CS 264 HW: ReAct Agent for Software Engineering Report

Hongxun Wu

December 14, 2025

1 Overview

This report describes improvements made to a minimal ReAct-style software engineering agent for the SWE-bench evaluation subset. The agent alternates between generating an action in a strict textual tool-call format and executing tools inside an isolated environment to produce a patch.

2 Key Improvements

2.1 Match the model's chat training distribution

The provided starter framework effectively places all history into a single large text blob, which can mismatch the distribution that modern chat models are trained on. I changed the agent to construct a *multi-turn* conversation (a list of messages with roles), and then pass those messages directly to the OpenAI Chat Completions API so that the API applies the correct chat template (OpenAI "harmony" chat formatting) internally. In practice, this reduced instruction drift and improved tool-use stability.

2.2 Repeat system instructions to improve long-context compliance

When the context grows long, the model often starts producing ill-formed function calls (missing markers, missing argument blocks, etc.). To mitigate this, I repeat the system message at the end of each round (in addition to the normal system message at the start). This provides a fresh copy of the tool descriptions and the required response format near the end of the prompt, improving instruction-following and reducing parse errors.

2.3 Make the end marker more forgiving

A frequent failure mode was the model emitting the end marker with slightly different punctuation, e.g. forgetting the trailing dashes in:

```
--END_FUNCTION_CALL--
```

To reduce brittleness, I changed the parser's end delimiter to the shorter prefix:

```
--END_FUNCTION_CALL
```

This accepts both the full marker and common "missing trailing dashes" variants, which significantly reduced parse failures.

3 Additional Engineering Guardrails

3.1 Respect OpenAI message-role constraints while keeping tool outputs distinct

This agent uses a *textual* function-call protocol (parsed by string markers) rather than OpenAI's native tool-calling interface. In OpenAI's native protocol, messages with `role="tool"` must correspond to a preceding assistant `tool_calls` entry with a matching `tool_call_id`. To avoid protocol violations while still keeping tool outputs distinct in the agent's internal state, the agent stores tool results as `role="tool"` internally but maps them to `role="user"` when sending the message list to the API.

3.2 Typed tool arguments

The textual tool-call format produces arguments as strings. Some tools (notably `replace_in_file`) require integer line numbers. I added automatic argument coercion based on the tool signature (e.g., converting `"266"` to `266` when an argument is annotated as `int`). This prevents common runtime errors such as comparisons between `str` and `int`.

3.3 Test-modification guardrail

SWE-bench evaluation disallows modifying or adding tests. The agent now detects changes to test paths (e.g., `tests/`, `test_*.py`, `*_test.py`) and automatically reverts/removes them, then instructs the model to re-implement the fix without touching tests.

4 Evaluation Results

Artifacts for this run:

- Predictions file: `results/preds.json`
- Per-instance trajectories: `results/<instance_id>/<instance_id>.traj.json`
- Evaluation summary: `gpt-5-mini.my_evaluation_run.json`

The evaluation summary reports:

- Total instances: 20
- Submitted: 20
- Completed: 20
- Resolved: 10
- Unresolved: 10
- Empty patch: 0
- Error: 0

Overall accuracy on this subset is **50%** (10/20 resolved).

4.1 Solved instances

- `django__django-11179`
- `django__django-13297`
- `django__django-13810`
- `django__django-14053`
- `django__django-16662`
- `django__django-7530`
- `scikit-learn__scikit-learn-26323`
- `sphinx-doc__sphinx-9658`
- `sympy__sympy-17655`
- `sympy__sympy-24213`

4.2 Unsolved instances

- `astropy__astropy-7166`
- `django__django-10973`
- `django__django-12406`
- `django__django-14011`
- `django__django-16631`
- `psf__requests-1921`
- `psf__requests-2931`
- `pytest-dev__pytest-7490`
- `sphinx-doc__sphinx-7590`
- `sphinx-doc__sphinx-9230`

5 Observation

The most impactful improvements were prompt-format alignment (multi-turn chat) and robustness against formatting drift (repeating the system message at the end of each round and relaxing the end marker). Together, these changes reduced the rate of invalid tool-call outputs and improved overall stability when operating in long contexts.

6 Statement of LLM Usage

I used an LLM to help write this report. Specifically, I verbally described (in natural language) the key design changes I made to the agent and the evaluation outcomes I observed, and then asked the LLM to draft an English LaTeX report based on those points. I reviewed and edited the generated text to ensure it matches my implementation and the reported results.