Enhancing Claude 3.5 Sonnet Reliability on SWE-bench Verified

Anushka Sheoran[1]

[1]Department of Computer and Information Science, University of Pennsylvania, Philadelphia, PA.

Abstract

Autonomous software engineering agents must navigate large codebases, reason about cross-file dependencies, and produce syntactically valid patches without human guidance, making real-world bug fixing one of the most demanding testbeds for agentic AI evaluation. This paper reproduces and analyzes the performance of Claude 3.5 Sonnet on SWE-bench Verified, a benchmark of 500 human-validated GitHub issues requiring autonomous code repair. A baseline agent using a minimal system prompt resolves 20% of 50 sampled tasks. Qualitative error analysis of all 27 unresolved tasks identifies Tool and Execution failures, primarily destructive over-deletion and syntax errors, as the dominant failure mode (44%), followed by Verification and Recovery failures (33%). Two prompt-based interventions are evaluated: a strict Test-Driven Development (TDD) protocol requiring a reproduction script before any edit, and a simplified mandatory self-verification protocol. The TDD protocol produces zero resolved tasks by causing the agent to stall before making any real fix. The simplified protocol maintains the 20% resolve rate while reducing raw errors (9 to 7), but an aggressive step limit reduction triggers a surge of over-deletion, revealing that step budget and prompt design interact non-trivially in agentic systems.

1 Introduction

Software engineering is one of the most complex real-world tasks for autonomous agents. Unlike isolated code generation, real-world bug fixing requires understanding large codebases, reasoning about cross-file dependencies, and producing syntactically valid, verified patches without human guidance. This makes it an ideal testbed for evaluating agentic AI systems.

Jimenez et al. [**jimenez2024**] introduced SWE-bench as a benchmark grounded in real GitHub history rather than synthetic puzzles. Each task provides an agent with a real bug report and a complete repository snapshot. Evaluation is binary and rigorous: a task is resolved only if the agent's patch satisfies two criteria simultaneously. First, FAIL_TO_PASS requires that tests which were failing on the buggy codebase now pass with the agent's fix. Second, PASS_TO_PASS requires that all tests which were passing before remain passing, ensuring no regressions are introduced. This dual requirement means a patch that fixes the target bug but breaks unrelated functionality still counts as unresolved. SWE-bench Verified [**openai2024**] is a 500-task human-validated subset that removes ambiguous or underspecified issues, providing a cleaner evaluation signal.

This project is exploratory in nature. Several directions were considered at the outset:

comparing resolve rates across model families at different price points, evaluating the effect of retrieval strategy on file localization, and examining whether multi-agent architectures meaningfully reduce the rate of destructive edits. The direction pursued here was narrower: given a fixed model and scaffold, how much does an explicit verification requirement in the system prompt change agent behavior and resolve rate? This question is tractable with limited compute budget and produces a self-contained result, while leaving the broader comparative questions open for future work.

I reproduce the SWE-bench Verified baseline using Claude 3.5 Sonnet and the mini-swe-agent scaffold [**minisweagent**], perform qualitative error analysis on all 27 unresolved tasks, and evaluate two prompt-based interventions of increasing strictness.

2 Materials and Methods

2.1 Experimental Design

I evaluate on SWE-bench Verified [**openai2024**] using a 50-task sample (slice 0:50 of the test split), drawing primarily from the Astropy and Django repositories. All experiments run on a Windows 10 host with WSL2 (Ubuntu) and Docker providing isolated sandboxes per task, using mini-swe-agent v2.2.6 [**minisweagent**] and the OpenRouter API for model access. Three configurations are evaluated: a baseline, a TDD pilot intervention, and a simplified verification intervention.

2.2 Baseline Configuration

The baseline uses Claude 3.5 Sonnet with a minimal vanilla system prompt: *"You are a helpful assistant that can interact with a computer shell to solve programming tasks."* The step limit is set to 250 and the per-task cost limit to $3.00, matching standard mini-swe-agent defaults.

2.3 Intervention Configurations

TDD Protocol (Pilot). The first intervention required the agent to follow a Test-Driven Development loop [**beck2002**], writing a `reproduce_issue.py` script catching the bug before editing any source file, run it to confirm failure, apply the fix, then rerun to confirm success. Submission was blocked until the reproduction script passed. This was evaluated on 27 tasks. The exact system prompt used was:

```
# CRITICAL: MANDATORY VERIFICATION PROTOCOL
# You are currently failing tasks due to
#   'NameErrors' and 'Malformed Patches.'
# You are REQUIRED to follow this
#   Test-Driven Development (TDD) loop:
# 1. Create a reproduction script
#    'reproduce_issue.py' that catches
#    the bug.
# 2. Run it and confirm it fails
#    (Exit Code != 0).
# 3. Apply your code fix using the
#    'edit' or 'sed' tools.
# 4. IMMEDIATELY run
#    'python reproduce_issue.py'
#    to verify your fix.
# 5. If you see a 'NameError',
#    'SyntaxError', or 'AttributeError',
#    you MUST fix your code before
#    submitting.
# 6. DO NOT use 'COMPLETE_TASK_AND_SUBMIT'
#    until 'reproduce_issue.py' passes
#    with exit code 0. Evidence of a
#    passing test is mandatory for
#    success.
```

Simplified Verification Protocol. The final intervention replaces the vanilla prompt with five rules designed to enforce self-verification without creating a hard submission gate. The step limit was reduced to 30 and the cost limit to $1.00 via the following configuration:

```
step_limit: 30
cost_limit: 1.
```

The exact system prompt used was:

```
You are a helpful assistant that can
interact with a computer shell to solve
programming tasks.
Before submitting, you MUST:
1. Run your fix and confirm it works
   (exit code 0)
2. Specifically make sure there are no
   syntax errors and you reference the
   correct variables
3. If it fails, fix it and retry |
   maximum 3 attempts.
4. Submit your best attempt after
   3 tries regardless.
5. DO NOT loop infinitely

Do NOT submit unverified code.
```

Rules 3 through 5 constitute a soft retry budget intended to prevent infinite loops. Whether these rules were ever triggered during the 50 tasks is not known from the available logs. The hard step limit of 30 was reached in exactly one of 50 tasks. The model and all infrastructure remain identical to the baseline.

2.4 Error Analysis Framework

All 27 unresolved baseline tasks are analyzed using two complementary frameworks. The first categorizes failures by behavioral mode following the course taxonomy: Goal and Constraint Errors, Planning and Decomposition Errors, State and Memory Errors, Tool and Execution Errors, and Verification and Recovery Failures. The second categorizes by technical manifestation visible in the evaluation harness output, including over-deletion, coding error, syntax error, and logic regression. The same analysis is applied to the 27 unresolved intervention tasks to measure the shift in failure distribution.

3 Results

3.1 Quantitative Outcomes

Baseline results and both intervention configurations are summarized in Table 1. The baseline resolves 10 of 50 tasks (20%). The TDD protocol resolves zero tasks across 27 trials, with 19 empty patches. The simplified protocol maintains 20% resolution while shifting the failure distribution.

Table 1: Results across all three configurations.

Configuration	Res.	Empty	Err.	%
Baseline (n=50)	10	4	9	20%
TDD pilot (n=27)	0	19	4	0%
Simplified (n=50)	10	6	7	20%

3.2 Baseline Error Analysis

Tool and Execution failures dominate at 12 of 27 unresolved tasks (44%), followed by Verification and Recovery at 9 (33%), Planning and Decomposition at 3 (11%), Goal and Constraint at 2 (7%), and Environment and Infrastructure at 1 (4%). Figure 1 shows the full distribution.

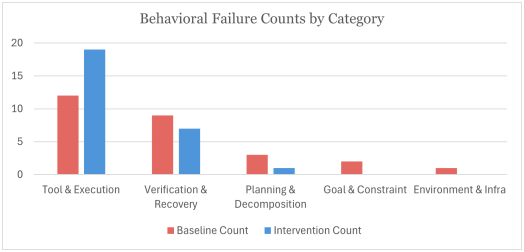

Figure 1: Behavioral failure categories for 27 unresolved baseline tasks. Tool and Execution failures dominate at 44%.

Within Tool and Execution failures, the dominant pattern is destructive over-deletion: the agent rewrites or removes large code sections rather than making surgical edits, deleting essential functions, classes, or infrastructure. For example, django-10097 removed all email validator functions and django-11299 deleted the

`chain` method from `Query`. Additional sub-types include coding errors, syntax errors, logic regressions, and edge case failures (Table 2).

Table 2: Technical failure sub-types across baseline and simplified intervention.

Sub-type	Base.	Interv.
Over-deletion	5	7
Logic regression	4	6
Syntax error	2	6
Coding error	3	5
Misreads feedback	3	1
Flawed planning / missing subgoal	3	1
Copy-paste / editing error	2	1
Misunderstands constraints	2	0
Edge case failure	2	0
Environment / library issue	1	0

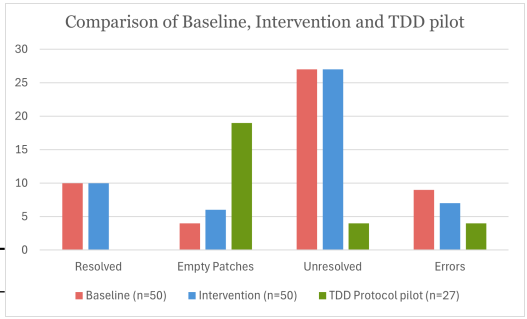

Figure 2: Baseline vs. simplified intervention outcomes across 50 tasks. Resolved rate is unchanged; errors decrease while empty patches increase.

3.3 Intervention Results

The TDD protocol failed entirely. Of 27 tasks, 19 produced empty patches and 4 produced infrastructure errors. Of the 19 empty patches, 18 were caused by API errors mid-run, indicating the agent exhausted its token budget inside the reproduction script loop rather than reaching the editing stage at all. The blocking submission rule meant that once the agent stalled on the reproduction script, no fix was ever attempted.

The simplified protocol maintains 20% resolution while shifting the failure distribution (Figure 2). Errors decrease from 9 to 7, but empty patches increase from 4 to 6. Behavioral analysis reveals that Verification and Recovery failures decrease from 9 to 7, providing evidence the prompt worked as intended, while Tool and Execution failures surge from 12 to 19. Over-deletion cases increase from 4 to 7, with several involving massive deletions: django-11333 removed approximately 500 lines of core URL resolver infrastructure, and django-11133 deleted the entire `HttpResponse` class definition.

4 Discussion

The null resolve rate result conceals two opposing effects operating simultaneously. Verification and Recovery failures decreased (9 to 7), providing evidence that the prompt intervention worked as intended for that failure mode. Tool and Execution failures increased sharply (12 to 19), driven by the step limit reduction from 250 to 30. These effects canceled at the aggregate resolve rate level.

The most likely mechanism for the over-deletion surge is behavioral rather than mechanical. The hard step limit of 30 was reached in only one of 50 tasks, and the soft retry budget in rules 3 through 5 of the prompt was never triggered. The agent was therefore not simply running out of steps. Instead, the tighter budget appears to have changed how the agent planned its approach: with less room to explore and iterate, it favored single large rewrites over careful incremental edits, increasing the risk of deleting too much in one pass. The increase in syntax errors (3 to 6), predominantly indentation errors, supports this interpretation: single-pass rewrites are more likely to introduce formatting mistakes than iterative edits with intermediate verification.

A cleaner framing of the result is that step budget and prompt design are not independent

levers. Changing the step limit changes not just how many edits the agent makes, but how it edits, qualitatively altering the failure distribution in ways that can offset improvements from better prompt design.

Model Capability and Cost Trade-offs

The current SWE-bench Verified leaderboard is dominated by high-reasoning models: Claude 4.5 Opus leads at 74.4% at approximately $0.75 per task, followed by Claude 4.5 Sonnet at 70.6% at $0.56 per task. Claude 3.5 Sonnet, used in this project, costs approximately $0.047 per task and achieves 20% on our 50-task sample, illustrating a steep capability-cost tradeoff. A natural next step is evaluating Claude Sonnet 4.6 ($0.37 per task), which sits between these extremes on cost and achieves 64.9% on the full leaderboard. The open question is whether the verification prompt intervention provides additive benefit on top of a stronger model, or whether higher-capability models make explicit verification instructions redundant by self-correcting more reliably.

Architectural Extensions

A single-agent loop is fundamentally limited because one model must simultaneously localize the bug, plan the edit, execute it, and verify the result. A Manager-Worker-Critic architecture would decompose these responsibilities: a Manager agent reads the issue and identifies which files to change; a Worker agent makes the targeted edit; and a Critic agent independently verifies the patch by running tests and checking for regressions before submission. This separation of concerns would directly address the over-deletion problem, since the Critic could catch destructive edits before they are submitted.

Limitations

A sample of 50 tasks limits statistical power; observed differences of 2 to 3 tasks may not be significant. The simultaneous change of prompt and step limit prevents clean causal attribution. Multiple failed intermediate runs, especially the TDD pilot, consumed API budget and limited the scope of experiments within a fixed cost constraint. Finally, all experiments draw from the first 50 tasks of the test split, which skews toward Astropy and Django; results may not generalize across the full repository distribution in SWE-bench Verified.

Acknowledgments

Author Contributions

A. Sheoran designed and conducted all experiments, performed error analysis, and wrote the manuscript.

Funding

This work was performed as part of coursework for CIS 7000 Agentic AI at the University of Pennsylvania.

Conflicts of Interest

The author declares that there is no conflict of interest regarding the publication of this article.

Data Availability

Experiment logs and error analysis data are available from the author upon request.

A Full Error Analysis Logs

Tables 3 and 4 list all 27 unresolved tasks for the baseline and simplified intervention respectively, with error message, technical sub-type, and behavioral category for each.

Table 3: Full baseline error log — all 27 unresolved tasks.

Case ID	**Final Error**	**Technical Sub-type**	**Agentic Category**
astropy-13977	NameError: name 'exc' not defined	Coding error	Tool & Execution
astropy-14182	ValueError: could not convert string	Missing subgoal	Planning & Decomposition
astropy-14365	ValueError: could not convert 'no'	Misreads feedback	Verification & Recovery
astropy-14508	VerifyError: Card not FITS standard	Misunderstands constraints	Goal & Constraint
astropy-14539	SyntaxError: continue not in loop	Syntax error	Tool & Execution
astropy-14598	SyntaxError: invalid syntax	Editing error	Tool & Execution
astropy-7671	ImportError: cannot import name	Over-deletion	Tool & Execution
astropy-8707	PytestRemovedIn8Warning (fatal)	Environment/library issue	Environment & Infra
astropy-8872	DeprecationWarning: distutils	Misreads feedback	Verification & Recovery
django-10097	AttributeError: no 'validate_email'	Over-deletion	Tool & Execution
django-10554	AttributeError: no 'SQLInsertCompiler'	Over-deletion	Tool & Execution
django-10880	AssertionError: COUNT(*) not found	Logic regression	Verification & Recovery
django-10999	AssertionError: timedelta mismatch	Logic regression	Verification & Recovery
django-11087	DoesNotExist: query result missing	Unintended side effects	Planning & Decomposition
django-11095	AssertionError: [] != (class,)	Edge case failure	Verification & Recovery
django-11138	SyntaxError: duplicate code block	Copy-paste error	Tool & Execution
django-11141	UnboundLocalError: 'module'	Coding error	Tool & Execution
django-11149	UnboundLocalError: 'instances'	Coding error	Tool & Execution
django-11179	IntegrityError: invalid FK	Flawed framework model	Planning & Decomposition
django-11206	AssertionError: '0.' != '0'	Logic regression	Verification & Recovery
django-11211	AssertionError: None != [...]	Logic error	Verification & Recovery
django-11239	AssertionError: SSL tuples differ	Misunderstands interface	Goal & Constraint
django-11265	AssertionError: Col instance error	Destructive refactor	Tool & Execution
django-11276	SyntaxError: unexpected character	Syntax error	Tool & Execution
django-11299	AttributeError: no 'chain' on Query	Over-deletion	Tool & Execution
django-11333	AssertionError: URLResolver identity	Logic regression	Verification & Recovery
django-11433	AssertionError: None != date(...)	Edge case failure	Verification & Recovery

Table 4: Full intervention error log — all 27 unresolved tasks (simplified protocol).

Case ID	**Final Error**	**Technical Sub-type**	**Agentic Category**
astropy-13236	IndentationError: unexpected indent	Syntax error	Tool & Execution
astropy-13398	TypeError: unsupported operand	Coding error	Tool & Execution
astropy-13453	AttributeError: _validate_write	Editing error	Tool & Execution
astropy-13579	(Build failed)	Missing subgoal	Planning & Decomposition
astropy-14182	IndexError: list index out of range	Coding error	Tool & Execution
astropy-14369	IndexError: grammar rule index	Coding error	Tool & Execution
astropy-14508	ImportError: Card not found	Over-deletion	Tool & Execution
astropy-14598	AssertionError: escaped quote logic	Misreads feedback	Verification & Recovery
astropy-14995	AssertionError: Arrays not equal	Logic regression	Verification & Recovery
astropy-7606	IndentationError: expected indent	Syntax error	Tool & Execution
astropy-8872	IndentationError: unexpected indent	Syntax error	Tool & Execution
django-10097	IndentationError: unexpected indent	Syntax error	Tool & Execution
django-10880	AssertionError: COUNT(*) whitespace	Logic regression	Verification & Recovery
django-10914	AssertionError: 420 != 511	Logic regression	Verification & Recovery
django-10999	FAIL: test_negative	Coding error	Tool & Execution
django-11087	ERROR: S.DoesNotExist	Over-deletion	Tool & Execution
django-11133	NameError: 'HttpResponse' missing	Over-deletion	Tool & Execution
django-11138	FAIL: query_convert_tz	Logic regression	Verification & Recovery
django-11141	AttributeError: add_nodes missing	Over-deletion	Tool & Execution
django-11179	AttributeError: del_batches missing	Over-deletion	Tool & Execution
django-11206	AssertionError: '0.0' mismatch	Logic regression	Verification & Recovery
django-11211	IndentationError: expected indent	Syntax error	Tool & Execution
django-11239	AssertionError: SSL tuples differ	Coding error	Tool & Execution
django-11265	FieldError: cannot resolve alias	Logic regression	Verification & Recovery
django-11276	NameError: _js_escapes missing	Over-deletion	Tool & Execution
django-11333	ImportError: ns_resolver deleted	Over-deletion	Tool & Execution
django-11433	IndentationError: expected indent	Syntax error	Tool & Execution