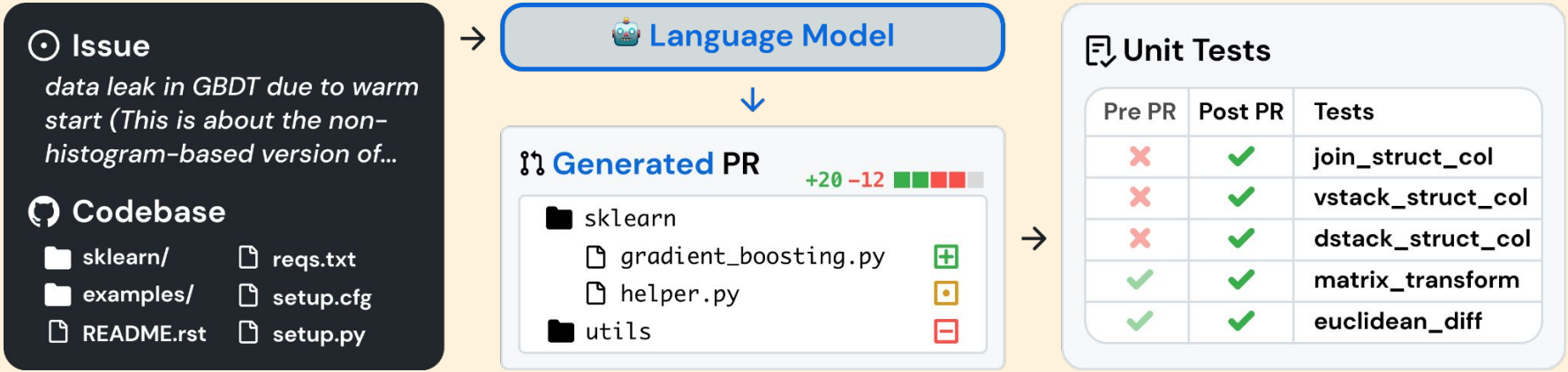
Evaluates general AI assistants on realistic multi-step tasks that often require reasoning, web browsing, multimodal understanding, and tool use.

WebArena



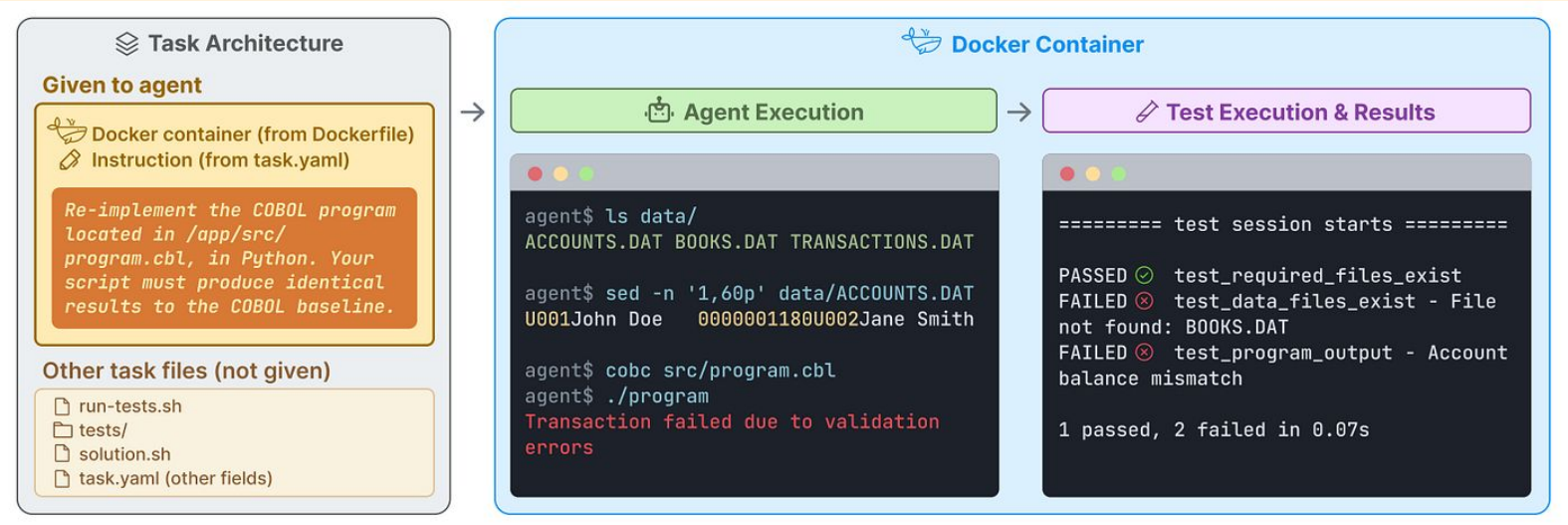
Evaluates autonomous web agents in realistic websites, focusing on browsing, navigation, and completing web-based tasks end to end.

SWE-bench



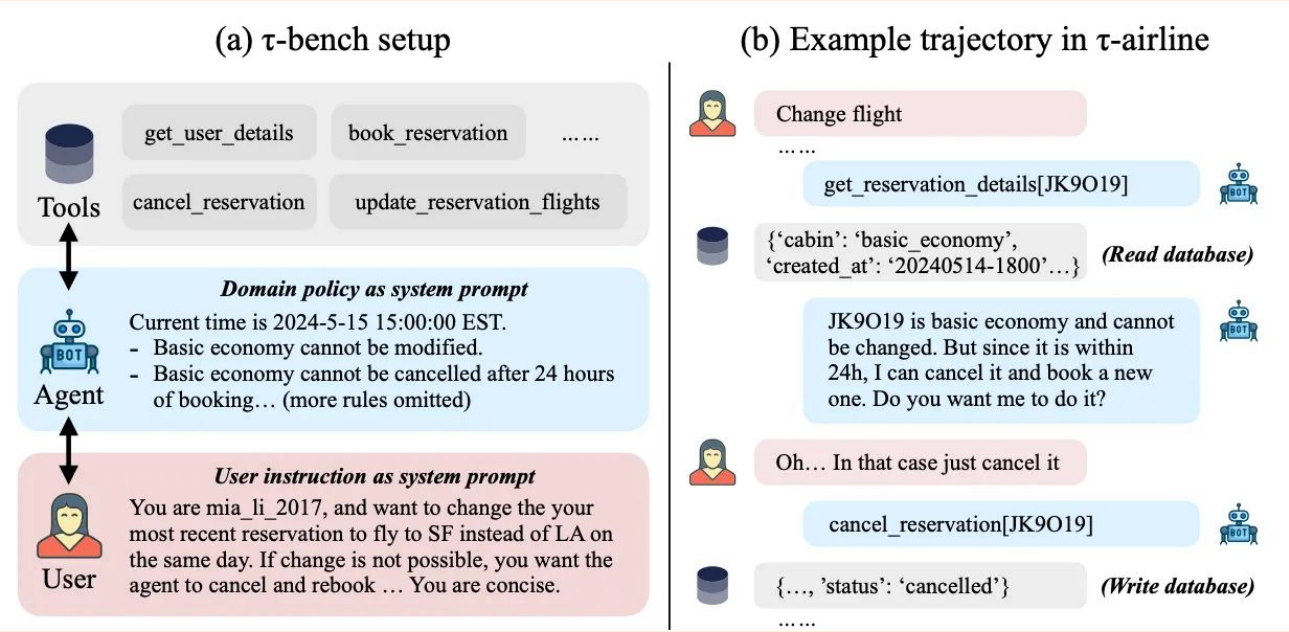
Evaluates coding agents on real GitHub issues using a human-validated subset designed to make software-engineering evaluation more reliable. A main benchmark for coding agents.

Terminal-Bench



Evaluates agents in real terminal environments on end-to-end tasks such as compiling code, debugging systems, training models, and other command-line workflows.

τ-bench



Evaluates tool-using conversational agents in customer-service-style settings, where the agent must interact with a user, follow domain policies, call APIs correctly, and end in the correct final state.

The key limitation of benchmarks

Benchmark scores are often incomplete signals.

→ Agents require a broader, more robust evaluation that includes custom tasks, real-world traces, safety testing, cost analysis, etc.

But… there is not enough high-quality human-labeled data to build a good evaluation set for every agent. **Synthetic data** offers a way to expand coverage and test more behaviors.

What is synthetic evaluation data?

Synthetic evaluation data consists of tasks, prompts, conversations, scenarios, or traces that are generated (1) programmatically or (2) using LLMs for evaluation purposes.

The key idea is that **these examples are not collected directly from users.** Instead, they are designed to probe specific behaviors, capabilities, or failure modes.

Why synthetic data is useful

- Can expand coverage cheaply and quickly.
- Can help create edge cases, rare failures, adversarial prompts, ambiguous instructions, or safety-sensitive scenarios that might not appear often in a small real dataset.
- Useful for stress-testing graders and evaluation pipelines.

Why synthetic data is risky

- Can create a false sense of coverage.
- Generated examples may be unrealistic, poorly specified, or repetitive, leading to a dataset broad in appearance but narrow in substance.
- Quality control requires human review and comparison with real data.

A basic synthetic data workflow

1. Define the task family and the kinds of failures we want to test.
2. Manually create a few seed examples
3. Use templates or LLMs to generate more cases
4. Validate: filter, deduplicate, and refine the set before using it in evaluation

The utility of synthetic data depends heavily on careful design, iteration, and review.

Today's Lecture

1. Evals overview
2. Benchmarking
3. **Vibe coding!**

Vibe coding tutorial from Davis!

Davis Brown

TA

davisbr at seas.upenn.edu

OH: Wed 2-3pm in AGH 300E

Davis is a 2nd year PhD working on AI safety & security. He likes running, surfing, and Robert Caro biographies. He dislikes added sugar.